ISPC
INSTITUTO SUPERIOR POLITÉCNICO CÓRDOBA

TECNICATURA SUPERIOR EN

Telecomunicaciones

PROYECTO INTEGRADOR I

Capa de Análisis

Manual de la restful API

Dirección General de EDUCACIÓN TÉCNICA Y FORMACIÓN PROFESIONAL

Ministerio de EDUCACIÓN

CÓRDOBA Seguimos haciendo

ÍNDICE

- GET /usuarios/
- GET /usuarios/int

- POST /usuarios/
- PUT /usuarios/int

- DELETE /usuarios/int

5. Ejemplos de Aplicación

Prólogo

Introducción

Este documento describe el proyecto de una API RESTful diseñada para gestionar datos de un sistema IoT (Internet de las Cosas). La API está construida utilizando el framework Flask en Python y se integra con una base de datos MySQL. El objetivo del proyecto es proporcionar una interfaz robusta y escalable para la administración de datos críticos en un entorno IoT, incluyendo la gestión de usuarios, dispositivos, proyectos, configuraciones y datos de dispositivos.

Implementación

El proyecto está implementado con una arquitectura modular que utiliza Blueprints de Flask. Esta estructura facilita la separación de responsabilidades y el mantenimiento del código, permitiendo que cada módulo maneje diferentes aspectos del sistema IoT de manera independiente. Las rutas (endpoints) están diseñadas para realizar operaciones CRUD (Create, Read, Update, Delete), proporcionando una manera eficiente y estructurada de interactuar con la base de datos.

Estructura del Proyecto

El proyecto está organizado en la siguiente estructura de directorios:

```
c_capa_de_analisis/
├── app.py
├── db_config.py
├── models.py
├── routes/
│   ├── configuraciones.py
│   ├── datos_dispositivos.py
│   ├── dispositivos.py
│   ├── proyectos.py
│   ├── seguridad.py
│   └── usuarios.py
├── requirements.txt
└── Dockerfile
```

Componentes del Proyecto

1. **app.py**: Archivo principal que inicia la aplicación Flask y registra los Blueprints para las rutas.
2. **db_config.py**: Configuración de la conexión a la base de datos.
3. **models.py**: Definición de los modelos de datos utilizando SQLAlchemy.
4. **routes/**: Directorio que contiene los Blueprints para las diferentes rutas de la API.
    - **configuraciones.py**: Rutas para la gestión de configuraciones.
    - **datos_dispositivos.py**: Rutas para la gestión de datos de dispositivos.
    - **dispositivos.py**: Rutas para la gestión de dispositivos.
    - **proyectos.py**: Rutas para la gestión de proyectos.
    - **seguridad.py**: Rutas para la gestión de seguridad.
    - **usuarios.py**: Rutas para la gestión de usuarios.
5. **requirements.txt**: Archivo de dependencias del proyecto.
6. **Dockerfile**: Archivo de configuración para la creación del contenedor Docker.

Detalle de Rutas

Las rutas están diseñadas para soportar operaciones CRUD, permitiendo crear, leer, actualizar y eliminar registros en la base de datos. A continuación, se detalla cada una de las rutas del proyecto.

Rutas

1. Configuraciones

Archivo: configuraciones.py
Descripción: Este módulo maneja las operaciones CRUD para las configuraciones de los dispositivos IoT.

- **GET /configuraciones/**: Obtiene todas las configuraciones.
- **GET /configuraciones/int:id**: Obtiene una configuración específica por ID.
- **POST /configuraciones/**: Crea una nueva configuración.
- **PUT /configuraciones/int:id**: Actualiza una configuración existente por ID.
- **DELETE /configuraciones/int:id**: Elimina una configuración por ID.

2. Datos Dispositivos

Archivo: datos_dispositivos.py
Descripción: Este módulo maneja las operaciones CRUD para los datos recolectados por los dispositivos IoT.

- **GET /datos_dispositivos/**: Obtiene todos los datos de dispositivos.
- **GET /datos_dispositivos/int:id**: Obtiene un dato específico por ID.
- **POST /datos_dispositivos/**: Crea un nuevo dato de dispositivo.
- **PUT /datos_dispositivos/int:id**: Actualiza un dato de dispositivo existente por ID.
- **DELETE /datos_dispositivos/int:id**: Elimina un dato de dispositivo por ID.

3. Dispositivos

Archivo: dispositivos.py
Descripción: Este módulo maneja las operaciones CRUD para los dispositivos IoT.

- **GET /dispositivos/**: Obtiene todos los dispositivos.
- **GET /dispositivos/int:id**: Obtiene un dispositivo específico por ID.
- **POST /dispositivos/**: Crea un nuevo dispositivo.
- **PUT /dispositivos/int:id**: Actualiza un dispositivo existente por ID.
- **DELETE /dispositivos/int:id**: Elimina un dispositivo por ID.

4. Proyectos

Archivo: proyectos.py
Descripción: Este módulo maneja las operaciones CRUD para los proyectos de IoT.

- **GET /proyectos/**: Obtiene todos los proyectos.
- **GET /proyectos/int:id**: Obtiene un proyecto específico por ID.
- **POST /proyectos/**: Crea un nuevo proyecto.
- **PUT /proyectos/int:id**: Actualiza un proyecto existente por ID.
- **DELETE /proyectos/int:id**: Elimina un proyecto por ID.

5. Seguridad

Archivo: seguridad.py
Descripción: Este módulo maneja las operaciones CRUD para la gestión de seguridad.

- **GET /seguridad/**: Obtiene todos los registros de seguridad.
- **GET /seguridad/int:id**: Obtiene un registro de seguridad específico por ID.
- **POST /seguridad/**: Crea un nuevo registro de seguridad.
- **PUT /seguridad/int:id**: Actualiza un registro de seguridad existente por ID.
- **DELETE /seguridad/int:id**: Elimina un registro de seguridad por ID.

6. Usuarios

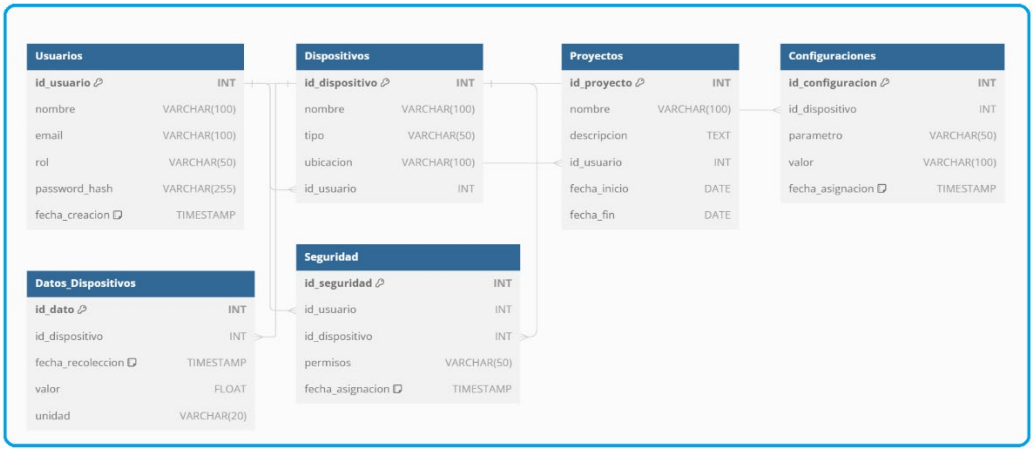
Archivo: usuarios.py
Descripción: Este módulo maneja las operaciones CRUD para los usuarios del sistema IoT.

- **GET /usuarios/**: Obtiene todos los usuarios.
- **GET /usuarios/int:id**: Obtiene un usuario específico por ID.
- **POST /usuarios/**: Crea un nuevo usuario.
- **PUT /usuarios/int:id**: Actualiza un usuario existente por ID.
- **DELETE /usuarios/int:id**: Elimina un usuario por ID.

Capítulo 1: Configuraciones

Introducción a la Base de Datos

La base de datos para el proyecto IoT se ha diseñado con varias tablas clave que gestionan diferentes aspectos del sistema IoT. Las tablas principales incluyen Usuarios, Dispositivos, Proyectos, Configuraciones, Datos_Dispositivos, y Seguridad. Cada tabla tiene una estructura específica y relaciones entre ellas para garantizar la integridad de los datos y facilitar las operaciones CRUD.



Este diagrama muestra la estructura de la base de datos, destacando las relaciones entre las tablas. Por ejemplo, un Usuario puede tener varios Dispositivos, y cada Dispositivo puede tener múltiples Configuraciones y Datos_Dispositivos. La tabla Seguridad gestiona los permisos de los usuarios sobre los dispositivos.

Metodología para el Desarrollo del Proyecto

El desarrollo del proyecto sigue una metodología ágil con iteraciones incrementales. La arquitectura se basa en el patrón de diseño RESTful, que permite una comunicación eficiente y escalable entre el cliente y el servidor. Las siguientes prácticas y conceptos clave se utilizan en el desarrollo del proyecto:

1. **Modularidad**: Separación del código en módulos independientes utilizando Blueprints de Flask.
2. **Autenticación**: Uso de claves API para restringir el acceso a la API.
3. **Persistencia de Datos**: Uso de MySQL como base de datos relacional para almacenar y gestionar los datos.
4. **Contenedores**: Uso de Docker para contenerizar la aplicación y facilitar el despliegue.

5. **CRUD**: Implementación de operaciones Create, Read, Update y Delete para cada entidad en la base de datos.

Desarrollo RESTful

El diseño RESTful de la API garantiza que las operaciones sobre las entidades de la base de datos se realicen de manera estándar y eficiente. Cada entidad tiene su propio conjunto de endpoints para gestionar las operaciones CRUD. Los endpoints siguen una estructura coherente que facilita la comprensión y el uso de la API.

Ruta: Configuraciones

El módulo configuraciones.py maneja las operaciones CRUD para las configuraciones de los dispositivos IoT. Estas configuraciones permiten ajustar y personalizar el comportamiento de los dispositivos.

Endpoints

- **GET /configuraciones/:** Obtiene todas las configuraciones.
- **GET /configuraciones/int:id:** Obtiene una configuración específica por ID.
- **POST /configuraciones/:** Crea una nueva configuración.
- **PUT /configuraciones/int:id:** Actualiza una configuración existente por ID.
- **DELETE /configuraciones/int:id:** Elimina una configuración por ID.

Explicación del Código

Archivo: configuraciones.py

```
from flask import Blueprint, request, jsonify
from db_config import get_db_connection

configuraciones_bp = Blueprint('configuraciones', __name__)

@configuraciones_bp.route('/', methods=['GET'])
def get_configuraciones():
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("SELECT * FROM Configuraciones")
    configuraciones = cursor.fetchall()
    cursor.close()
    return jsonify(configuraciones)

@configuraciones_bp.route('/<int:id>', methods=['GET'])
```

```
    def get_configuracion(id):
        conn = get_db_connection()
        cursor = conn.cursor(dictionary=True)
        cursor.execute("SELECT * FROM Configuraciones WHERE
id_configuracion = %s", (id,))
        configuracion = cursor.fetchone()
        cursor.close()
        return jsonify(configuracion)

    @configuraciones_bp.route('/', methods=['POST'])
    def add_configuracion():
        nueva_configuracion = request.json
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute(
            "INSERT INTO Configuraciones (id_dispositivo, parametro, valor)
VALUES (%s, %s, %s)",
            (nueva_configuracion['id_dispositivo'], nueva_configuracion['parametro'],
nueva_configuracion['valor'])
        )
        conn.commit()
        cursor.close()
        return '', 201

    @configuraciones_bp.route('/<int:id>', methods=['PUT'])
    def update_configuracion(id):
        configuracion_actualizada = request.json
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute(
            "UPDATE Configuraciones SET id_dispositivo = %s, parametro = %s,
valor = %s WHERE id_configuracion = %s",
            (configuracion_actualizada['id_dispositivo'],
configuracion_actualizada['parametro'], configuracion_actualizada['valor'], id)
        )
        conn.commit()
        cursor.close()
        return '', 204

    @configuraciones_bp.route('/<int:id>', methods=['DELETE'])
```

```
def delete_configuracion(id):
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("DELETE FROM Configuraciones WHERE id_configuracion
= %s", (id,))
    conn.commit()
    cursor.close()
    return '', 204
```

Descripción de los Endpoints

1. **GET /configuraciones/**

Este endpoint obtiene todas las configuraciones almacenadas en la base de datos. La función get_configuraciones() establece una conexión a la base de datos, ejecuta una consulta para seleccionar todas las configuraciones y devuelve el resultado en formato JSON.

Ejemplo de Uso:

curl -H "X-API-KEY: jade" https://api.gonaiot.com/jade/configuraciones/

2. **GET /configuraciones/int:id**

Este endpoint obtiene una configuración específica por su ID. La función get_configuracion(id) toma un ID como parámetro, ejecuta una consulta para seleccionar la configuración correspondiente y devuelve el resultado en formato JSON.

Ejemplo de Uso:

curl -H "X-API-KEY: jade" https://api.gonaiot.com/jade/configuraciones/1

3. **POST /configuraciones/**

Este endpoint crea una nueva configuración. La función add_configuracion() toma los datos de la nueva configuración del cuerpo de la solicitud, los inserta en la base de datos y devuelve un estado HTTP 201 si la operación es exitosa.

Ejemplo de Uso:

curl -H "Content-Type: application/json" -H "X-API-KEY: jade" -X POST -d '{"id_dispositivo": 1, "parametro": "volumen", "valor": "50"}' https://api.gonaiot.com/jade/configuraciones/

4. **PUT /configuraciones/int:id**

Este endpoint actualiza una configuración existente por su ID. La función update_configuracion(id) toma los datos actualizados del cuerpo de la solicitud y el ID de la configuración a actualizar, y ejecuta una consulta de actualización en la base de datos.

Ejemplo de Uso:

curl -H "Content-Type: application/json" -H "X-API-KEY: jade" -X PUT -d '{"id_dispositivo": 1, "parametro": "volumen", "valor": "60"}' https://api.gonaiot.com/jade/configuraciones/1

5. **DELETE /configuraciones/int:id**

Este endpoint elimina una configuración por su ID. La función delete_configuracion(id) toma el ID de la configuración a eliminar y ejecuta una consulta de eliminación en la base de datos.

Ejemplo de Uso:

curl -H "X-API-KEY: jade" -X DELETE https://api.gonaiot.com/jade/configuraciones/1

Capítulo 2: Datos_Dispositivos

Introducción

En el Capítulo 1, cubrimos la configuración de dispositivos IoT. Ahora, pasamos a los datos que estos dispositivos recopilan. La tabla Datos_Dispositivos en nuestra base de datos almacena información recolectada por los dispositivos, como valores de sensores y otros datos relevantes.

La función principal de esta tabla es registrar los datos recopilados por los dispositivos IoT en tiempo real. Estos datos son cruciales para el análisis y la toma de decisiones en sistemas IoT.

Estructura de la Tabla Datos_Dispositivos

La tabla Datos_Dispositivos se define de la siguiente manera:

- **id_dato**: Identificador único para cada registro de dato.
- **id_dispositivo**: Identificador del dispositivo que recopiló el dato (clave foránea).
- **fecha_recoleccion**: Fecha y hora en que se recopiló el dato.
- **valor**: Valor numérico del dato recopilado.
- **unidad**: Unidad de medida del dato (por ejemplo, Celsius, metros, etc.).

Esta estructura permite una relación directa entre los dispositivos y los datos que recopilan, asegurando integridad referencial y una gestión eficiente de la información.

Desarrollo de la API RESTful para Datos_Dispositivos

A continuación, se presentan los endpoints y el código correspondiente para gestionar los datos de dispositivos.

Endpoints de Datos_Dispositivos

1. **GET /datos_dispositivos/**: Obtiene todos los datos de dispositivos.
2. **GET /datos_dispositivos/{id}**: Obtiene un dato específico por ID.
3. **POST /datos_dispositivos/**: Crea un nuevo registro de dato.
4. **PUT /datos_dispositivos/{id}**: Actualiza un registro de dato existente por ID.
5. **DELETE /datos_dispositivos/{id}**: Elimina un registro de dato por ID.

Código de la Ruta Datos_Dispositivos

A continuación, se detalla el código del archivo datos_dispositivos.py que implementa los endpoints mencionados.

```
from flask import Blueprint, request, jsonify
from db_config import get_db_connection

datos_dispositivos_bp = Blueprint('datos_dispositivos', __name__)

@datos_dispositivos_bp.route('/', methods=['GET'])
def get_datos_dispositivos():
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("SELECT * FROM Datos_Dispositivos")
    datos_dispositivos = cursor.fetchall()
    cursor.close()
    return jsonify(datos_dispositivos)

@datos_dispositivos_bp.route('/<int:id>', methods=['GET'])
def get_dato_dispositivo(id):
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("SELECT * FROM Datos_Dispositivos WHERE id_dato =
%s", (id,))
    dato_dispositivo = cursor.fetchone()
    cursor.close()
    return jsonify(dato_dispositivo)

@datos_dispositivos_bp.route('/', methods=['POST'])
def add_dato_dispositivo():
    nuevo_dato = request.json
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute(
        "INSERT INTO Datos_Dispositivos (id_dispositivo, fecha_recoleccion,
valor, unidad) VALUES (%s, %s, %s, %s)",
        (nuevo_dato['id_dispositivo'], nuevo_dato['fecha_recoleccion'],
nuevo_dato['valor'], nuevo_dato['unidad'])
    )
    conn.commit()
    cursor.close()
    return '', 201

@datos_dispositivos_bp.route('/<int:id>', methods=['PUT'])
```

```
def update_dato_dispositivo(id):
    dato_actualizado = request.json
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute(
        "UPDATE Datos_Dispositivos SET id_dispositivo = %s, fecha_recoleccion
= %s, valor = %s, unidad = %s WHERE id_dato = %s",
        (dato_actualizado['id_dispositivo'],  dato_actualizado['fecha_recoleccion'],
dato_actualizado['valor'], dato_actualizado['unidad'], id)
    )
    conn.commit()
    cursor.close()
    return '', 204

@datos_dispositivos_bp.route('/<int:id>', methods=['DELETE'])
def delete_dato_dispositivo(id):
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("DELETE FROM Datos_Dispositivos WHERE id_dato = %s",
(id,))
    conn.commit()
    cursor.close()
    return '', 204
```

Descripción de los Endpoints y Ejemplos

1. Obtener Todos los Datos de Dispositivos

- **Método**: GET
- **Endpoint**: /datos_dispositivos/
- **Descripción**: Obtiene una lista de todos los datos de dispositivos registrados en la base de datos.
- **Ejemplo de Uso**:
  curl -H "X-API-KEY: jade" https://api.gonaiot.com/jade/datos_dispositivos/

2. Obtener un Dato de Dispositivo por ID

- **Método**: GET
- **Endpoint**: /datos_dispositivos/{id}
- **Descripción**: Obtiene un registro de dato específico utilizando su ID.
- **Ejemplo de Uso**:

curl -H "X-API-KEY: jade" https://api.gonaiot.com/jade/datos_dispositivos/1

3. Crear un Nuevo Registro de Dato

- **Método**: POST
- **Endpoint**: /datos_dispositivos/
- **Descripción**: Crea un nuevo registro de dato en la base de datos.
- **Ejemplo de Uso**:

curl -H "Content-Type: application/json" -H "X-API-KEY: jade" -X POST -d '{"id_dispositivo": 1, "fecha_recoleccion": "2024-06-24 14:30:00", "valor": 23.5, "unidad": "Celsius"}' https://api.gonaiot.com/jade/datos_dispositivos/

4. Actualizar un Registro de Dato

- **Método**: PUT
- **Endpoint**: /datos_dispositivos/{id}
- **Descripción**: Actualiza un registro de dato existente.
- **Ejemplo de Uso**:

curl -H "Content-Type: application/json" -H "X-API-KEY: jade" -X PUT -d '{"id_dispositivo": 1, "fecha_recoleccion": "2024-06-24 14:30:00", "valor": 25.0, "unidad": "Celsius"}' https://api.gonaiot.com/jade/datos_dispositivos/1

5. Eliminar un Registro de Dato

- **Método**: DELETE
- **Endpoint**: /datos_dispositivos/{id}
- **Descripción**: Elimina un registro de dato de la base de datos.
- **Ejemplo de Uso**:

curl -H "X-API-KEY: jade" -X DELETE https://api.gonaiot.com/jade/datos_dispositivos/1

Capítulo 3: Dispositivos

Introducción a la Tabla Dispositivos

La tabla **Dispositivos** en nuestra base de datos IoT es crucial para gestionar los dispositivos IoT que están conectados a nuestra red. Cada dispositivo puede tener varios sensores y actuadores que recolectan datos y realizan acciones basadas en las configuraciones recibidas. La estructura de esta tabla está diseñada para registrar la información esencial de cada dispositivo, permitiendo una administración eficiente y escalable.

Estructura de la Tabla Dispositivos

- **id_dispositivo (INT)**: Identificador único del dispositivo.
- **nombre (VARCHAR(100))**: Nombre del dispositivo.
- **tipo (VARCHAR(50))**: Tipo de dispositivo (sensor, actuador, etc.).
- **ubicacion (VARCHAR(100))**: Ubicación física del dispositivo.
- **id_usuario (INT)**: Identificador del usuario al que está asignado el dispositivo, referenciado a la tabla Usuarios.

Descripción de la Ruta Dispositivos

La ruta **Dispositivos** maneja las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para los dispositivos IoT. Este módulo se encuentra en routes/dispositivos.py y proporciona las siguientes funcionalidades:

- **GET /dispositivos/**: Obtiene todos los dispositivos registrados.
- **GET /dispositivos/**
  : Obtiene la información de un dispositivo específico por ID.
- **POST /dispositivos/**: Crea un nuevo dispositivo.
- **PUT /dispositivos/**
  : Actualiza la información de un dispositivo existente por ID.
- **DELETE /dispositivos/**
  : Elimina un dispositivo por ID.

Ejemplo de Implementación de la Ruta Dispositivos

Vamos a detallar la implementación de cada una de las operaciones CRUD en la ruta **Dispositivos**.

GET /dispositivos/

Este endpoint devuelve una lista de todos los dispositivos registrados en la base de datos.

```
@dispositivos_bp.route('/', methods=['GET'])
def get_dispositivos():
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("SELECT * FROM Dispositivos")
    dispositivos = cursor.fetchall()
    cursor.close()
    return jsonify(dispositivos)
```

GET /dispositivos/

Este endpoint devuelve la información de un dispositivo específico identificado por su ID.

```
@dispositivos_bp.route('/<int:id>', methods=['GET'])
def get_dispositivo(id):
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("SELECT * FROM Dispositivos WHERE id_dispositivo = %s",
(id,))
    dispositivo = cursor.fetchone()
    cursor.close()
    return jsonify(dispositivo)
```

POST /dispositivos/

Este endpoint permite la creación de un nuevo dispositivo.

```
@dispositivos_bp.route('/', methods=['POST'])
def add_dispositivo():
    nuevo_dispositivo = request.json
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute(
        "INSERT INTO Dispositivos (nombre, tipo, ubicacion, id_usuario) VALUES
(%s, %s, %s, %s)",
        (nuevo_dispositivo['nombre'], nuevo_dispositivo['tipo'],
nuevo_dispositivo['ubicacion'], nuevo_dispositivo['id_usuario'])
```

```
    )
    conn.commit()
    cursor.close()
    return '', 201
```

PUT /dispositivos/

Este endpoint permite actualizar la información de un dispositivo existente.

```
@dispositivos_bp.route('/<int:id>', methods=['PUT'])
def update_dispositivo(id):
    dispositivo_actualizado = request.json
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute(
        "UPDATE Dispositivos SET nombre = %s, tipo = %s, ubicacion = %s, id_usuario = %s WHERE id_dispositivo = %s",
        (dispositivo_actualizado['nombre'], dispositivo_actualizado['tipo'], dispositivo_actualizado['ubicacion'], dispositivo_actualizado['id_usuario'], id)
    )
    conn.commit()
    cursor.close()
    return '', 204
```

DELETE /dispositivos/

Este endpoint permite eliminar un dispositivo de la base de datos.

```
@dispositivos_bp.route('/<int:id>', methods=['DELETE'])
def delete_dispositivo(id):
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("DELETE FROM Dispositivos WHERE id_dispositivo = %s", (id,))
    conn.commit()
    cursor.close()
    return '', 204
```

Uso de la API

Para interactuar con la API y gestionar los dispositivos, se utilizan las siguientes solicitudes HTTP. Aquí algunos ejemplos de cómo utilizar los diferentes endpoints.

Obtener Todos los Dispositivos

curl -H "X-API-KEY: jade" https://api.gonaiot.com/jade/dispositivos/

Obtener un Dispositivo por ID

curl -H "X-API-KEY: jade" https://api.gonaiot.com/jade/dispositivos/1

Crear un Nuevo Dispositivo

curl -H "Content-Type: application/json" -H "X-API-KEY: jade" -X POST -d '{"nombre": "Sensor de Temperatura", "tipo": "sensor", "ubicacion": "Oficina", "id_usuario": 1}' https://api.gonaiot.com/jade/dispositivos/

Actualizar un Dispositivo Existente

curl -H "Content-Type: application/json" -H "X-API-KEY: jade" -X PUT -d '{"nombre": "Sensor de Humedad", "tipo": "sensor", "ubicacion": "Laboratorio", "id_usuario": 1}' https://api.gonaiot.com/jade/dispositivos/1

Eliminar un Dispositivo

curl -H "X-API-KEY: jade" -X DELETE https://api.gonaiot.com/jade/dispositivos/1

Capítulo 4: Proyectos

Introducción a la Tabla Proyectos

La tabla **Proyectos** es esencial para la gestión de los proyectos dentro del entorno IoT. Esta tabla permite organizar y categorizar los diferentes proyectos en los que los dispositivos IoT están involucrados. La estructura de esta tabla está diseñada para almacenar información detallada sobre cada proyecto, facilitando el seguimiento y la gestión de las iniciativas IoT.

Estructura de la Tabla Proyectos

- **id_proyecto (INT)**: Identificador único del proyecto.
- **nombre (VARCHAR(100))**: Nombre del proyecto.
- **descripcion (TEXT)**: Descripción detallada del proyecto.
- **id_usuario (INT)**: Identificador del usuario responsable del proyecto, referenciado a la tabla Usuarios.
- **fecha_inicio (DATE)**: Fecha de inicio del proyecto.
- **fecha_fin (DATE)**: Fecha de finalización del proyecto.

Descripción de la Ruta Proyectos

La ruta **Proyectos** maneja las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para los proyectos. Este módulo se encuentra en routes/proyectos.py y proporciona las siguientes funcionalidades:

- **GET /proyectos/**: Obtiene todos los proyectos registrados.
- **GET /proyectos/**
  : Obtiene la información de un proyecto específico por ID.
- **POST /proyectos/**: Crea un nuevo proyecto.
- **PUT /proyectos/**
  : Actualiza la información de un proyecto existente por ID.
- **DELETE /proyectos/**
  : Elimina un proyecto por ID.

Ejemplo de Implementación de la Ruta Proyectos

Vamos a detallar la implementación de cada una de las operaciones CRUD en la ruta **Proyectos**.

GET /proyectos/

Este endpoint devuelve una lista de todos los proyectos registrados en la base de datos.

```
@proyectos_bp.route('/', methods=['GET'])
def get_proyectos():
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("SELECT * FROM Proyectos")
    proyectos = cursor.fetchall()
    cursor.close()
    return jsonify(proyectos)
```

GET /proyectos/

Este endpoint devuelve la información de un proyecto específico identificado por su ID.

```
@proyectos_bp.route('/<int:id>', methods=['GET'])
def get_proyecto(id):
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("SELECT * FROM Proyectos WHERE id_proyecto = %s", (id,))
    proyecto = cursor.fetchone()
    cursor.close()
    return jsonify(proyecto)
```

POST /proyectos/

Este endpoint permite la creación de un nuevo proyecto.

```
@proyectos_bp.route('/', methods=['POST'])
def add_proyecto():
    nuevo_proyecto = request.json
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute(
        "INSERT INTO Proyectos (nombre, descripcion, id_usuario, fecha_inicio, fecha_fin) VALUES (%s, %s, %s, %s, %s)",
```

```
            (nuevo_proyecto['nombre'], nuevo_proyecto['descripcion'],
nuevo_proyecto['id_usuario'], nuevo_proyecto['fecha_inicio'],
nuevo_proyecto['fecha_fin'])
            )
            conn.commit()
            cursor.close()
            return '', 201
```

PUT /proyectos/

Este endpoint permite actualizar la información de un proyecto existente.

```
@proyectos_bp.route('/<int:id>', methods=['PUT'])
def update_proyecto(id):
    proyecto_actualizado = request.json
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute(
        "UPDATE Proyectos SET nombre = %s, descripcion = %s, id_usuario =
%s, fecha_inicio = %s, fecha_fin = %s WHERE id_proyecto = %s",
        (proyecto_actualizado['nombre'], proyecto_actualizado['descripcion'],
proyecto_actualizado['id_usuario'], proyecto_actualizado['fecha_inicio'],
proyecto_actualizado['fecha_fin'], id)
    )
    conn.commit()
    cursor.close()
    return '', 204
```

DELETE /proyectos/

Este endpoint permite eliminar un proyecto de la base de datos.

```
@proyectos_bp.route('/<int:id>', methods=['DELETE'])
def delete_proyecto(id):
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("DELETE FROM Proyectos WHERE id_proyecto = %s", (id,))
    conn.commit()
    cursor.close()
    return '', 204
```

Uso de la API

Para interactuar con la API y gestionar los proyectos, se utilizan las siguientes solicitudes HTTP. Aquí algunos ejemplos de cómo utilizar los diferentes endpoints.

Obtener Todos los Proyectos

curl -H "X-API-KEY: jade" https://api.gonaiot.com/jade/proyectos/

Obtener un Proyecto por ID

curl -H "X-API-KEY: jade" https://api.gonaiot.com/jade/proyectos/1

Crear un Nuevo Proyecto

curl -H "Content-Type: application/json" -H "X-API-KEY: jade" -X POST -d '{"nombre": "Proyecto IoT", "descripcion": "Monitorización de temperatura y humedad", "id_usuario": 1, "fecha_inicio": "2024-06-01", "fecha_fin": "2024-12-31"}' https://api.gonaiot.com/jade/proyectos/

Actualizar un Proyecto Existente

curl -H "Content-Type: application/json" -H "X-API-KEY: jade" -X PUT -d '{"nombre": "Proyecto IoT Actualizado", "descripcion": "Actualización de monitorización", "id_usuario": 1, "fecha_inicio": "2024-06-01", "fecha_fin": "2024-12-31"}' https://api.gonaiot.com/jade/proyectos/1

Eliminar un Proyecto

curl -H "X-API-KEY: jade" -X DELETE https://api.gonaiot.com/jade/proyectos/1

Capítulo 5: Seguridad

Introducción a la Tabla Seguridad

La tabla **Seguridad** es crucial para la gestión de permisos y accesos dentro del entorno IoT. Esta tabla permite definir y controlar qué usuarios tienen acceso a qué dispositivos y con qué permisos. Esto asegura que solo usuarios autorizados puedan interactuar con los dispositivos, manteniendo la integridad y seguridad del sistema.

Estructura de la Tabla Seguridad

- **id_seguridad (INT)**: Identificador único del registro de seguridad.
- **id_usuario (INT)**: Identificador del usuario, referenciado a la tabla Usuarios.
- **id_dispositivo (INT)**: Identificador del dispositivo, referenciado a la tabla Dispositivos.
- **permisos (VARCHAR(50))**: Detalles de los permisos otorgados (por ejemplo, lectura, escritura, administración).
- **fecha_asignacion (TIMESTAMP)**: Fecha y hora en que se asignaron los permisos.

Descripción de la Ruta Seguridad

La ruta **Seguridad** maneja las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para la gestión de permisos de seguridad. Este módulo se encuentra en routes/seguridad.py y proporciona las siguientes funcionalidades:

- **GET /seguridad/**: Obtiene todos los registros de seguridad.
- **GET /seguridad/**
  : Obtiene un registro de seguridad específico por ID.
- **POST /seguridad/**: Crea un nuevo registro de seguridad.
- **PUT /seguridad/**
  : Actualiza un registro de seguridad existente por ID.
- **DELETE /seguridad/**
  : Elimina un registro de seguridad por ID.

Ejemplo de Implementación de la Ruta Seguridad

Vamos a detallar la implementación de cada una de las operaciones CRUD en la ruta **Seguridad**.

GET /seguridad/

Este endpoint devuelve una lista de todos los registros de seguridad en la base de datos.

```
@seguridad_bp.route('/', methods=['GET'])
def get_seguridad():
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("SELECT * FROM Seguridad")
    seguridad = cursor.fetchall()
    cursor.close()
    return jsonify(seguridad)
```

GET /seguridad/

Este endpoint devuelve la información de un registro de seguridad específico identificado por su ID.

```
@seguridad_bp.route('/<int:id>', methods=['GET'])
def get_seguridad_entry(id):
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("SELECT * FROM Seguridad WHERE id_seguridad = %s", (id,))
    seguridad_entry = cursor.fetchone()
    cursor.close()
    return jsonify(seguridad_entry)
```

POST /seguridad/

Este endpoint permite la creación de un nuevo registro de seguridad.

```
@seguridad_bp.route('/', methods=['POST'])
def add_seguridad():
    nueva_seguridad = request.json
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute(
        "INSERT INTO Seguridad (id_usuario, id_dispositivo, permisos) VALUES (%s, %s, %s)",
        (nueva_seguridad['id_usuario'], nueva_seguridad['id_dispositivo'], nueva_seguridad['permisos'])
```

```
    )
    conn.commit()
    cursor.close()
    return '', 201
```

PUT /seguridad/

Este endpoint permite actualizar la información de un registro de seguridad existente.

```
@seguridad_bp.route('/<int:id>', methods=['PUT'])
def update_seguridad(id):
    seguridad_actualizada = request.json
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute(
        "UPDATE Seguridad SET id_usuario = %s, id_dispositivo = %s, permisos = %s WHERE id_seguridad = %s",
        (seguridad_actualizada['id_usuario'], seguridad_actualizada['id_dispositivo'], seguridad_actualizada['permisos'], id)
    )
    conn.commit()
    cursor.close()
    return '', 204
```

DELETE /seguridad/

Este endpoint permite eliminar un registro de seguridad de la base de datos.

```
@seguridad_bp.route('/<int:id>', methods=['DELETE'])
def delete_seguridad(id):
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("DELETE FROM Seguridad WHERE id_seguridad = %s", (id,))
    conn.commit()
    cursor.close()
    return '', 204
```

Uso de la API

Para interactuar con la API y gestionar los registros de seguridad, se utilizan las siguientes solicitudes HTTP. Aquí algunos ejemplos de cómo utilizar los diferentes endpoints.

Obtener Todos los Registros de Seguridad

curl -H "X-API-KEY: jade" https://api.gonaiot.com/jade/seguridad/

Obtener un Registro de Seguridad por ID

curl -H "X-API-KEY: jade" https://api.gonaiot.com/jade/seguridad/1

Crear un Nuevo Registro de Seguridad

curl -H "Content-Type: application/json" -H "X-API-KEY: jade" -X POST -d '{"id_usuario": 1, "id_dispositivo": 1, "permisos": "lectura"}' https://api.gonaiot.com/jade/seguridad/

Actualizar un Registro de Seguridad Existente

curl -H "Content-Type: application/json" -H "X-API-KEY: jade" -X PUT -d '{"id_usuario": 1, "id_dispositivo": 1, "permisos": "escritura"}' https://api.gonaiot.com/jade/seguridad/1

Eliminar un Registro de Seguridad

curl -H "X-API-KEY: jade" -X DELETE https://api.gonaiot.com/jade/seguridad/1

Capítulo 6: Usuarios

Introducción a la Tabla Usuarios

La tabla **Usuarios** es esencial en el proyecto IoT para gestionar la información de los usuarios que interactúan con el sistema. Esta tabla almacena datos clave que permiten autenticar y autorizar a los usuarios, garantizando que solo aquellos con los permisos adecuados puedan acceder y modificar los datos y configuraciones de los dispositivos IoT.

Estructura de la Tabla Usuarios

- **id_usuario (INT)**: Identificador único del usuario.
- **nombre (VARCHAR(100))**: Nombre del usuario.
- **email (VARCHAR(100))**: Correo electrónico del usuario.
- **rol (VARCHAR(50))**: Rol del usuario dentro del sistema (por ejemplo, administrador, usuario estándar).
- **password_hash (VARCHAR(255))**: Hash de la contraseña del usuario para asegurar el almacenamiento seguro de las contraseñas.
- **fecha_creacion (TIMESTAMP)**: Fecha y hora en que el usuario fue creado.

Descripción de la Ruta Usuarios

La ruta **Usuarios** maneja las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para la gestión de usuarios en el sistema IoT. Este módulo se encuentra en routes/usuarios.py y proporciona las siguientes funcionalidades:

- **GET /usuarios/**: Obtiene todos los usuarios registrados.
- **GET /usuarios/**
  : Obtiene un usuario específico por ID.
- **POST /usuarios/**: Crea un nuevo usuario.
- **PUT /usuarios/**
  : Actualiza un usuario existente por ID.
- **DELETE /usuarios/**
  : Elimina un usuario por ID.

Ejemplo de Implementación de la Ruta Usuarios

Vamos a detallar la implementación de cada una de las operaciones CRUD en la ruta **Usuarios**.

GET /usuarios/

Este endpoint devuelve una lista de todos los usuarios registrados en la base de datos.

```
@usuarios_bp.route('/', methods=['GET'])
def get_usuarios():
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("SELECT * FROM Usuarios")
    usuarios = cursor.fetchall()
    cursor.close()
    return jsonify(usuarios)
```

GET /usuarios/

Este endpoint devuelve la información de un usuario específico identificado por su ID.

```
@usuarios_bp.route('/<int:id>', methods=['GET'])
def get_usuario(id):
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("SELECT * FROM Usuarios WHERE id_usuario = %s", (id,))
    usuario = cursor.fetchone()
    cursor.close()
    return jsonify(usuario)
```

POST /usuarios/

Este endpoint permite la creación de un nuevo usuario.

```
@usuarios_bp.route('/', methods=['POST'])
def add_usuario():
    nuevo_usuario = request.json
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute(
        "INSERT INTO Usuarios (nombre, email, rol, password_hash) VALUES (%s, %s, %s, %s)",
        (nuevo_usuario['nombre'], nuevo_usuario['email'], nuevo_usuario['rol'], nuevo_usuario['password_hash'])
    )
```

```
    conn.commit()
    cursor.close()
    return '', 201
```

PUT /usuarios/

Este endpoint permite actualizar la información de un usuario existente.

```
@usuarios_bp.route('/<int:id>', methods=['PUT'])
def update_usuario(id):
    usuario_actualizado = request.json
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute(
        "UPDATE Usuarios SET nombre = %s, email = %s, rol = %s, password_hash = %s WHERE id_usuario = %s",
        (usuario_actualizado['nombre'], usuario_actualizado['email'], usuario_actualizado['rol'], usuario_actualizado['password_hash'], id)
    )
    conn.commit()
    cursor.close()
    return '', 204
```

DELETE /usuarios/

Este endpoint permite eliminar un usuario de la base de datos.

```
@usuarios_bp.route('/<int:id>', methods=['DELETE'])
def delete_usuario(id):
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("DELETE FROM Usuarios WHERE id_usuario = %s", (id,))
    conn.commit()
    cursor.close()
    return '', 204
```

Uso de la API

Para interactuar con la API y gestionar los usuarios, se utilizan las siguientes solicitudes HTTP. Aquí algunos ejemplos de cómo utilizar los diferentes endpoints.

Obtener Todos los Usuarios

curl -H "X-API-KEY: jade" https://api.gonaiot.com/jade/usuarios/

Obtener un Usuario por ID

curl -H "X-API-KEY: jade" https://api.gonaiot.com/jade/usuarios/1

Crear un Nuevo Usuario

curl -H "Content-Type: application/json" -H "X-API-KEY: jade" -X POST -d '{"nombre": "Juan Perez", "email": "juan.perez@example.com", "rol": "usuario", "password_hash": "hashed_password"}' https://api.gonaiot.com/jade/usuarios/

Actualizar un Usuario Existente

curl -H "Content-Type: application/json" -H "X-API-KEY: jade" -X PUT -d '{"nombre": "Juan Perez", "email": "juan.perez@example.com", "rol": "admin", "password_hash": "new_hashed_password"}' https://api.gonaiot.com/jade/usuarios/1

Eliminar un Usuario

curl -H "X-API-KEY: jade" -X DELETE https://api.gonaiot.com/jade/usuarios/1

Referencias Bibliográficas

- Manuales y cuadernillos digitales de Ciencias de la Computación

https://program.ar/material-didactico/

- Unidad 1. Fundamentos de Informática SOPORTE LÓGICO EN UN ORDENADOR PERSONAL: EL SOFTWARE

https://www3.gobiernodecanarias.org/medusa/ecoblog/mgoncal/