Documentación de API FastAPI para Dispositivos IoT

Diseño de Sistemas Embebidos

LOGO UAT

Universidad Autónoma de Tamaulipas

Facultad de Ingeniería

Equipo:

Nombre del Estudiante 1
Nombre del Estudiante 2
Nombre del Estudiante 3
Nombre del Estudiante 4

14 de mayo de 2025

Índice general

Capítulo 1

Introducción

1.1. Descripción General

Esta documentación describe una aplicación FastAPI para gestionar dispositivos IoT y sus registros. La API proporciona operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para varios tipos de dispositivos y sus datos.

FastAPI es un framework moderno y de alto rendimiento para construir APIs con Python 3.7+. Se basa en estándares abiertos como OpenAPI y JSON Schema, y proporciona características como validación automática, serialización, documentación interactiva y mucho más.

1.2. Características Principales

La API de Dispositivos IoT ofrece las siguientes características principales:

- Gestión completa de dispositivos IoT con operaciones CRUD
- Registro y seguimiento de datos de dispositivos
- Soporte para diferentes tipos de dispositivos (luces, controladores de voltaje, etc.)
- Documentación interactiva con Swagger UI y ReDoc
- Validación automática de datos con Pydantic
- Persistencia de datos con SQLAlchemy

Capítulo 2

Instalación y Configuración

2.1. Requisitos Previos

Para ejecutar la API, necesitarás:

- Python 3.7 o superior
- pip (gestor de paquetes de Python)
- Acceso a una base de datos (SQLite por defecto)

2.2. Instalación de Dependencias

Instala las dependencias requeridas con el siguiente comando:

```
pip install fastapi uvicorn sqlalchemy pydantic python-dotenv
```

2.3. Configuración de la Base de Datos

La aplicación utiliza SQLAlchemy como ORM (Object-Relational Mapping) para interactuar con la base de datos. Por defecto, se configura para usar SQLite, pero puede configurarse para usar otras bases de datos.

Configura tu conexión a la base de datos en el archivo `.env`:

```
DATABASE_URL=sqlite:///./iot_devices.db
```

Para usar otras bases de datos, modifica la URL según el formato requerido por SQLAlchemy.

2.4. Ejecución de la API

Para ejecutar la API, utiliza el siguiente comando desde el directorio `server`:

```
uvicorn main:app --reload
```

Esto iniciará el servidor en `http://localhost:8000`. La opción `--reload` permite que el servidor se reinicie automáticamente cuando detecta cambios en el código.

Capítulo 3

Arquitectura de la Aplicación

3.1. Estructura del Proyecto

La aplicación sigue una estructura modular con los siguientes componentes principales:

- `main.py`: Punto de entrada de la aplicación y definición de endpoints
- `models.py`: Definición de modelos de datos (SQLAlchemy y Pydantic)
- `database.py`: Configuración de la conexión a la base de datos
- `crud.py`: Operaciones CRUD para interactuar con la base de datos

3.2. Flujo de Datos

El flujo de datos en la aplicación sigue este patrón:

1. El cliente envía una solicitud HTTP a un endpoint específico
2. FastAPI valida la solicitud utilizando los modelos Pydantic
3. El controlador (función en `main.py`) procesa la solicitud
4. Se realizan operaciones CRUD en la base de datos según sea necesario
5. Se devuelve una respuesta al cliente, validada por los modelos Pydantic

Capítulo 4

Modelos de Datos

4.1. Modelos SQLAlchemy

Los modelos SQLAlchemy definen la estructura de las tablas en la base de datos:

4.1.1. DevicesInfoDB

Almacena información básica sobre los dispositivos:

```
class DevicesInfoDB(Base):
    __tablename__ = "DevicesInfo"
    id_device = Column(Integer, primary_key=True, index=True, nullable=
    False, unique=True, autoincrement=True)
    id_type = Column(Numeric, primary_key=False, index=False, nullable=
    False, unique=True, autoincrement=False)
    id_signal_type = Column(Numeric, primary_key=False, index=False,
    nullable=False, unique=True, autoincrement=False)
    nombre = Column(String, unique=False, index=True, nullable=False)
    vendor = Column(String, unique=False, index=True, nullable=False)
```

4.1.2. DevicesRecordsDB

Almacena registros de datos de los dispositivos:

```
class DevicesRecordsDB(Base):
    __tablename__ = "DevicesRecords"
    id_record = Column(Integer, primary_key=True, index=True, nullable=
    False, unique=True, autoincrement=True)
    id_device = Column(Numeric, primary_key=False, index=False, nullable=
    False, unique=False, autoincrement=False)
    current_value = Column(Numeric, index=False, unique=False, nullable=
    False)
    date_record = Column(Date, index=False, unique=False, nullable=False)
```

4.1.3. TomaDecisionesDB

Almacena decisiones basadas en datos:

```python
class TomaDecisionesDB(Base):
    __tablename__ = "TomaDecisiones"
    id_decision = Column(Integer, primary_key=True, index=True, nullable=False, unique=True, autoincrement=True)
    velocidad = Column(Numeric, index=False, unique=False, nullable=False)
    decision = Column(Numeric, index=False, unique=False, nullable=False)
    date_record = Column(Date, index=False, unique=False, nullable=False)
```

4.1.4. LucesDB

Almacena información específica de dispositivos de iluminación:

```python
class LucesDB(Base):
    __tablename__ = "Luces"
    id_device = Column(Integer, primary_key=True, index=True, nullable=False, unique=True, autoincrement=True)
    lumens = Column(Numeric, index=False, unique=False, nullable=False)
    nombre = Column(String, unique=False, index=True, nullable=False)
    vendor = Column(String, unique=False, index=True, nullable=False)
```

4.1.5. ControladorVoltajeDB

Almacena información específica de controladores de voltaje:

```python
class ControladorVoltajeDB(Base):
    __tablename__ = "ControladorVoltaje"
    id_device = Column(Integer, primary_key=True, index=True, nullable=False, unique=True, autoincrement=True)
    encendido = Column(Integer, index=False, unique=False, nullable=False)
     # Usa Integer para booleano
    voltaje = Column(Numeric, index=False, unique=False, nullable=False)
    nombre = Column(String, unique=False, index=True, nullable=False)
    vendor = Column(String, unique=False, index=True, nullable=False)
```

4.2. Modelos Pydantic

Los modelos Pydantic se utilizan para validar datos de entrada y salida en la API:

4.2.1. DevicesInfo y DevicesInfoResponse

```python
class DevicesInfo(BaseModel):
    id_device: int
    id_type: int
    id_signal_type: int
    nombre: str
    vendor: str

class DevicesInfoResponse(BaseModel):
    id_device: int
```

```
    id_type: float
    id_signal_type: float
    nombre: str
    vendor: str

    class Config:
        from_attributes = True
```

4.2.2. DevicesRecords y DevicesRecordsResponse

```
class DevicesRecords(BaseModel):
    id_record: int
    id_device: int
    current_value: float
    date_record: datetime

class DevicesRecordsResponse(BaseModel):
    id_record: int
    id_device: float
    current_value: float
    date_record: datetime

    class Config:
        from_attributes = True
```

4.2.3. TomaDecisiones y TomaDecisionesResponse

```
class TomaDecisiones(BaseModel):
    id_decision: int
    velocidad: float
    decision: float
    date_record: datetime

class TomaDecisionesResponse(BaseModel):
    id_decision: int
    velocidad: float
    decision: float
    date_record: datetime

    class Config:
        from_attributes = True
```

4.2.4. Luces y LucesResponse

```
class Luces(BaseModel):
    lumens: float
    id_device: int
    nombre: str
    vendor: str
```

```python

class LucesResponse(BaseModel):
    id_device: int
    lumens: float
    nombre: str
    vendor: str

    class Config:
        from_attributes = True
```

4.2.5. ControladorVoltaje y ControladorVoltajeResponse

```python
class ControladorVoltaje(BaseModel):
    encendido: bool
    voltaje: float
    id_device: int
    nombre: str
    vendor: str

class ControladorVoltajeResponse(BaseModel):
    id_device: int
    encendido: bool
    voltaje: float
    nombre: str
    vendor: str

    class Config:
        from_attributes = True
```

Capítulo 5

Endpoints de la API

5.1. Endpoint Raíz

- **GET /**
- **Descripción:** Devuelve un mensaje de bienvenida
- **Respuesta:** {`"message": "Bienvenido a la API de Dispositivos IoT"`}

5.2. Endpoints de Test

5.2.1. GET /tests/

- **Descripción:** Obtiene todos los tests
- **Respuesta:** Lista de objetos TestModel

5.2.2. GET /tests/{test_id}

- **Descripción:** Obtiene un test por ID
- **Parámetros:** test_id (entero)
- **Respuesta:** Objeto TestModel

5.2.3. POST /tests/?title={title}

- **Descripción:** Crea un nuevo test
- **Parámetros:** title (cadena)
- **Respuesta:** Objeto TestModel creado

5.2.4. DELETE /tests/{test_id}

- **Descripción:** Elimina un test
- **Parámetros:** test_id (entero)
- **Respuesta:** Mensaje de confirmación

5.3. Endpoints de DevicesInfo

5.3.1. GET /devices/info/

- **Descripción:** Obtiene información de todos los dispositivos
- **Respuesta:** Lista de objetos DevicesInfoResponse

5.3.2. GET /devices/info/{id_device}

- **Descripción:** Obtiene información de un dispositivo por ID
- **Parámetros:** id_device (entero)
- **Respuesta:** Lista de objetos DevicesInfoResponse

5.3.3. POST /devices/info/

- **Descripción:** Crea información de un nuevo dispositivo
- **Cuerpo:** Objeto DevicesInfo
- **Respuesta:** Objeto DevicesInfoResponse creado

5.3.4. PUT /devices/info/{id_device}

- **Descripción:** Actualiza información de un dispositivo
- **Parámetros:** id_device (entero)
- **Cuerpo:** Objeto DevicesInfo
- **Respuesta:** Objeto DevicesInfoResponse actualizado

5.3.5. DELETE /devices/info/{id_device}

- **Descripción:** Elimina información de un dispositivo
- **Parámetros:** id_device (entero)
- **Respuesta:** Mensaje de confirmación

5.4. Endpoints de DevicesRecords

5.4.1. GET /devices/records/

- **Descripción:** Obtiene todos los registros de dispositivos
- **Respuesta:** Lista de objetos DevicesRecordsResponse

5.4.2. GET /devices/records/{id_record}

- **Descripción:** Obtiene un registro de dispositivo por ID
- **Parámetros:** id_record (entero)
- **Respuesta:** Objeto DevicesRecordsResponse

5.4.3. GET /devices/records/device/{id_device}

- **Descripción:** Obtiene registros de dispositivo por ID de dispositivo
- **Parámetros:** id_device (entero)
- **Respuesta:** Lista de objetos DevicesRecordsResponse

5.4.4. POST /devices/records/

- **Descripción:** Crea un nuevo registro de dispositivo
- **Cuerpo:** Objeto DevicesRecords
- **Respuesta:** Objeto DevicesRecordsResponse creado

5.4.5. PUT /devices/records/{id_record}

- **Descripción:** Actualiza un registro de dispositivo
- **Parámetros:** id_record (entero)
- **Cuerpo:** Objeto DevicesRecords
- **Respuesta:** Objeto DevicesRecordsResponse actualizado

5.4.6. DELETE /devices/records/{id_record}

- **Descripción:** Elimina un registro de dispositivo
- **Parámetros:** id_record (entero)
- **Respuesta:** Mensaje de confirmación

5.5. Endpoints de TomaDecisiones

5.5.1. GET /decisiones/

- **Descripción:** Obtiene todas las decisiones
- **Respuesta:** Lista de objetos TomaDecisionesResponse

5.5.2. GET /decisiones/{id_decision}

- **Descripción:** Obtiene una decisión por ID
- **Parámetros:** id_decision (entero)
- **Respuesta:** Objeto TomaDecisionesResponse

5.5.3. POST /decisiones/

- **Descripción:** Crea una nueva decisión
- **Cuerpo:** Objeto TomaDecisiones
- **Respuesta:** Objeto TomaDecisionesResponse creado

5.5.4. PUT /decisiones/{id_decision}

- **Descripción:** Actualiza una decisión
- **Parámetros:** id_decision (entero)
- **Cuerpo:** Objeto TomaDecisiones
- **Respuesta:** Objeto TomaDecisionesResponse actualizado

5.5.5. DELETE /decisiones/{id_decision}

- **Descripción:** Elimina una decisión
- **Parámetros:** id_decision (entero)
- **Respuesta:** Mensaje de confirmación

5.6. Endpoints de Luces

5.6.1. GET /luces/

- **Descripción:** Obtiene todas las luces
- **Respuesta:** Lista de objetos LucesResponse

5.6.2. GET /luces/{id_device}

- **Descripción:** Obtiene una luz por ID
- **Parámetros:** id_device (entero)
- **Respuesta:** Objeto LucesResponse

5.6.3. POST /luces/

- **Descripción:** Crea una nueva luz
- **Cuerpo:** Objeto Luces
- **Respuesta:** Objeto LucesResponse creado

5.6.4. PUT /luces/{id_device}

- **Descripción:** Actualiza una luz
- **Parámetros:** id_device (entero)
- **Cuerpo:** Objeto Luces
- **Respuesta:** Objeto LucesResponse actualizado

5.6.5. DELETE /luces/{id_device}

- **Descripción:** Elimina una luz
- **Parámetros:** id_device (entero)
- **Respuesta:** Mensaje de confirmación

5.7. Endpoints de ControladorVoltaje

5.7.1. GET /controladores/

- **Descripción:** Obtiene todos los controladores de voltaje
- **Respuesta:** Lista de objetos ControladorVoltajeResponse

5.7.2. GET /controladores/{id_device}

- **Descripción:** Obtiene un controlador de voltaje por ID
- **Parámetros:** id_device (entero)
- **Respuesta:** Objeto ControladorVoltajeResponse

5.7.3. POST /controladores/

- **Descripción:** Crea un nuevo controlador de voltaje
- **Cuerpo:** Objeto ControladorVoltaje
- **Respuesta:** Objeto ControladorVoltajeResponse creado

5.7.4. PUT /controladores/{id_device}

- **Descripción:** Actualiza un controlador de voltaje
- **Parámetros:** id_device (entero)
- **Cuerpo:** Objeto ControladorVoltaje
- **Respuesta:** Objeto ControladorVoltajeResponse actualizado

5.7.5. DELETE /controladores/{id_device}

- **Descripción:** Elimina un controlador de voltaje
- **Parámetros:** id_device (entero)
- **Respuesta:** Mensaje de confirmación

Capítulo 6

Ejemplos de Uso

6.1. Ejemplos de Solicitudes HTTP

6.1.1. Obtener Información de Todos los Dispositivos

```
GET http://localhost:8000/devices/info/
Accept: application/json
```

6.1.2. Crear Información de un Nuevo Dispositivo

```
POST http://localhost:8000/devices/info/
Content-Type: application/json
Accept: application/json

{
  "id_device": 1,
  "id_type": 1,
  "id_signal_type": 1,
  "nombre": "Sensor de Temperatura",
  "vendor": "Acme Inc."
}
```

6.1.3. Actualizar Información de un Dispositivo

```
PUT http://localhost:8000/devices/info/1
Content-Type: application/json
Accept: application/json

{
  "id_device": 1,
  "id_type": 1,
  "id_signal_type": 1,
  "nombre": "Sensor de Temperatura Actualizado",
  "vendor": "Acme Inc."
}
```

6.1.4. Crear un Nuevo Registro de Dispositivo

```
POST http://localhost:8000/devices/records/
Content-Type: application/json
Accept: application/json

{
  "id_record": 1,
  "id_device": 1,
  "current_value": 25.5,
  "date_record": "2023-06-01T12:00:00"
}
```

6.1.5. Crear una Nueva Luz

```
POST http://localhost:8000/luces/
Content-Type: application/json
Accept: application/json

{
  "id_device": 1,
  "lumens": 800,
  "nombre": "Luz de Sala de Estar",
  "vendor": "Philips"
}
```

6.1.6. Crear un Nuevo Controlador de Voltaje

```
POST http://localhost:8000/controladores/
Content-Type: application/json
Accept: application/json

{
  "id_device": 1,
  "encendido": true,
  "voltaje": 220.0,
  "nombre": "Controlador de Energ a Principal",
  "vendor": "Siemens"
}
```

Capítulo 7

Documentación Interactiva

FastAPI genera automáticamente documentación interactiva para la API. Una vez que el servidor está en funcionamiento, puedes acceder a:

- **Swagger UI:** `http://localhost:8000/docs`
- **ReDoc:** `http://localhost:8000/redoc`

Estas interfaces te permiten:

- Explorar todos los endpoints disponibles
- Ver los esquemas de solicitud y respuesta
- Probar los endpoints directamente desde el navegador
- Entender los posibles códigos de estado y respuestas de error

Capítulo 8

Buenas Prácticas y Recomendaciones

8.1. Manejo de Errores

La API implementa manejo de errores estándar utilizando códigos de estado HTTP:

- **200 OK:** Solicitud exitosa
- **201 Created:** Recurso creado exitosamente
- **204 No Content:** Solicitud exitosa sin contenido de respuesta (usado para DELETE)
- **404 Not Found:** Recurso no encontrado
- **422 Unprocessable Entity:** Datos de solicitud inválidos

8.2. Seguridad

Para mejorar la seguridad de la API, considera implementar:

- Autenticación (OAuth2, JWT)
- Autorización basada en roles
- HTTPS para cifrar las comunicaciones
- Limitación de tasa para prevenir ataques de fuerza bruta
- Validación de entrada para prevenir inyecciones

8.3. Escalabilidad

Para mejorar la escalabilidad de la API, considera:

- Implementar caché para respuestas frecuentes

- Utilizar bases de datos más robustas para producción (PostgreSQL, MySQL)
- Configurar un balanceador de carga para distribuir el tráfico
- Implementar un sistema de colas para tareas asíncronas

Capítulo 9

Conclusiones

La API de Dispositivos IoT proporciona una solución completa para la gestión de dispositivos IoT y sus datos. Utilizando FastAPI, se ha creado una API moderna, rápida y fácil de usar que sigue las mejores prácticas de desarrollo.

Las principales ventajas de esta implementación incluyen:

- Rendimiento de alta velocidad gracias a FastAPI
- Validación automática de datos con Pydantic
- Documentación interactiva generada automáticamente
- Estructura modular y fácil de mantener
- Soporte para diferentes tipos de dispositivos IoT

Esta API puede servir como base para sistemas más complejos de IoT, integrándose con otras tecnologías como sistemas de análisis de datos, interfaces de usuario o servicios en la nube.