PROYECTO IOT

Desarrollo Aplicación

David Ramos Domingo
dramdom@alumno.upv.es

ÍNDICE

Introducción.

El desarrollo y uso de esta aplicación requieren tener en cuenta varios aspectos técnicos. La aplicación cuenta con una interfaz gráfica de usuario (GUI) creada utilizando la biblioteca **PyQt5**, por lo que es imprescindible instalar esta biblioteca para garantizar su correcto funcionamiento y visualización.

Si bien **Anaconda** incluye PyQt5 en su entorno de desarrollo integrado **Spyder**, permitiendo que la aplicación funcione de manera adecuada dentro de este entorno, será necesario instalar la biblioteca de forma manual si se desea ejecutar la aplicación fuera de Anaconda.

Además, para habilitar la funcionalidad de visualización en tiempo real de los localizadores, es necesario instalar el módulo adicional **PyQt5.QtWebEngineWidgets**. Este módulo permite la representación de mapas o gráficos que facilitan el monitoreo de las entidades geolocalizadas.

Ya existe una empresa española en Madrid que ofrece estos servicios con sus sensores GPS para ganado digitanimal.com

Caso de uso:

Este proyecto consiste en el desarrollo de una aplicación dirigida a un cliente ganadero con necesidades específicas relacionadas con la gestión y el monitoreo de sus animales.
El cliente enfrenta problemas recurrentes con animales que se alejan del grupo durante el pastoreo, lo que dificulta su localización. Por tanto, requiere una solución que permita geolocalizar a los animales en tiempo real y establecer un perímetro seguro para evitar que se pierdan.
Además, el cliente necesita un sistema que facilite el control e identificación de cada animal, almacenando información relevante como peso, fecha de nacimiento y estado de salud.

Requerimientos funcionales:

1. **RF1: Sistema de geofencing.**
   El sistema debe definir un perímetro geográfico basado en coordenadas que pueda ajustarse dinámicamente. Esto es esencial para delimitar el área de interés y detectar cuándo un animal se encuentra fuera del área establecida.

2. **RF2: Detección en tiempo real.**
   El sistema debe procesar actualizaciones de posición en tiempo real para identificar salidas del perímetro con una latencia máxima de 2 segundos. Este requisito es crucial para garantizar una respuesta inmediata ante posibles incidencias.

3. **RF3: Notificaciones automáticas.**
   Cuando un animal salga del perímetro, el sistema debe enviar notificaciones automáticas a un puerto específico (en este caso, el puerto 8181). Las notificaciones deben incluir datos como el ID del animal, la posición actual y la hora en la que ocurrió el evento.

Requerimientos no funcionales:

1. **RNF1: Escalabilidad.**
   El sistema debe ser capaz de manejar un gran número de entidades simultáneamente (animales) sin afectar negativamente al rendimiento. Esto asegura que el sistema sea útil incluso en escenarios con una alta densidad de dispositivos.
2. **RNF2: Seguridad de los datos.**
   La comunicación entre el sistema y las notificaciones debe emplear protocolos seguros, como HTTPS, para proteger los datos sensibles y evitar manipulaciones o intercepciones. Adicionalmente, se evaluarán y utilizarán modelos de comunicación ligeros como MQTT y CoAP, según las necesidades específicas del proyecto.

Modelo de datos de las entidades:
El sistema incluye cinco tipos de entidades principales, con un total de 15 instancias individuales, cada una representando un animal o dispositivo. Todas las entidades móviles son geolocalizables y tienen atributos de **latitud** y **longitud**, además de otros tres atributos: **fecha de nacimiento**, **peso** (enviado a través de MQTT) y **estado de salud** (reportado manualmente mediante CoAP).
El sistema está diseñado para ser **totalmente escalable**, permitiendo agregar más entidades, tipos de entidades o incluso más atributos sin afectar el rendimiento general. Esto asegura que, a medida que crezca la operación del cliente, el sistema podrá adaptarse fácilmente a nuevas necesidades.

CLASES	ENTIDADES	ATRIBUTOS				
Cabra	Cabra_1 Cabra_2 Cabra_3	Latitud	Longitud	Fecha Nacimiento	Peso	Salud
Cerdo	Cerdo_1 Cerdo_2 Cerdo_3	Latitud	Longitud	Fecha Nacimiento	Peso	Salud
Oveja	Oveja_1 Oveja_2 Oveja_3	Latitud	Longitud	Fecha Nacimiento	Peso	Salud
Vaca	Vaca_1 Vaca_2 Vaca_3	Latitud	Longitud	Fecha Nacimiento	Peso	Salud
Perro	Perro_1 Perro_2 Perro_3	Latitud	Longitud	Fecha Nacimiento	Peso	Salud

Programa en Python para la creación de entidades

La aplicación permite crear entidades de manera manual mediante el uso del método **POST**, lo que ofrece facilidad para incluir entidades, especificar su clase y añadir tantos atributos como se desee.
Además, la aplicación incorpora la funcionalidad de creación masiva de entidades a partir de un archivo en formato **JSON**. Este archivo debe contener la información necesaria de las entidades, como sus clases y atributos, para que puedan ser procesadas y creadas de forma automatizada en el sistema.
Esta funcionalidad es especialmente útil para gestionar grandes cantidades de datos y optimizar el tiempo de creación de entidades en proyectos más complejos.

Programa en Python para eliminar entidades.

La aplicación permite eliminar todas las entidades almacenadas en el sistema, ya sea de forma individual, por clase o en su totalidad. Esto lo realizará con el método DELETE. Si no se especifica nada en los campos, realizará una petición GET, recorrerá todas las entidades y las borrará. Antes de esto mandará un mensaje de advertencia y de confirmación.

Programa en Python que permita actualizar los atributos

La aplicación permite actualizar los valores de cualquier entidad desde la línea de comandos, seleccionando el identificador, el atributo y el nuevo valor. Los cambios en los atributos se realizan utilizando los métodos **POST**, **PATCH** y **PUT**:

- **POST**: Si el atributo ya existe en la entidad, se actualiza con el nuevo valor; si no existe, se crea un nuevo atributo con el valor proporcionado.

- **PATCH**: Solo actualiza el atributo si ya existe en la entidad, dejando los demás atributos intactos.
- **PUT**: Reemplaza todos los atributos de la entidad con los nuevos valores proporcionados, sobrescribiendo completamente los atributos anteriores.

Programa en Python que modifique automáticamente la posición de las entidades

La aplicación permite modificar automáticamente las posiciones de todas las entidades con atributos de latitud y longitud. Esto se logra mediante un script llamado **Sensores Movimiento**, que simula el movimiento de los animales. El proceso funciona de la siguiente manera:

1. **Petición GET**: Se realiza una solicitud GET a la base de datos de entidades, seleccionando aquellas que tengan los atributos de **latitud** y **longitud**. De esta manera, se identifican las entidades con coordenadas geográficas.
2. **Simulación del Movimiento**: A continuación, el script simula el movimiento de las entidades (por ejemplo, desplazando su posición de manera lineal para nuestro caso.
3. **Petición PATCH**: Con los nuevos valores de latitud y longitud generados, se realiza una solicitud PATCH para actualizar las posiciones de las entidades. Esta operación permite modificar solo la posición (latitud y longitud) sin afectar a los demás atributos de las entidades.

De esta forma, se simula el movimiento de los animales, actualizando sus posiciones en tiempo real según un patrón predeterminado o generado dinámicamente.

Aplicación en Python que permita mostrar gráficamente las diferentes entidades.

La aplicación, desarrollada en Python, utiliza las bibliotecas **Folium** y **PyQt5.QtWebEngineWidgets** para representar gráficamente, en un mapa y en tiempo real, la localización georreferenciada de las entidades (animales).

El proceso comienza con una solicitud **GET** para detectar todas las entidades que disponen de atributos de **latitud** y **longitud**. Estas clases se presentan en forma de casillas configurables, lo que permite al usuario seleccionar cuáles desea visualizar en el mapa.

El mapa permite establecer un punto central a través de coordenadas ingresadas por el usuario, además de incluir una barra de zoom para ajustar el nivel de detalle. Finalmente, la aplicación genera el mapa deseado y muestra las posiciones

actualizadas de las entidades, permitiendo un seguimiento dinámico de su movimiento.

Implementar un programa en Python que ejerza de agente IoT que permita que un atributo de una entidad se modifique de forma periódica o bajo demanda mediante MQTT.

La implementación incluye un programa en Python que actúa como un agente IoT y permite modificar de forma periódica o bajo demanda un atributo de una entidad, utilizando el protocolo **MQTT**. Para este propósito, es necesario desplegar un broker MQTT que gestione la comunicación entre los sensores y la aplicación.

En este caso, el cliente, al recoger el ganado, utiliza una pasarela para el pesaje de los animales. Los datos recopilados, como el peso, se envían mediante **MQTT**. La aplicación es capaz de suscribirse a un **topic** específico y recibir esta información en tiempo real, configurando adecuadamente el puerto y el topic correspondiente.

Para validar el funcionamiento, se incluye un script que simula sensores de peso, publicando valores en el broker MQTT. La aplicación procesa esta información y actualiza los atributos de peso de las entidades correspondientes, verificando y mostrando que la modificación se realiza correctamente.

Implementar un programa en Python que ejerza de agente IoT que permita modificar el atributo de una entidad mediante el uso de COAP.

La implementación incluye un programa en Python que actúa como un agente IoT, permitiendo modificar el atributo de una entidad utilizando el protocolo **CoAP**. Para ello, se ha desarrollado un servidor que escucha en el puerto **5683** y gestiona las peticiones recibidas a través de este protocolo.

Cuando la aplicación envía una solicitud **CoAP**, el servidor la procesa. En este caso, se ha configurado para actualizar el atributo relacionado con la **salud de los animales**. Tras recibir y validar la información enviada, el servidor comunica los datos al sistema **Orion**, donde se actualizan los atributos correspondientes de la entidad.

Este flujo asegura una integración fluida entre CoAP y la gestión de entidades, demostrando la capacidad del sistema para interactuar con protocolos ligeros de IoT y mantener los datos actualizados en tiempo real.

Implementar una notificación de manera que se cree una alerta cuando se produzca algún evento.

La aplicación incluye un sistema de alertas basado en eventos relacionados con las entidades. Permite al usuario crear un área geográfica delimitada mediante una caja de **5 coordenadas**, calculada automáticamente a partir de un punto central en función a un perímetro especificado por el usuario. Este sistema de geofencing es clave para controlar el movimiento de los animales.

Cuando un animal salga del perímetro definido, la aplicación genera una **notificación automática** que identifica claramente la entidad que ha traspasado los límites. Adicionalmente, la aplicación realiza una **suscripción automática con Orion**, lo que permite que, en caso de un evento como la salida del perímetro, Orion reciba las notificaciones correspondientes, actualizando en tiempo real el estado de las entidades.

Asimismo, la aplicación ofrece la opción de configurar una suscripción a un **puerto específico**, elegido por el usuario, donde enviará notificaciones adicionales en tiempo real. Estas notificaciones pueden redirigirse a un servidor externo para gestionar la información sobre las posiciones de los animales.

Este sistema asegura una supervisión eficaz, permite actuar rápidamente ante situaciones críticas y garantiza la integración con la infraestructura IoT a través de Orion.

Conclusiones del Proyecto IoT

El desarrollo de este proyecto ha permitido implementar una solución integral y escalable para la gestión y monitoreo de entidades geolocalizadas en un entorno ganadero. A lo largo del proceso, hemos cumplido con los objetivos definidos, logrando avances significativos en los siguientes aspectos:

1. **Diseño y Usabilidad de la Aplicación**
   La aplicación desarrollada combina funcionalidad y facilidad de uso, proporcionando una interfaz gráfica intuitiva creada con PyQt5 que simplifica la interacción del usuario con las diversas funciones del sistema. Además, la inclusión de herramientas como Folium para la visualización gráfica en tiempo real mejora la experiencia del usuario y facilita el monitoreo de las entidades.
2. **Gestión Dinámica de Entidades**
   - **Creación, Actualización y Eliminación**: Se implementaron funcionalidades para la gestión eficiente de las entidades, permitiendo la creación masiva, actualización dinámica de atributos y eliminación de datos de forma controlada. Estas características aseguran flexibilidad y adaptabilidad a las necesidades del cliente.

- **Escalabilidad**: El sistema está preparado para manejar grandes cantidades de entidades, demostrando ser apto para operaciones a mayor escala en el futuro.

3. **Integración con Protocolos IoT**
   La integración de protocolos ligeros como MQTT y CoAP ha demostrado la capacidad del sistema para interactuar con tecnologías IoT modernas:
   - **MQTT**: Utilizado para transmitir datos críticos, como el peso de los animales, a través de un sistema de comunicación fiable y en tiempo real.
   - **CoAP**: Permite la modificación de atributos, como el estado de salud, de manera eficiente y con un uso optimizado de los recursos.
4. **Simulación y Supervisión en Tiempo Real**
   - **Simulación de Movimiento**: Se implementaron algoritmos que permiten simular el movimiento de las entidades, actualizando en tiempo real sus posiciones en el sistema.
   - **Supervisión Dinámica**: La visualización georreferenciada de las entidades y la configuración de geofencing aseguran un monitoreo constante, detectando eventos críticos como la salida del perímetro de los animales.
5. **Notificaciones y Alertas Automatizadas**
   El sistema de alertas basado en eventos es un componente clave que ofrece:
   - Notificaciones automáticas en tiempo real, tanto en Orion como en un puerto configurado por el usuario, asegurando la supervisión y respuesta inmediata ante eventos críticos.
   - Flexibilidad para integrarse con otros sistemas externos, consolidando la interoperabilidad de la solución.
6. **Colaboración con Orion Context Broker**
   La suscripción automática a Orion refuerza la funcionalidad del sistema, garantizando que las actualizaciones de los atributos y las notificaciones estén sincronizadas con la infraestructura IoT.
7. **Relevancia y Aplicabilidad**
   Este proyecto aborda de manera eficaz las necesidades del cliente ganadero, ofreciendo una solución personalizada para la gestión de su ganado. Además, las características escalables y modulares del sistema lo convierten en una herramienta aplicable en otros entornos IoT con requisitos similares.

Reflexión Final

Este proyecto IoT ha mostrado cómo las nuevas tecnologías pueden usarse para resolver problemas prácticos, como gestionar el ganado de forma más eficiente. Al combinar protocolos IoT, herramientas de visualización en tiempo real y sistemas de notificación, hemos creado una solución sólida y fácil de ampliar, que puede servir de base para mejoras y avances futuros en agricultura inteligente y gestión de recursos.