Tecnicatura Superior en Innovación con Tecnologías 4.0

PROGRAMACIÓN II

Nombre Instituto: Instituto Superior Politécnico Córdoba
Nombre de la Tecnicatura: Tecnicatura Superior en Innovación con Tecnologías 4.0
Módulo: Programación II
Nombre del Trabajo: Proyecto Integrador
Nombre del Docente: Carlos Charletti
Nombre del Grupo: Ferreyra Santiago, Cesar Martins, Gaston Trejo, M. Victoria Urcola
Año: Cohorte 2023
Github

Proyecto P . i . N Pets Identification Node

"Paseos que Conectan, Creando Comunidad"

- Introducción

El proyecto P.i.N (Pets Identification Node) tiene como objetivo principal crear un dispositivo innovador que transforme cada mascota en un nodo de una red especial, brindando a los dueños la posibilidad de establecer conexiones sociales para sus mascotas y para sí mismo, para facilitar la interacción entre ellos y fomentar una comunidad sólida de apoyo; además de tener ubicación de sus mascotas en tiempo real. Este informe detalla el trabajo realizado durante el curso, así como los logros alcanzados y las proyecciones futuras del proyecto.

-Desarrollo

Este proyecto presenta un sistema IoT que simula la interacción entre varios dispositivos inteligentes, inspirado en el proyecto PiN (Pets Identification Node). PiN es un innovador dispositivo diseñado para ser colocado en el collar de las mascotas, integra varias funcionalidades tecnológicas que permiten a los dueños monitorear la actividad de sus animales en tiempo real y su la salud

Para diseñar un sistema IoT que simula la interacción entre varios dispositivos inteligentes y basado en el proyecto PiN (Pets Identification Node), podemos utilizar principios de programación orientada a objetos (POO). A continuación, se presenta un ejemplo básico de cómo se puede implementar este sistema en Python, utilizando clases para representar los dispositivos y sus interacciones, aplicando encapsulamiento, herencia y polimorfismo.

Paso 1: Definir las Clases Base

Primero, definimos una clase base `Dispositivo` que tendrá métodos comunes para todos los dispositivos.

```python
class Dispositivo:

    def __init__(self, id_dispositivo):   # Inicializa un nuevo dispositivo.

        self.id_dispositivo = id_dispositivo

    def leer_datos(self):
            pass

    def enviar_datos(self):
            pass

    def recibir_datos(self):
            pass
```

Paso 2: Crear Clases Heredadas

Luego, creamos clases heredadas para dispositivos específicos, como PiN y otros sensores.

La clase `PiN` hereda de `Dispositivo` y tiene métodos específicos para leer sensores de salud y actividad, así como para enviar datos.
Cada sensor hereda de `Dispositivo` y representa diferentes funcionalidades.

```python
class SensorGPS(Dispositivo):

    def __init__(self, id_dispositivo):
        super().__init__(id_dispositivo)
        self.ubicacion = {}

    def leer_datos(self):

        self.ubicacion = leer_gps()  # Lectura simulada
        print(f"Ubicación GPS leída: {self.ubicacion}")

    def enviar_datos(self):

        print(f"Enviando datos de ubicación GPS: {self.ubicacion}")

class SensorRitmoCardiaco(Dispositivo):

    def __init__(self, id_dispositivo):
        super().__init__(id_dispositivo)
        self.frecuencia_cardiaca = None

    def leer_datos(self):

        self.frecuencia_cardiaca = leer_ritmo_cardiaco()  # Lectura simulada
        print(f"Frecuencia cardíaca leída: {self.frecuencia_cardiaca} BPM")

    def enviar_datos(self):

        print(f"Enviando datos de frecuencia cardíaca: {self.frecuencia_cardiaca} BPM")

class SensorAcelerometro(Dispositivo):

    def __init__(self, id_dispositivo):
        super().__init__(id_dispositivo)
        self.aceleracion = {}

    def leer_datos(self):
```

```python
class PiN(Dispositivo):
    def __init__(self, id_dispositivo, nombre_mascota):
        super().__init__(id_dispositivo)
        self.nombre_mascota = nombre_mascota
        self.gps = None
        self.ritmo_cardiaco = None
        self.acelerometro = None
        self.temperatura_corporal = None
        self.proximidad = None

    def leer_datos(self):

        if self.gps:
            self.gps.leer_datos()
        if self.ritmo_cardiaco:
            self.ritmo_cardiaco.leer_datos()
        if self.acelerometro:
            self.acelerometro.leer_datos()
        if self.temperatura_corporal:
            self.temperatura_corporal.leer_datos()
        if self.proximidad:
            self.proximidad.leer_datos()

    def enviar_datos(self):
        if self.gps:
            self.gps.enviar_datos()
        if self.ritmo_cardiaco:
            self.ritmo_cardiaco.enviar_datos()
        if self.acelerometro:
            self.acelerometro.enviar_datos()
        if self.temperatura_corporal:
            self.temperatura_corporal.enviar_datos()
        if self.proximidad:
            self.proximidad.enviar_datos()
```

Paso 3: Implementar la Comunicación entre Dispositivos

Para simular la comunicación entre dispositivos, creamos una clase `RedIoT` que gestiona los dispositivos y facilita la comunicación.

```python
class RedIoT:

    def __init__(self):
        self.dispositivos = []

    def agregar_dispositivo(self, dispositivo):

        self.dispositivos.append(dispositivo)

    def leer_todos_los_datos(self):

        for dispositivo in self.dispositivos:
            dispositivo.leer_datos()

    def enviar_todos_los_datos(self):

        for dispositivo in self.dispositivos:
            dispositivo.enviar_datos()
```

Paso 4: Documentar el Código

Este código implementa una red IoT básica con dispositivos específicos y simula la lectura y el envío de datos. Los sensores están integrados en el dispositivo PiN y se gestionan a través de la clase `RedIoT`. Cada sensor tiene su propia clase con métodos para leer y enviar datos, utilizando principios de programación orientada a objetos como encapsulamiento, herencia y polimorfismo.

-Conclusión

Hemos implementado a través de python la simulación de las funcionalidades de nuestro dispositivo PiN, el cual como propuesta innovadora, conlleva la implementación de nuevas tecnologías y su integración como sistema IOT.-