Doge Power

Collar de Perro Inteligente

Introducción

Cada vez son más las personas que tienen un animal de compañía; en España, el porcentaje alcanza el 40% (según la Asociación Madrileña de Veterinarios de Animales de Compañía), de los cuales el 60% son de perros. A pesar de un porcentaje tan alto, el mundo de las mascotas y la tecnología se encuentran actualmente muy alejados, lo que deja un amplio campo de ideas para facilitar el cuidado de estos por parte de sus dueños. Es por esto por lo que hemos decidido desarrollar un collar que facilite el cuidado de estos gracias a funciones como localizador GPS, sistema para encontrarlo en la oscuridad y modo amaestramiento entre otros.

Para su desarrollo, hemos planteado mecanismos que a priori suponemos que serán factibles, tanto a nivel de conocimientos como económico, por lo que es posible que conforme avance el proyecto vayamos añadiendo y/o eliminando funcionalidades.

El producto va destinado principalmente a los dueños de perros de todas las razas, pero su uso también es adecuado para otras mascotas como gatos, hurones, caballos, etc., pero no descartamos que alguien quiera usarlo para, por ejemplo, una iguana.

Funcionalidades

- **Localizador**

  Mediante un módulo GPS, podremos obtener la ubicación de la mascota en tiempo real. Esta información se enviará constantemente al servidor, y el usuario podrá consultarlo para comprobar las coordenadas

- **Luz**

  El usuario podrá enviar una petición para encender un led y así poder encontrar a su perro en situaciones de poca visibilidad. Podremos modificar la intensidad y la intermitencia de este led para mejorar esta función.

- **Sensor de sonido**

  El collar implementará un micrófono que irá registrando los valores obtenidos por este, y que el usuario podrá ir consultando sin problema. Esta función es útil para detectar si el perro ha ladrado, para poder detectar, por ejemplo, si ha habido algún intruso.

Alcántara Laguna, Alejandro
Chamber González de Quevedo, Enrique

- **Sensor ahorcamiento**

  Mediante un sensor de presión se almacenará la presión que recibe el collar. Sus finalidades son dos: detectar una presión excesivamente alta para poder detectar si el perro se está ahorcando y detectar una presión excesivamente baja para detectar si el perro no lleva el collar (en caso de habérselo quitado alguien o él mismo).

- **Modo amaestramiento**

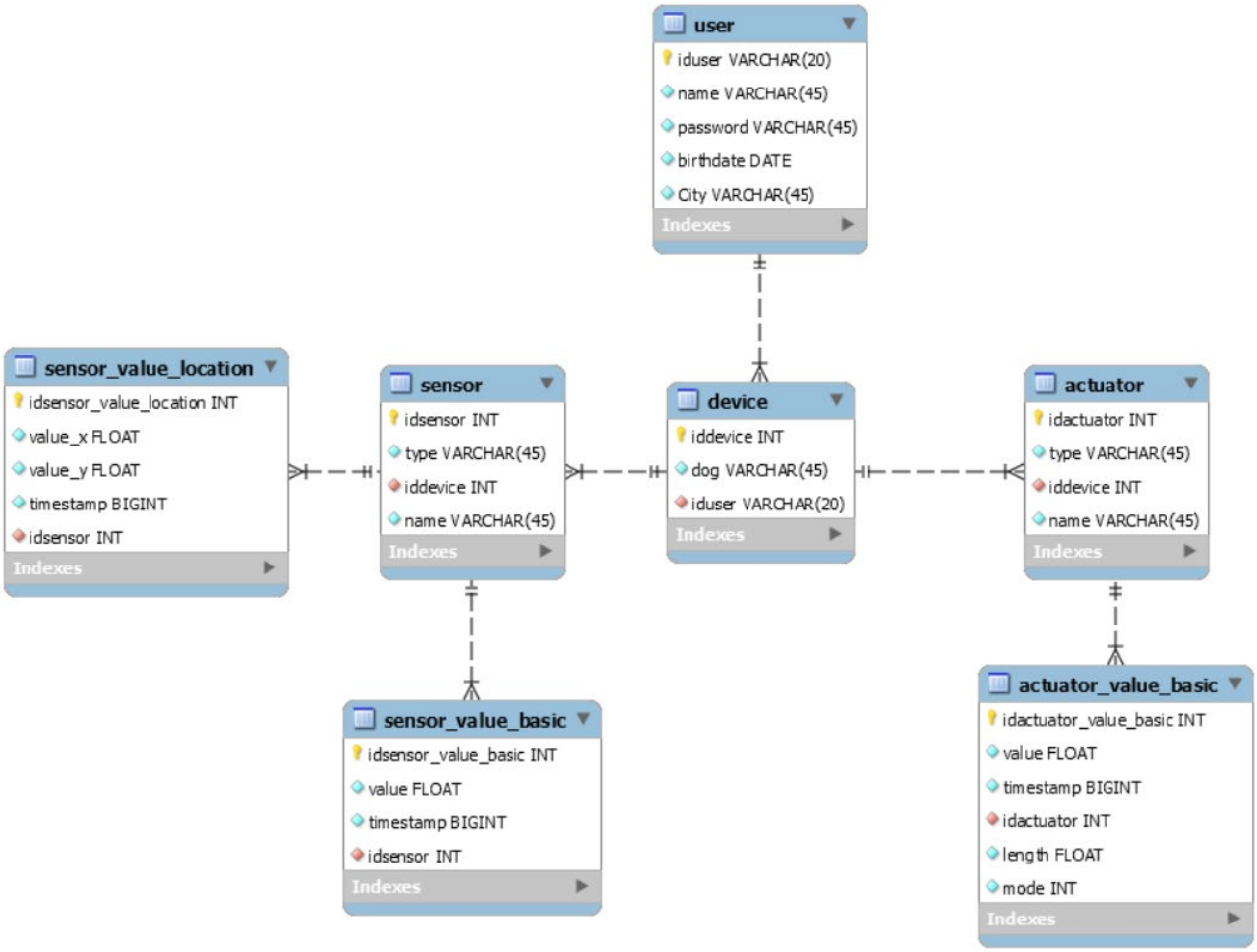
  El collar tendrá implementado un motor de vibración para funcionar como los collares de amaestramiento. Simplemente el usuario mandará una señal para que el collar vibre con distinta intensidad según la necesidad (limitado para no herir al perro) y así amaestrarlo.

- **App**

  En lugar de usar un bot de Telegram como interfaz de usuario, hemos implementado una aplicación para Android (mediante la herramienta MIT app inventor) para mayor comodidad. Desde esta podremos manejar todos los actuadores y leer los sensores, además de poder registrar nuevos y usuarios y dispositivos.

Base de datos

La base de datos diseñada para nuestro proyecto es relativamente simple, puesto que la única información que almacenaremos será la de los usuarios, dispositivos, sensores y actuadores. Para ello, hemos creado una tabla para cada uno, excepto para los dos últimos, que presentan tablas para la información del sensor/actuador, y por separado tablas para los valores. Estos, por regla general, solo precisan del id del sensor, el valor y el timestamp, pero luego existen otros distintos, como es por ejemplo la ubicación, que precisa de dos coordenadas, además de otras extras. El resultado es el siguiente diagrama UML:



Para casi todas las tablas hemos usado solo atributos básicos con el objetivo de obtener una base de datos simple y que no nos lleve una gran cantidad de tiempo trabajar con ella. Como hemos indicado anteriormente, para sensor y actuator tenemos atributos tales como el dispositivo al que pertenecen, el nombre del sensor/actuador (pressure, led, etc.) y el tipo. Con el tipo nos referimos a si es básico (sensor/actuador, valor y timestamp) o no, que en nuestro caso es únicamente el tipo Location, ya que estos sensores recogen el valor de las coordenadas en x e y.

API REST

En el diseño de la API REST principalmente hemos creado funciones básicas que nos conecten con la base de datos, ya que, tras analizar varias veces el proyecto, no hemos visto necesaria la creación de ninguna función especial. En resumen, nuestra API REST está formada por:

- **Métodos GET:**

  Hemos creado un método GET para cada tabla. Para evitar un código excesivo, hemos unificado la función para obtención de los valores de los

sensores (también para los actuadores) independientemente del tipo de sensor, puesto que, al haber tres tipos distintos, necesitamos tres consultas distintas, pero con un simple switch-case según el tipo, se ha solucionado. Además, tanto para los valores de sensores como para los de actuadores hemos añadido la opción de filtrar mediante timestamp, lo cual es realizable mediante la misma función, ya que esta comprueba si existe el parámetro de timestamp, y dependiendo del caso, la consulta difiere. También hemos creado una función que nos devuelve todos los dispositivos pertenecientes a un usuario dado. También tenemos una función que nos devuelve la información de los dispositivos de un usuario, otra que nos devuelve los IDs de los sensores y otra de los actuadores de un dispositivo.

- **Métodos POST:**

  Para insertar nuevos elementos hemos usado los métodos POST. Uno para cada tabla, excepto para device, actuator y sensor, ya que, creando un dispositivo, creamos automáticamente sus sensores y actuadores, ya que no tiene sentido que este uno se genere sin los otros. Además hemos implementado otra función para inicio de sesión.

- **Métodos PUT:**

  Hemos usado los métodos PUT para actualizar, por lo que solo han sido necesarios para actualizar los datos de usuario y del dispositivo.

- **Métodos DELETE:**

  Al igual que los PUT, solo son necesarios para eliminar usuarios y dispositivos, puesto que no tiene sentido eliminar sensores o actuadores solo, y el historial de valores no puede modificarse.

Para estas peticiones hemos usado URLs intuitivas y cortas. La nomenclatura es "/api/(tipo de elemento)/: (id)" independientemente del tipo de petición, puesto que esto no supone un problema, pero para algunas peticiones es distinta:

- Para obtener los dispositivos de un usuario, en '(tipo de elemento)' colocamos "devicesOf".
- Para obtener los sensores o actuadores de un usuario, en '(tipo de elemento)' colocamos "sensorsOf" o "actuatorsOf".
- Cuando tratamos con los valores de los sensores, la nomenclatura es: "/api/(sensor o actuator)/values/: (id)".
- Para los nuevos usuarios o dispositivos, en lugar de ': (id)' usamos "new", ya que los parámetros se los pasaremos todos en el cuerpo ("new" no sería necesario, puesto que sin él no tendríamos problemas tampoco, pero lo mantenemos para hacerlo más intuitivo).

Por último, en cuanto al cuerpo de las peticiones para los métodos POST y PUT, simplemente usamos un JSON para las columnas en la base de datos, aunque es necesario aclarar que el cuerpo debe contener todos los atributos, de lo contrario no será efectiva la petición. Esto provoca que debamos introducir también el id en el cuerpo, el cual será ignorado puesto que, en los POST, la BBDD es quien lo asigna, y en los PUT se toma el dato pasado por la URL.

MQTT

El uso de MQTT en nuestro proyecto es corto pero importante, ya que su uso es necesario para un sistema de actuadores eficiente. En nuestro caso, solo nos será necesario el uso de dos canales, uno por actuador, además siendo los dos similares en atributos, por lo que se simplifica aún más.

La estructura del servidor MQTT, cuya estructura es genérica para todos los proyectos en la totalidad del código prácticamente. La estructura de la que hablamos se trata de una clase con los métodos para iniciar e inicializar el servidor y para las acciones como clientes de conectarnos, desconectarnos, suscribirnos y darnos de baja. El único punto del que debemos fijarnos es en los canales, pues es aquí donde debemos especificar de qué canales haremos uso; en nuestro caso, estos son led y vibración.

ESP32

El código para el ESP32 sigue una estructura en la que la mayoría de los casos solo debemos adaptarlo a nuestro proyecto:

- **Setup**: Iniciamos la conexión WiFi e inicializamos el cliente MQTT, y después obtenemos los IDs de los sensores y actuadores del dispositivo para las peticiones que vendrán posteriormente. Por último obtenemos el valor medio del sensor de sonido para calibrarlo.
- **Loop**: En caso de no estar conectados al servidor MQTT nos volvemos a conectar, y luego mediante la función loop() de la librería de MQTT vamos comprobando si existen nuevos datos para los sensores, y luego actualizaremos sus valores. Cada 5 segundos postearemos los valores de los sensores, y un segundo después, durante 5 segundos, iremos tomando los valores de sonido y presión y obtendremos el mayor valor obtenido. Esto es porque postear el sonido o presión cada 5 segundos provocará que nos perdamos varios datos importantes, ya que varían en cuestión de décimas de segundo, y solo necesitamos fijarnos en los valores máximos que se alcanzan.
- **Funciones MQTT**: Tenemos la función Callback, que recibe el JSON y guarda los valores en las variables globales; y MQTTReconnect, que se conecta al servidor y se suscribe a los topics.

- **Funciones Actuadores**: Presentamos la función que nos obtiene los IDs de los actuadores a través de una petición HTTP. También tenemos dos funciones análogas entre ellas para escribir los valores recibidos en los actuadores. Estas funciones debemos llamarlas constantemente para conseguir la intermitencia de los actuadores a la intensidad que recibimos.
- **Funciones Sensores**: Tenemos una función análoga a la de los actuadores para obtener los IDs, además de funciones para hacer POST, en las que creamos el JSON y lo enviamos. También tenemos dos funciones leeSonido y leeGPS, ya que la lectura de estos dos sensores no podemos realizarla simplemente mediante una lectura de pin. En leeSonido obtenemos una media de los valores recogidos, y después realizamos algunos cálculos para normalizarlo, como por ejemplo elevarlo al cuadrado, para resaltar así los cambios. En leeGPS tenemos que comprobar que existan datos y luego usar una función de una librería para obtener las coordenadas.