Universidad del País Vasco
Euskal Herriko Unibertsitatea

Proyecto de Fin de Carrera

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

BirdTrackingSink: Soporte a la explotación de datos y gestión de una red de sensores

Autor
Xabier Acosta Pacheco

10 de julio de 2014

Resumen del proyecto

El seguimiento de distintas especies de animales contribuye en gran medida a su estudio y, por tanto, a su conservación y control. Los avances tecnológicos de los últimos años han facilitado las posibilidades de seguimiento con la creación de distintos dispositivos que permiten conocer los movimientos de la especie que se desea estudiar. Uno de los sistemas más utilizados consiste en la utilización de dispositivos GPS incorporados al espécimen sobre el que se realiza el seguimiento y cuya señal es recogida por satélites que se encargan de almacenar y posteriormente reenviar la información para su almacenamiento y procesamiento en el laboratorio. El principal problema de este sistema es su elevado coste. Existen alternativas que no presentan un coste tal alto, tales como el uso de módulos basados en telefonía móvil. Sin embargo, tienen limitaciones de cobertura, por lo que no es aplicable en todos los ámbitos.

Este proyecto forma parte de una propuesta que ofrece realizar seguimiento de ejemplares de una especie de ave, la gaviota Patiamarilla, en Gipuzkoa mediante la utilización de una red de sensores y que tiene varias ventajas frente a las opciones presentadas anteriormente. En este proyecto en concreto se ha diseñado e implementado el módulo que permite recoger la información obtenida por el conjunto de sensores (cada ejemplar lleva incorporado un sensor que permite registrar su posición) y enviarla a un servidor centralizado para su posterior consulta y análisis. Adicionalmente, también se permite consultar el último estado registrado de cada dispositivo de seguimiento, además de contemplar la posibilidad de actualizar su software.

Índice general

Índice de figuras

Indice de tablas

1. CAPÍTULO

Documento de Objetivos del Proyecto

1.1. Introducción

El seguimiento de los movimientos de la fauna constituye un aspecto fundamental en su estudio, conservación y control. Mediante el uso de la tecnología, más concretamente, el uso de dispositivos GPS, se puede satisfacer gran parte de la demanda que existe en el campo de la investigación de las especies animales que son objeto de estudio.

A la hora de elegir los dispositivos para la monitorización, además de la gestión de energía eficiente que deben realizar, existen también limitaciones físicas, ya que el dispositivo no debe afectar negativamente al comportamiento de los ejemplares. Por ello, existen ciertos límites referentes al tamaño y al peso que pueden tener estos sensores. En el caso del marcaje de aves se debe evitar que el peso de estos dispositivos sobrepase en un 3-5 % el peso del ejemplar.

Uno de los sistemas más empleados es la utilización de dispositivos GPS que, mediante satélites, reenvían la información a un sistema de almacenamiento. El problema de este sistema es su elevado coste, no sólo por el coste del dispositivo, sino también el de los contratos del servicio vía satélite. Por ejemplo, Microwave Telemetry Inc. [Telemetry] ofrece cada dispositivo por $2900, a lo que hay que sumar los más de 3000€ del contrato de datos vía satélite. Un estudio típico de una colonia de aves durante un año con una muestra de 20 ejemplares requeriría una inversión de 100.000 € en dispositivos, la mayoría de los cuales no se recuperarán.

Otra alternativa a este sistema se basa en la utilización de módulos basados en telefonía móvil 2G o 3G, como GPRS o UMTS, con contratos menos costosos. Estos módulos no se pueden emplear en determinados estudios como es el caso de este proyecto, ya que la limitación de cobertura es un factor clave.

La propuesta contemplada en este proyecto propone la utilización de una red de sensores como soporte harware. Una red de sensores inalámbrica (*Wireless Sensor Networks* o WSN) es una red formada por un gran número de dispositivos pequeños de baja potencia, llamados nodos o **motas**, que se encuentran espacialmente distribuidos formando una red temporal sin la asistencia de ningún

tipo de administración central. Los nodos pueden recolectar, procesar, y transmitir datos. Las motas cuentan con una unidad de energía limitada, por lo que su consumo energético debe ser bajo para garantizar el mayor tiempo posible de vida operativa. Para poder conseguir ese consumo reducido, se limita la capacidad de procesamiento y el sistema de comunicación, además de utilizar protocolos específicos para estos dispositivos. Es habitual encontrar una o varias motas que ejercen un papel de coordinador, por ejemplo para recopilar los datos recogidos por las motas y transmitirlos al exterior de la red. En este proyecto ese nodo recibe el nombre de **sink** o **estación base**. Para recoger los datos, las motas deben comunicarse con el sink, por lo que la colocación del mismo debe ser en un lugar clave, es decir, un lugar que las motas visiten de manera habitual para que la información pueda ser transferida.

Mediante el uso de motas se pretende conseguir reducir el gasto económico necesario para realizar un estudio, ya que el coste de cada dispositivo es de unos pocos cientos de euros la unidad.

En el presente proyecto se hará uso de esta última tecnología para para poder analizar la dispersión de algunos ejemplares de gaviota Patiamarilla nacidos en Gipuzkoa. La estación base se colocará en la colonia de las gaviotas, que generalmente anidan en acantilados, por lo que habrá que tener en cuenta que estará a la intemperie y en un lugar de difícil acceso a la hora de elegir el hardware utilizado y el sistema de comunicación. Las estaciones base contarán con un sistema de alimentación solar y batería (si no se dispone de red eléctrica).

En este proyecto se desarrollará la gestión del *sink* para exportación de los datos recogidos por las motas (incorporadas a los ejemplares). La interacción entre el sink y las motas forma parte del proyecto global y queda fuera del alcance de este proyecto. Además de la exportación, también se desarrollará un módulo para hacer accesibles dichos datos por medio de un servicio web. Al desarrollar un sistema específico para la estación base, se optará por unas características lo más generales posible, de tal modo se pueda seleccionar el hardware que mejor se ajuste al escenario de aplicación.

1.2. Objetivos del proyecto

El objetivo principal de este proyecto es desarrollar el prototipo de una aplicación que recibe y gestiona los datos sobre seguimiento recogidos por las motas.

Las funcionalidades requeridas son las siguientes:

- El sistema deberá mostrar los datos recogidos por las motas. Debido a que el proyecto del que va a formar parte este módulo utilizará motas con sensores de localización, se ha considerado que la opción más adecuada para mostrar los datos sea mediante un mapa.
- Se deberá mostrar la información del estado de las motas para que se puedan detectar posibles problemas.
- Debe permitir el envío de datos al sink.

Cómo el proyecto principal todavía se encuentra en fase de desarrollo, la aplicación trabajará sobre datos teóricos que deberemos generar para la fase de pruebas del proyecto. Partiremos de la base

de que estos datos se encuentran en el sink, ya que la comunicación entre el sink y las motas se desarrolla fuera del ámbito de este proyecto. Al no tratarse de información confidencial, puesto que son datos teóricos, la seguridad de la misma se ha dejado fuera del alcance de este proyecto.

1.3. Arquitectura del Sistema de localizaciones

Inicialmente, en el proyecto global, tendremos una red de sensores (motas) que irán colocadas sobre los ejemplares. Estas motas se encargarán de recopilar continuamente los datos de posicionamiento y de transferirlos al nodo coordinador. En este proyecto partiremos de la base de que la información que han recogido las motas se encuentra disponible en la estación base.

Para la arquitectura del sistema se han planteado dos posibilidades en función de cómo se distribuyan los dos roles de la aplicación (*recolector* de información de las motas y *servidor* de la información recolectada).

La aproximación más sencilla consiste en que la base encargada de recopilar los datos de las motas sea la encargada de mostrar los datos, por lo que debería ser accesible para poder mostrar los datos. Además de eso, debe tener capacidad para poder recopilar datos y mostrarlos. Esto significa que deberá contar con suficiente almacenamiento para poder guardar los datos recogidos y un hardware con la potencia necesaria para poder recoger y mostrar dichos datos.

Esta posibilidad queda descartada por diversas razones:

- Para poder consultar los datos el sink necesitará tener la conexión activa, lo que supone un mayor consumo. Puede que no se disponga de una instalación eléctrica, por lo que dependería de una batería, lo que significa que hay que limitar el consumo en la medida de lo posible. Se podría activar la conexión de manera intermitente, de manera que el consumo sea menor, pero limitando la posibilidad de consulta de datos a los momentos en los que la conexión se encuentre activa.
- Tener que mostrar los datos al mismo tiempo que recopila los datos supone un aumento de carga, y, en consecuencia, un aumento de consumo, como en el caso anterior, es deseable que este consumo sea lo menor posible.
- Al encontrarse a la intemperie la posibilidad de fallo del sistema es más probable, lo que podría resultar en una pérdida de los datos recopilados hasta el momento. Se podrían realizar copias de seguridad, pero, al igual que en los casos anteriores, supone un mayor gasto energético.

Teniendo en cuenta las razones expuestas, se ha decidido que la estructura del sistema este compuesta por un nodo coordinador y un sistema central que trabajan del siguiente modo. El nodo coordinador se encargará de recopilar los datos de las motas. Estos datos en ocasiones se podrían recopilar manualmente, pero en otros casos, los lugares en los que se colocarán las estaciones pueden tener un acceso difícil. Para reducir la necesidad de intervención humana, se propone que cada cierto tiempo activará el sistema de comunicación, que puede ser mediante radio, Internet, etc. y enviará estos datos a un servidor central, que será el encargado de mostrar los datos recopilados. Con esto evitamos la carga de tener que mostrar los datos desde el sink y al mismo tiempo permitimos

que la conexión no tenga que estar activa en todo momento, puesto que solo se activaría cuando se fueran a enviar los datos.

Al no tratarse de información crítica, esta tarea se puede ejecutar una vez al día, reduciendo en gran medida tanto el consumo de datos de la conexión como el consumo eléctrico del sistema, además de tener un sistema encargado de mostrar los datos más accesible, puesto que el servidor central tendrá una conexión y una fuente de energía más estable.

También podremos introducir datos de configuración de las motas en el servidor. Al igual que en el caso anterior, el nodo coordinador tendrá una tarea periódica que se encargará de descargar los parámetros de configuración y ponerlos accesibles a las motas. De esta manera, cuando las motas se conecten con el nodo, actualicen sus parámetros.

Este sistema nos proporciona unas ventajas:

- Al tener el servidor en un entorno más controlado, como una habitación, la gestión y administración es más sencilla.
- El servidor tendrá una conexión a la red eléctrica y una mejor conexión, pudiendo disponer de los datos en todo momento.
- Reducimos en gran medida el consumo energético del sink. Si bien con este sistema sigue existiendo la posibilidad de que se quede sin batería, el consumo está más optimizado.

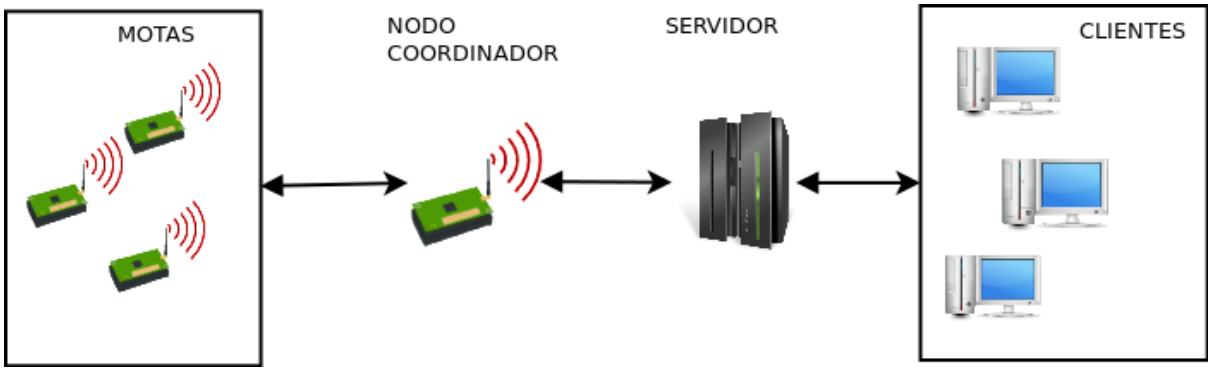

Figura 1.1. Arquitectura del sistema

Para consultar los datos será necesario un sistema que tenga acceso al servidor central, ya sea poniendo accesible el servidor desde Internet o estando conectado en la misma red local u otro sistema de comunicación.

1.4. Análisis de factibilidad, alternativas y antecedentes

1.4.1. Antecedentes

Como ya se ha comentado, en el mercado actualmente existen alternativas al sistema que se va a desarrollar. En la siguiente tabla (fuente: [TecnicasMarcaje]) se presenta un resumen de los métodos de localización disponibles.

Tabla 1.1. Características de los dispositivos de seguimiento remoto más habituales

Tipo de dispositivo	Precisión	Frec de señales	Peso	Autonomía	Logística y limitaciones	Coste del equipo
Emisores de radio	Media-baja en el mar (1-10 km)	Depende del esfuerzo	Bajo (< 1g)	Media(semanas-meses)	Logística muy costosa. Se requiere la adquisición directa de datos en tiempo real.	Bajo-medio
Geolocalizadores (GLS)	Muy baja (>100 km)	Baja (1-2 / día)	Bajo (< 1g)	Alta (años)	Se requiere recuperar el dispositivo para recuperar los datos. Recuperación muy espaciada respecto al marcaje (meses/años).	Bajo-medio
Emisores vía satélite (PTT)	Media-alta (< 1-10 km)	Media-baja (< 10 / día)	Medio (> 5g)	Media-alta (hasta >> 1 año en PTT solares)	Coste de seguimiento limitado a alquiler de la señal de satélite. Datos accesibles en tiempo real vía Internet.	Medio-alto
Registradores de compás	Media-alta (<1-10 km, acumulativa)	Elevada (segs-mins)	Medio-alto (> 17g)	Baja (días/semanas)	Se requiere recuperar el dispositivo para descargar los datos.	Medio
Registradores de GPS	Muy buena (< 10m)	Elevada (segs - mins)	Medio-alto (> 10 g)	Baja (días-semanas)	Se requiere recuperar el dispositivo para descargar los datos	Bajo
Emisores vía satélite con GPS	Muy buena (< 10m)	Media-baja (<10 / día)	Alto (> 22g)	Media-alta (hasta > 1 año en PTT solares)	Coste de seguimiento limitado a alquiler de la señal de satélite. Datos accesibles en tiempo real vía Internet	Muy alto
GPS-GSM	Muy buena (< 10m)	Media-alta (mins - horas)	Muy alto (> 100 - 150g)	Media (semanas-meses)	Descarga de datos restringida a tener cobertura telefónica, que es baja en el mar	Alto

En general el uso de satélites y tecnología móvil son las opciones más utilizadas ya que resultan muy cómodas y de aplicación directas. Además, los dispositivos que ofrecen están bastante optimizados en cuanto a peso y tamaño. Sin embargo, debido a los costes de uno y limitaciones geográficas del otro, se plantea utilizar el sistema de localización mediante motas sin perder las ventajas que supone el uso de esos sistemas.

El sistema de motas con sinks ya ha sido explorado en el proyecto *UvA Bird Tracking System* de la Universidad de Amsterdam [UvA]. Tras varios años de optimizaciones, los dispositivos no superan los 6g de peso en su versión más ligera. Estos dispositivos se han utilizado en estudios de aves migratorias (véase por ejemplo [BoutenBSC13]) instalando estaciones base y antenas repetidoras en lugares de paso de las aves para la descarga de datos.

El sistema desarrollado en [UvA] se basa en sistemas electrónicos específicos, lo que permite ajustar mucho las características globales del dispositivo. El problema de este enfoque es que implica costes elevados. La UvA ofrece sus sistema por 1200-1500€ dispositivo, 3000€ por antena y 5000€ por estación base. Comparados con los sistemas comentados anteriormente, es una inversión menor. Sin embargo se trata de un sistema cerrado, por lo que no se puede utilizar esta tecnología libremente.

Como podemos observar, el sistema planteado mediante el uso de motas nos ofrece varias ventajas con respecto a los distintos sistemas disponibles. Se sacrifica la opción de un seguimiento en tiempo real que ofrece el sistema de satélites, a cambio de un menor coste. A diferencia de los registradores de GPS/compás, no se requiere recuperar el dispositivo para poder descargar los datos.

1.4.2. Análisis de Factibilidad

La propuesta en torno a la que gira este proyecto propone el uso de estaciones base de construcción sencilla y de bajo coste. Al no requerirse un equipo específico, se puede elegir entre una gran variedad de sistemas. Esto nos proporcionará un gran abanico de posibilidades, pudiendo optar por equipos de mayor o menor coste, dependiendo de las necesidades del estudio.

El coste de las motas es de unos pocos cientos de euros frente a los varios miles de euros de los dispositivos de localización por satélite o GSM [1]. Como ya se ha comentado, existe el riesgo de que estos dispositivos no puedan recuperarse, por lo que realizar un segundo estudio supondría de nuevo otro gran desembolso si se usan los dispositivos de mayor coste.

Al no existir ningún módulo disponible para el sistema de motas que se está desarrollando en el proyecto principal y ofrecer una solución de bajo coste, este proyecto resulta interesante para complementar el sistema.

Por ello, este proyecto resulta interesante tanto a nivel económico como a nivel de prestaciones.

1.4.3. Estudio de alternativas

Consulta de la infomación: aplicación gráfica vs web

Se han planteado dos maneras de realizar este proyecto: aplicación gráfica o aplicación web. En este apartado se justifica la elección de la segunda.

Cada día la web ofrece más posibilidades y es más potente. Tras estudiar los requisitos necesarios se ha comprobado que una aplicación web es factible y tiene varias ventajas:

- Se puede usar en cualquier sistema operativo que tenga disponible un navegador web. Hoy en día, con el aumento del uso de dispositivos móviles como smartphones o tablets, esto es un factor muy importante.
- Se puede acceder desde cualquier sitio. Basta con tener acceso al servidor donde se encuentra alojada la aplicación para poder conectarse, sin necesidad de instalaciones.
- Toda la información está centralizada: bastará con tener un buen sistema de respaldo para el servidor central para evitar la perdida de datos.
- Hay herramientas disponibles que facilitan desarrollar tareas requeridas en este proyecto, como un sistema de generación de mapas para mostrar las localizaciones.
- Para realizar una actualización de la aplicación, basta con actualizar el servidor, mientras que la aplicación gráfica requeriría actualizar cada uno de los clientes.

La desventaja de una aplicación web frente a una aplicación gráfica puede ser el rendimiento. Sin embargo, con los sistemas de hoy en día y los grandes avances en cuanto a sistemas web, esta diferencia de rendimiento ha ido disminuyendo considerablemente.

[1] Wikipedia - GSM (http://es.wikipedia.org/wiki/Sistema_global_para_las_comunicaciones_m%C3%B3viles)

Lenguaje de programación

Por preferencias del cliente, la aplicación se desarrollará en Python. Se ha estudiado esta posibilidad e inicialmente es perfectamente asequible hacerlo en dicho lenguaje. A no ser que en una fase avanzada del proyecto no nos de la suficiente funcionalidad para cumplir los objetivos, cumpliremos con las preferencias del cliente. Además, al tratarse de un lenguaje multiplataforma, nos proporciona más opciones a la hora de elegir sistema operativo sobre el que diseñar la aplicación.

Una desventaja de Python es que se trata de un lenguaje interpretado. Esto significa que requiere un intérprete para traducirlo, de manera que el procesador pueda procesarlo. Este procesamiento se traduce en un mayor consumo. Si en un futuro se desea conseguir un menor consumo del sistema, se debería pensar en sustituir Python por un lenguaje compilado.

Aplicación para la generación de mapas

Como ya se ha comentado, las motas se van a encargar de recoger localizaciones y transmitirlas al sink. Se ha considerado que la mejor manera de mostrar este tipo de datos es posicionándolos en un mapa, por lo que se han considerado distintas opciones para realizarlo.

La primera opción es utilizar un generador propio de mapas. Se ha descartado esta opción debido a que el sistema quedaría limitado a una zona, o requeriría un tiempo de desarrollo elevado, lo que podría significar no cumplir con los objetivos del proyecto a tiempo. Por lo tanto, se ha optado por utilizar soluciones ya disponibles.

Entre estas opciones se ofrecen varias alternativas:

- Google Maps: Una de las soluciones más conocidas y completas ofrecida por Google.
- Yahoo Maps: Buena buena alternativa para visualizar mapas, en combinación la imagen en satélite.
- Bing Maps: Del buscador de Microsoft, Bing ofrece también ser un competidor de Google Maps.
- OpenStreetMap: Se trata de una plataforma que permite ver mapas cartográficos del mundo, es la alternativa libre a Google Maps. Que su contenido sea libre significa que se puede hacer uso de él con total libertad y sin coste alguno, bajo la licencia Open Database License.
- Mapas de Ask.com: El buscador nos ofrece una alternativa muy buena también, casi al estilo Google, con vista desde el satélite o topográfica.

Las desventajas de estos sistemas es que requieren que el cliente esté conectado a Internet para hacer uso de ellos.

Para el desarrollo del prototipo se ha optado por el uso de Google Maps, ya conocida por el autor de este proyecto y que cuenta con una buena documentación, lo que ayudará a reducir el tiempo de desarrollo de manera que se puedan cumplir con los plazos previstos. Además de esto, su uso

es gratuito con la licencia básica[2], perfecto para cubrir las necesidades de un prototipo básico. En caso de querer seguir utilizando en el proyecto final habría que consultar si se pueden cumplir las normas de uso de la licencia básica. Destacar que, en caso de hacer uso de la licencia básica, hay un límite de consultas diarias y el servicio debe estar disponible de forma pública y gratuita. Google Maps también ofrece una licencia de pago que permite usar el servicio para sitios privados, por lo que se podría mantener el sistema actual con las nuevas condiciones.

Nodo coordinador

Como ya hemos comentado, el sistema de motas requiere un nodo coordinador que permita recopilar los datos y, en nuestro caso, enviarlos a un servidor para que los ponga disponibles a través de Internet.

Necesitamos que cumpla varias características como son:

- Bajo consumo, debido a que en muchas ocasiones se encontrarán alimentados mediante placas solares y baterías.
- Que pueda comunicarse con otros sistemas, para poder enviar los datos al servidor central.
- Soporte para Python, ya que es el lenguaje de programación por el que se ha optado.
- Que se pueda aislar correctamente para su uso en exteriores, puesto que generalmente estarán situados en lugares de acceso habitual para los ejemplares estudiados.

Las opciones son muy amplias, tanto en potencia como coste. Estos pueden ser placas de bajo coste, como puede ser una Raspberry Pi, dispositivo con un coste de 40€ y una caja de aislamiento de 40 € y que nos proporciona un equipo que cumple las características indicadas, al que se le puede incluir el sistema operativo *Raspbian*, basado en *Debian*. La desventaja de estos equipos de bajo coste es su potencia, al tratarse de un único procesador ARM a 700MHz. Gracias a su baja potencia también se consigue un bajo consumo, de tan sólo 3W. También puede adquirirse un sistema *rugged*, que se tratan de ordenadores más potentes, con distintas características en cuanto a memoria y procesador, resistentes al agua y a golpes. En este caso nos encontramos ante equipos más potentes y ya preparados para estar a la intemperie; la desventaja, su precio, que ronda los 2000€, con unos consumos superiores de 16W.

Debido al presupuesto limitado para este proyecto y al tratarse de un prototipo, se ha optado por el uso de una Raspberry Pi. Con esto conseguimos un sink simple y económico que cumple con las necesidades del desarrollo.

Hay dos modelos disponibles de la Raspberry Pi: el A y el B, con diferencias importantes en cuanto a Hardware. El modelo A cuenta con 256 MB de RAM, 1 USB y no cuenta con ningún tipo de conectividad de red. El modelo B tiene 512 MB de RAM, 2 USB y cuenta con un puerto 10/100 Ethernet (RJ-45).

El nodo coordinador no va a necesitar mucha memoria RAM, puesto que solo va a necesitar enviar los datos de las localizaciones y recibir los datos de configuración. Se ha considerado que el modelo

[2] Licencia Google Maps (https://developers.google.com/maps/licensing?hl=es)

A dispone de memoria suficiente para esta tarea. Sin embargo, las motas con las que contamos se conectan a través de USB. Al necesitar un USB para conectar la mota encargada de recibir el resto de motas y otro puerto que proporcione la comunicación con el servidor, se ha optado por el modelo B.

Como ya se ha comentado, necesitamos que tenga un bajo consumo, y, debido a la adición del puerto Ethernet y el USB extra, el consumo en el modelo B es bastante mayor que el modelo A, pasando de consumir 500mA (2.5W) en el modelo A a los 700mA(3.5W) del modelo B. Además de eso, la Raspberry Pi nos permite editar la velocidad de la CPU, pudiendo reducirla o aumentarla. Al reducirla, conseguiríamos reducir el consumo energético aun más, pudiendo establecerlo en 100MHz (en lugar de los 700 originales) y reducir el consumo hasta los 90mA. Obviamente, esto también reduciría la capacidad del sistema, por lo que habría que hacer unas pruebas de carga hasta conseguir una relación rendimiento/consumo óptima. Dichas pruebas quedan fuera del alcance de este proyecto.

Otra de las ventajas es que no cuenta con sistemas mecánicos, por lo que es resistente a golpes. Además de generar poco calor debido al bajo consumo, por lo que permite un aislamiento perfecto para el clima exterior.

La otra ventaja es la posibilidad de elegir la capacidad de almacenamiento, ya que cuenta con una ranura SD para introducir una tarjeta del tamaño deseado. Dependiendo de la cantidad de motas que se vayan a gestionar nos permitirá decidirnos por un tamaño u otro. Se ha realizado un breve estudio aproximado del tamaño necesario y se ha considerado que una tarjeta de 8GB será más que suficiente, por las siguientes razones:

Partimos del caso de que se usará *Raspbian*, que es un sistema operativo GNU/Linux basado en *Debian Wheezy armhf (7.0)*, especialmente adaptado para la Raspberry, lo que ocupará unos 2GB inicialmente. Creando un fichero con los datos teóricos utilizados por la aplicación (localización, fecha, identificador...) se ha observado un tamaño de 10MB por cada 220.000 registros. Suponiendo el caso de que se gestionen 1000 motas, con un intervalo de tiempo entre cada toma de localización de 1 minuto, generarían 60.000 registros por hora. Suponiendo que el envío de datos se haga cada 6 horas serían 360.000 registros (15MB~). Al hacer el envío este fichero se eliminará, manteniendo solo aquellos registros que hayan fallado al enviarse.

Además, existirá un *log* con el estado de las tareas de sincronización de datos. Suponiendo una tasa de error del 5 % en cada sincronización, es decir, unos 20000 registros del log por cada sincronización, y una sincronización cada 6 horas, serían 10MB de registros cada 3 días. Esto haría un total de 100MB al mes, 1,2GB al año. Contando con una tarjeta de 8GB se tardarían unos 5 años en llenarla. Eso si se quiere mantener el log en todo momento, y suponiendo una tasa de error bastante elevada. En el mejor de los casos se generarán unos 20 registros diarios.

Por ello, a falta de más información del uso final y datos reales, se ha optado por una tarjeta de 8GB.

Para el uso en exterior se requiere de un aislamiento específico: protección contra el agua, temperaturas bajas y altas. Ya existen soluciones de este tipo, como puede ser la caja PiCE [3], lo que nos proporciona resistencia al agua y trabajar en un rango de temperaturas desde los -100º hasta los 200º.

[3] PiCE is a Metal, Outdoor Raspberry Pi case http://elsondesigns.com/pice-raspberry-pi-camera-enclosure

La elección de este dispositivo para el desarrollo del prototipo es adecuada por varias razones:

- Es un sistema de bajo coste: Una Raspberry Pi con una protección para montar en el exterior y un sistema de conexión WiFi o 3G no excede los 100€. A esto habría que añadirle el coste de una placa solar junto a la batería. Al ser un equipo de mucho menor consumo que un PC normal, se pueden adquirir conjuntos menos potentes, y por lo tanto, más baratos.
- Se puede probar el sistema en un equipo de potencia limitada, por lo que si la carga es aceptable, se puede seguir utilizando un equipamiento similar y si no, siempre se puede optar por un equipo más potente.
- El sistema operativo utilizado se trata de software libre basado en Debian GNU/Linux, por lo que los usuarios de Linux tendrán más facilidades para adaptarse a este sistema.

Comunicación con entre roles

Como ya se ha expuesto en la definición de la arquitectura del sistema, los datos recogidos por el sink necesitan ser transferidos al servidor, que será el encargado de mostrar los datos.

Hay diversas opciones:

- Reemplazar la memoria del sink, llevar la existente al servidor y descargar los datos. Esta opción, además de requerir personal, puede ser realmente difícil. Como se ha comentado, las estaciones base pueden estar en lugares de difícil acceso. Por lo tanto está opción sólo se plantea si el resto de posibilidades no son posibles.
- Bluetooth: Debido al corto alcance del Bluetooth (~30m) [4] estaríamos en un caso similar al anterior, con la desventaja que tendríamos un dispositivo que consumiría más batería, ya que necesitaríamos que estuviera siempre encendido.
- Radio/3G/WiFi: Con la tecnología actual, es una de las opciones mas recomendables, ya que los datos se pueden enviar directamente al servidor a través de Internet por un coste bajo. Se ahorraría en personal para invertir en tecnología.

Transferencia de datos entre roles

Para la comunicación entre los roles existen distintas formas de transmitir los datos. Una opción es mediante el protocolo *File Transfer Protocol* (FTP). Un problema básico de este sistema es que está pensado para ofrecer la máxima velocidad en la conexión, pero no la seguridad, ya que todo el intercambio de datos se realiza en texto plano, sin ningún tipo de cifrado. Como ya se ha expuesto anteriormente, en este proyecto usaremos datos teóricos, por lo que la seguridad de los datos no es de gran importancia, sin embargo, una solución a este problema es Secure Copy (SCP), ya que los datos son cifrados durante su transferencia. Otra opción es mediante servicios web como pueden ser *Simple Object Access Protocol* (SOAP) o *Representational State Transfer* (REST). REST resulta más útil cuando el ancho de banda es importante y necesita ser limitado, donde la sobrecarga de las

[4] Bluetooth (http://es.wikipedia.org/wiki/Bluetooth)

cabeceras y capas adicionales de los elementos SOAP debe ser restringida, mientras que SOAP es más útil para describir con detalles el servicio web.

La ventaja del servicio web frente a los anteriores es que permitiría enviar las localizaciones y tenerlas disponibles en el servidor mientras se envían, mientras que con FTP o SCP, hasta que no se copiase el archivo entero no se podrían procesar. Se podría reducir el tamaño de los ficheros a enviar mediante compresión, esto requeriría una potencia de procesamiento superior, lo que se traduce en un mayor consumo del sink, que, como ya hemos comentado, tiene una fuente de energía limitada, por lo que sería recomendable evitar esta opción.

Almacenamiento de los datos

La modo en el que los datos se almacenan influye en gran medida en el rendimiento del sistema, por lo que es importante seleccionar el más adecuado para el sistema.

Para un sistema de lectura y escritura simple, un fichero puede ser adecuado. Sin embargo, para llevar un control de estos datos una base de datos nos resuelve ciertos problemas que se podrían presentar en caso de usar ficheros:

- Acceso concurrente: Permite a varios usuarios actualizar los datos simultáneamente sin dar lugar a datos inconsistentes.
- Existe la posibilidad de que cuando ocurra un fallo se restauren los datos a un estado de consistencia que existía antes del fallo.
- Ofrecen sistemas para consultar datos específicos, como puede ser un intervalo de fechas.
- Seguridad: Permite gestionar que usuarios pueden acceder a que datos.
- Integridad: Los datos deben satisfacer ciertas restricciones de consistencia, de manera que sea más difícil equivocarse al introducir un dato.
- Centralizado: Permite que los datos se encuentren en un solo lugar en vez de distribuidos en distintos ficheros y carpetas, facilitando las copias de seguridad.

Una de las posibles desventajas puede ser en cuanto a consumo de potencia. Para cantidades pequeñas de datos, un fichero consume menos recursos que una base de datos. Sin embargo, al tratar con ficheros grandes, una base de datos es más eficiente que un fichero.

Por ello se ha optado por usar un sistema de base de datos para conseguir una arquitectura cliente/-servidor en 3 niveles. Esto nos ofrece mayor seguridad, ya que se puede definir el nivel de seguridad independientemente para cada nivel, y un mejor rendimiento, ya que las tareas se comparten entre los distintos servidores.

1.5. Planificación del Proyecto

1.5.1. Gestión del Alcance

Para cubrir las necesidades del sistema el proyecto deberá contar con distintas partes:

- *Tracking*: Mostrar las localizaciones recogidas por las motas en un mapa. Para ello necesitaremos un sistema que permita al nodo coordinador enviar las localizaciones al servidor.
- *Estado de las motas*: Se deberá mostrar un resumen con el estado de las motas, que indicará la fecha del último dato recogido por cada una y alertará en caso de que lleve una semana sin enviar ningún dato.
- *Gestión de motas*: Se podrán establecer una serie de parámetros en el servidor que posteriormente serán trasmitidos al nodo coordinador, de forma que las motas puedan recoger estos nuevos parámetros, como pueden ser el intervalo de tiempo entre cada captura de localización o el número de versión del software de la mota, así, en caso de ser menor del indicado, tendrán que realizar una actualización (inicialmente no se considera dentro de este proyecto la actualización del software de las motas). Para esto se contarán con dos funcionalidades: una que permita introducir los parámetros indicados, y la otra será la del envío de datos desde el servidor al sink.

No forma parte de este proyecto la comunicación entre las motas y el nodo coordinador, pero se ofrecerá una guía explicando el formato en el que se espera recibir los datos.

Restricciones y limitaciones

Para el desarrollo del proyecto se propone desde el cliente utilizar el lenguaje Python.

A la hora de elegir los componentes y la arquitectura del sistema se ha de tener en cuenta el entorno en el que va a encontrarse el dispositivo.

- **Entorno**: El sistema que se implante tiene que estar protegido contra la lluvia, temperaturas altas y bajas o golpes.
- **Consumo**: Hay que tener en cuenta que puede que no llegue la corriente a los lugares donde estará colocado, por lo que se desea una implantación que pueda funcionar con unas baterías conectadas a paneles solares. Para esto, se necesita un sistema de bajo consumo.
- **Conexión**: Al igual que sucede con el consumo, lo mismo para la conexión. Hay que considerar que no se dispondrá de una buena conexión y que se usará algún sistema alternativo al habitual por cable como puede ser 3G, GPRS, etc. Por lo que la disponibilidad puede ser limitada

1.5.2. Estructura de Descomposición del Trabajo

El trabajo a realizar se puede clasificar en tres tipos de procesos: tácticos, operativos y formativos, tal y como se presenta en la siguiente figura. .. figure:: _static/img/edt.png

Estructura de descomposición del trabajo

En los siguientes apartados se describirán dichos procesos.

Tácticos

- *Objetivos y alcance*: Inicialmente se hará un análisis de las necesidades y los requisitos necesarios para el desarrollo del proyecto.
- *Planificación*: Tras analizar los objetivos se descompondrá en varias tareas, para las que habrá que organizar el tiempo para que el proyecto se pueda entregar en el plazo previsto.
- *Plan de contingencia*: Se estudiarán las causas por las que el proyecto no se pueda desarrollar correctamente y se prepararán una serie de respuestas a los posibles problemas presentados.
- *Método de trabajo*: En esta etapa se pensará la mejor forma de trabajar de manera que el objetivo final cumpla con las necesidades reales del cliente.
- *Estudio de alternativas*: Se analizarán las distintas posibilidades para que el sistema se adapte al cliente y resuelva los problemas que se presentan actualmente de la mejor manera posible.
- *Reuniones*: Se harán varias reuniones con el cliente para mostrar el progreso y detallar necesidades que puedan haberse olvidado en un comienzo
- *Redacción de la memoria*: Redacción y revisión de la memoria del proyecto.

Operativos

- *Captura de requisitos*: Se estudiarán las necesidades del usuario para que la aplicación cumpla con los objetivos establecidos al comienzo.
- *Análisis*: Se analizará la forma de uso del sistema por parte del usuario, de manera que sea fácil y eficiente.
- *Diseño*: En esta etapa se especificará el funcionamiento del sistema.
- *Implementación*: Desarrollo de la aplicación para que cumpla con los objetivos establecidos.
- *Pruebas*: Etapa para la comprobación del correcto funcionamiento del sistema.

Formación

- *Investigación*: En esta etapa se hará una investigación de las tecnologías disponibles y de como usarlas.
- *Reuniones*: En caso de que se presenten problemas que no se puedan resolver, se harán reuniones con el profesor para plantear distintas posibilidades.

1.5.3. Metodología y Herramientas a utilizar

Para este proyecto se emplearán las metodologías basadas en el desarrollo ágil de software[5], metodología que permite obtener resultados en cada iteración, a diferencia de otras metodologías,

[5] Wikipedia - Desarrollo Ágil de software (http://es.wikipedia.org/wiki/Desarrollo_%C3%A1gil_de_software)

derivadas del ciclo de vida en cascada. Estas metodologías requieren una mayor planificación y documentación, dividiendo el desarrollo en iteraciones de identificación de requisitos, casos de uso, análisis, diseño, implementación y pruebas, en el orden indicado. Al tratarse de un proyecto relativamente acotado en presupuesto y que no está completamente definido dado que depende de otro proyecto que desarrolla en paralelo, podrán surgir cambios a lo largo del desarrollo.

El desarrollo ágil sustituye las iteraciones del desarrollo en cascada, en el cual, hasta que no se completan todas las iteraciones, no hay una versión funcional del software, mientras que en la metodología ágil disponemos de una versión funcional del software al final de cada iteración. Se harán iteraciones de un mes, teniendo, al final de cada iteración, una versión del software disponible. Esto no implica que la no se requiera planificación y documentación, sino que no tiene que ser tan estricta y facilita los cambios en los requisitos, puesto que el cliente va siguiendo el desarrollo continuamente. Se permite hacer cambios de los objetivos iniciales al final de cada iteración, simplificando así la tarea, ya que en el desarrollo en cascada, un cambio importante de objetivos en mitad de proyecto supone comenzar de nuevo casi desde el principio.

El desarrollo en cascada suele ser mejor para proyectos bien definidos, en los que hay pocas probabilidades de que surjan cambios durante el proceso de desarrollo. Por el contrario, las metodologías ágiles son más adecuadas para proyectos pequeños donde es muy probable que aparezcan cambios, como es el caso de este proyecto. Al no estar completamente definidos los objetivos, esto nos permite trabajar en pequeños módulos, sin necesidad de definir el sistema completo de antemano, de manera que sea más fácil responder a los cambios en requisitos indicados por el cliente si fueran necesarios.

1.5.4. Planificación

El presupuesto inicial del proyecto es el correspondiente a 6 créditos que, a razón de unas 25 horas por crédito, suman 150 horas.

Comenzando el proyecto en Octubre se ha puesto como fecha límite el 1 de Abril. Inicialmente planteando una inversión de 10 horas semanales, puesto que el tiempo disponible es limitado al compaginar el proyecto con el trabajo.

En caso de no cumplir con las expectativas iniciales en los cumplimientos de plazos se aumentará el tiempo invertido hasta las 20 horas semanales, siendo está la cantidad limite dedicada semanalmente.

En este intervalo de tiempo se deberá completar el desarrollo del proyecto, incluyendo la formación, el desarrollo de la aplicación y de la memoria.

En el siguiente apartado se detallan las tareas en las que se divide el proyecto.

1.5.5. Gestión de tareas

En este apartado se detallarán las tareas en las que se ha descompuesto el proyecto. Cada actividad del proyecto esta descompuesta por sus respectivas tareas

Tabla 1.2. Gestión de tareas

Tarea	Horas
Memoria	60 horas
Formación	30 horas
Desarrollo de la aplicación	
Sistema de comunicación entre roles	20 horas
Sistema de tracking	20 horas
Estado de las motas	10 horas
Configuración de las motas	10 horas
Total	**150 horas**

A continuación se describe cada uno de los apartados:

- **Memoria** Esta tarea, que estará activa a lo largo de todo el proyecto, consistirá en la redacción de la memoria. Se hará una versión inicial para presentar al cliente las opciones y el análisis del proyecto y posteriormente se irán agregando las correcciones y el proceso de desarrollo o manuales de uso e instalación.
- **Formación** Esta actividad recoge todas las tareas de aprendizaje requeridas para este proyecto

  1. Tecnológica

     - Formación en el uso de Python
     - Herramientas para el desarrollo de la aplicación
     - Sphinx para la creación de documentos
     - Herramientas para la puesta en marcha de la aplicación

- **Sistema de tracking** En esta tarea se incluye la puesta en marcha de la aplicación hasta el punto de disponer de un sistema para consultar los datos almacenados, con una gestión de usuarios y permisos.
- **Estado de las motas** Apartado donde se mostrarán las motas que estén en un estado crítico [6] o no tengan datos y el estado del nodo coordinador
- **Configuración de las motas** Permitirá introducir valores para los parámetros configurados para las motas a los usuarios que tengan permiso para hacerlo.
- Comunicación entre roles

  - **Envío de localizaciones**: Nos permitirá enviar las localizaciones almacenadas en el sink al servidor.
  - **Obtención de configuración**: Nos permitirá enviar los datos de configuración desde el servidor al sink

[6] Se considerará que está en estado crítico si ha pasado más de una semana desde la última localización recogida.

1.5.6. Análisis de Riesgos

En este apartado se detallan varios riesgos considerados, que generalmente son comunes a todos los proyectos informáticos, y la forma de hacerles frente en caso de que surjan durante el desarrollo de este proyecto.

No cumplimiento de plazos

Al tratarse de un proyecto de gran envergadura, es posible que la planificación sea incorrecta, por infravalorar la complejidad de alguna tarea. Por ello, se ha propuesto realizar el proyecto, planificando con un margen de 2 meses respecto a la fecha de entrega final. Esto permitirá ajustar las horas en caso de que sea necesario.

Cambio de requisitos

En caso de que durante el desarrollo del proyecto se necesiten cambiar los objetivos, esto puede conllevar un retraso en las entregas. Gracias a las metodologías ágiles se podrá solventar este problema, al ser planificaciones individuales para cada funcionalidad, y no global para todo el proyecto.

Pérdida total o parcial del código fuente o documentación

Un error humano o de hardware puede implicar perder todo (o parte) del desarrollo del proyecto. Para estar protegido contra esta situación se utilizarán sistemas de almacenamiento en la nube que nos permitirá recuperar los datos en caso de que surja algún problema. Se utilizará BitBucket como repositorio Git, además de Dropbox y el equipo personal, que dificultará que se dé el caso de la pérdida.

Desconocimiento de las tecnologías a utilizar

Debido a que es la primera aplicación en Python, el desconocimiento de las tecnologías significará que el proyecto llevará más horas que si se usará un lenguaje ya conocido. Además de esto, puede presentarse el problema de que las herramientas utilizadas no sean suficiente potentes para las necesidades del proyecto, en cuyo caso habría que buscar alternativas, y tal vez requiera comenzar desde el principio. Si este problema apareciese en una fase avanzada del proyecto podría suponer un gran retraso en la finalización del proyecto.

Teniendo en cuenta esto, se han analizado las necesidades del proyecto y las posibilidades de las tecnologías y con los objetivos planteados inicialmente son suficientes. A menos que haya cambios de última hora que se desvíen mucho de la idea inicial, esto no debería suponer ningún problema para el desarrollo del proyecto, salvo un aumento de las horas necesarias inicialmente dedicadas a formación.

Para solventar este problema se dedicará un periodo inicial de 30 horas para investigar sobre las tecnologías que se utilizarán en el proyecto.

2. CAPÍTULO

Desarrollo Técnico

2.1. Análisis de requisitos

2.1.1. Requisitos funcionales

A continuación figuran los requisitos funcionales de la aplicación:

- Mostrar la información recogida por las motas.

  Al tratarse de localizaciones se ha optado por mostrarlas en un mapa. Para facilitar la visualización se podrán seleccionar las motas que se quieren mostrar y un intervalo de fechas de las localizaciones.

- Pantalla para la gestión de motas, usuarios y permisos
- Sistema que permita introducir localizaciones a usuarios con los permisos adecuados.
- Pantalla para modificar los parámetros de configuración de las motas a los usuarios con los permisos adecuados.
- Sistema que muestre los parámetros de configuración de las motas
- Mostrar el estado del nodo coordinador y las motas que se encuentran en estado crítico o sin datos. Se considera estado crítico cuando no se han recibido localizaciones de una mota en la última semana.

2.1.2. Jerarquía de actores

Se han identificado tres perfiles de usuario para la aplicación: *Administrador*, *Investigador* y *Usuario*:

- El *usuario* podrá consultar los datos de las localizaciones de las motas, ver el estado y consultar los parámetros de configuración de las motas.

- El *investigador* podrá configurar los parámetros de las motas e introducir nuevas localizaciones, además de realizar las mismas acciones que el usuario normal.
- El *administrador* podrá gestionar los usuarios y sus permisos y añadir nuevas motas, además de todo lo mencionado anteriormente.

La siguiente figura muestra la jerarquía de actores

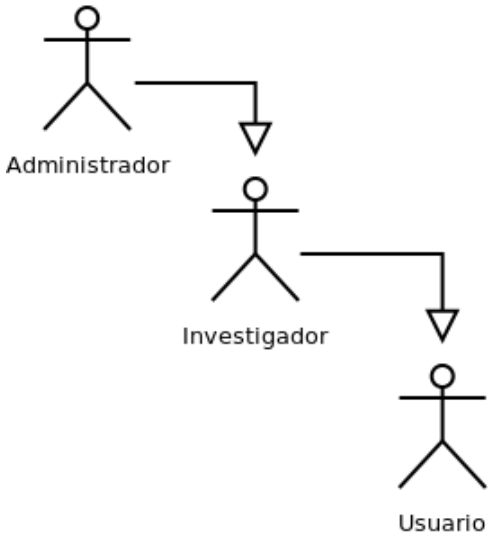

Figura 2.1. Actores

2.1.3. Casos de uso del Usuario

El siguiente diagrama presenta los casos de uso del Usuario. El Usuario va a tener la posibilidad de realizar una serie de tareas con el Sistema: Mostrar las localizaciones en el mapa (**Mostrar localizaciones**), consultar los parámetros de configuración de las motas (**Mostrar configuración**) y ver el estado de las motas (**Mostrar estado**)

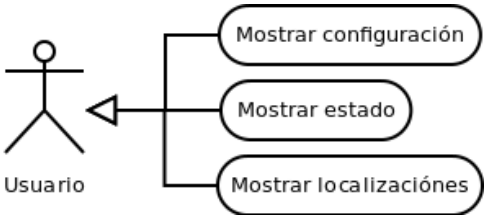

Figura 2.2. Casos de uso del usuario

Caso de uso Mostrar localizaciones

Este caso de uso muestra un mapa y unos parámetros configurables por el usuario.

El usuario puede elegir las motas que desea mostrar en el mapa y un intervalo de fechas, de manera que solo se muestren las localizaciones que cumplan los parámetros introducidos.

Tras seleccionar los parámetros, en el mapa se mostrará el recorrido de cada mota, mostrando información adicional y precisa de cada localización, como el nombre de la mota, la fecha en que fueron tomadas las coordenadas, latitud, altitud y longitud.

Caso de uso Mostrar estado

Este caso de uso muestra el estado del nodo coordinador y una lista de las motas con problemas.

El estado del nodo se basa en el envío de localizaciones, señalándolo con estado crítico si hace más de 24 horas que no se reciben datos.

Las motas problemáticas son aquellas que no tienen datos de localizaciones o que hace más de 24 horas que no envían datos.

Caso de uso Mostrar configuración

Este caso de uso muestra una lista de los parámetros de configuración. Se muestra el nombre descriptivo del parámetro y su valor actual.

2.1.4. Casos de uso del investigador

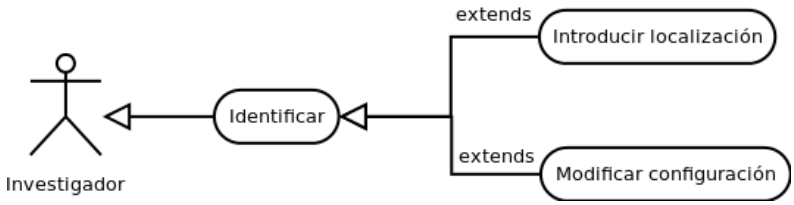

Figura 2.3. Casos de uso del investigador

Caso de uso Identificar

Este caso de uso permite a un usuario registrado identificarse frente al Sistema

El usuario introduce su nombre de usuario y clave. En el caso de una correcta identificación se le proporcionará acceso a distintas opciones asociadas al usuario. En caso de que el usuario no exista o la clave no se corresponda con la del nombre de usuario, se mostrará de nuevo la pantalla de identificación para reintentarlo.

Además, en cualquier momento, el Sistema permite salir de este caso de uso.

Caso de uso Introducir localización

Si el usuario ha sido identificado por el Sistema (Caso de uso Identificar) y el usuario tiene los permisos necesarios, este caso de uso permite introducir una localización mediante el web service, para poder realizar esta acción la mota debe estar dada de alta en el sistema previamente por un Administrador.

Caso de uso Modificar configuración

Si el usuario ha sido identificado por el Sistema (Caso de uso Identificar), el sistema presenta los diferentes parámetros de configuración, mostrando su nombre identificativo y el valor asociado. El Sistema permite modificar estos parámetros.

2.1.5. Casos de uso del administrador

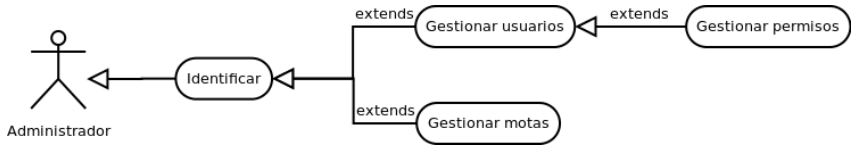

Figura 2.4. Casos de uso del administrador

Caso de uso Gestionar usuarios

Si el Administrador ha sido identificado por el Sistema (Caso de uso Identificar), el Sistema presenta la lista de usuarios, indicando su nombre de usuario y su correo.

El Sistema permite modificar los usuarios, añadir uno nuevo o borrar alguno de los existentes.

Caso de uso Gestionar motas

Si el Administrador ha sido identificado por el Sistema (Caso de uso Identificar), el Sistema presenta la lista de usuarios, indicando su nombre de usuario y su correo.

El Sistema permite modificar los usuarios, añadir uno nuevo o borrar alguno de los existentes.

Caso de uso Gestionar permisos

Si el Administrador ha sido identificado por el Sistema (Caso de uso Identificar), el Sistema presenta la lista de permisos. Estos permisos pueden ser asignados a los distintos usuarios mediante el caso de uso Gestionar usuarios.

Esto nos permite gestionar el acceso a las distintas partes de la aplicación.

2.1.6. Caso de uso completo del sistema

Inicialmente, el administrador tendrá que iniciar sesión en el sistema. Esto le permitirá agregar usuarios, permisos y motas.

Para que las localizaciones de las motas puedan ser introducidas, previamente el administrador debe introducir los identificadores en el sistema. Al hacer esto nos permite identificar las motas propias del sistema, por si el sink ha recogido datos de motas distintas.

Para poder introducir localizaciones, se deberá usar un usuario del tipo administrador o investigador. Uno de estos usuarios será utilizado por el sink, por lo que es recomendable que una vez establecido no se vuelva a modificar, ya que si no habría que recuperar el sink para modificar los datos de conexión. Mediante el uso de este usuario, el sink introducirá las localizaciones en el sistema.

Un investigador podrá identificarse para introducir los datos de configuración de las motas.

Todos los usuarios podrán consultar las localizaciones y el estado de las motas.

2.1.7. Modelo de dominio

Cada **mota** tiene un nombre descriptivo

Las **localizaciones** pertenecen a una mota concreta. Cada localización tiene una fecha y unas coordenadas (latitud, longitud y altitud).

Los **parámetros de configuración** son comunes para todas las motas. Cada parámetro tiene un nombre descriptivo y un valor.

En el siguiente diagrama se muestra el modelo de dominio utilizado. Aparte de los señalados en el diagrama, Django cuenta con un sistema de gestión de permisos y usuarios del que se hará uso también, para que solo determinados usuarios (investigadores y administradores) puedan introducir localizaciones o parámetros de configuración.

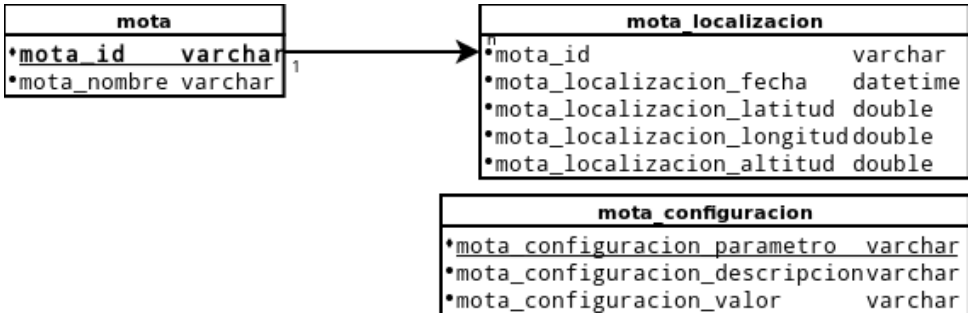

Figura 2.5. Modelo de dominio

2.2. Tecnologías empleadas

Para poder desarrollar el proyecto se han utilizado diversas herramientas que se detallan a continuación. Se especifican las versiones de los programas que serán necesarios en un entorno de producción para hacer funcionar el sistema.

2.2.1. Estructura del sistema

Tras observar que con Python se podían cumplir todos los requisitos indicados, había que estudiar la forma de hacerlo. Al poco de iniciar la investigación se descubrió que hacerlo en Python, iba a ser una

tarea difícil. El uso de mucho código concentrado para mostrar algo simple, repetición de código, etc. Estos problemas supondrían dificultad a la hora de desarrollarlo, pero sobre todo, de seguir desarrollando el proyecto una vez cumplidos los requisitos iniciales. Por ello se ha decidido utilizar un framework. Se trata un conjunto de herramientas, librerías, convenciones y buenas prácticas que pretenden encapsular las tareas repetitivas en módulos genéricos fácilmente reutilizables. Además de ahorrar el trabajo de tener que plantear una estructura y facilitar el mantenimiento, puesto que al estandarizarse es más fácil la modificación del código, esto facilita tanto el trabajo inicial como posteriores modificaciones.

Se ha optado por realizar el sistema cómo una aplicación web por las ventajas que presenta. Tras el estudio de varios frameworks para realizar webs con Python (web2py, Django, TurboGears...) se ha decidido utilizar Django[1].

Las razones para la elección de este framework son las siguientes:

- Actualizaciones constantes: Mientras hay algunos frameworks que no reciben actualizaciones desde hace 4 años, Django tiene un desarrollo activo, por lo que si existen fallos de seguridad o de algún otro tipo, podemos esperar que se encarguen de solucionarlos.
- Patrón MVC: LA separación entre la lógica de los datos y la capa de presentación ya viene definida por el framework, por lo que para futuros desarrollos será más fácil conocer la estructura del proyecto si se ha trabajado con Django anteriormente.
- Además ofrece un sistema de plantillas muy completo, permitiendo introducir en el código HTML la información obtenida mediante Python de manera fácil y cómoda.
- Antigüedad: Esto se traduce en más usuarios y aplicaciones de ejemplo, facilitando la resolución de problemas si surgen. Al estar tan extendido, será más fácil encontrar más desarrolladores que ayuden a ampliar y mantener el proyecto sin necesidad de aprender desde cero un sistema completamente nuevo.
- Documentación y comunidad: Al no conocer ninguno de los sistemas y necesitar una formación, este es uno de los puntos más importantes, ya que sin conocimiento del sistema, no se puede realizar un desarrollo. Al estar tan extendido este framework y con una comunidad tan grande, los problemas que nos podamos encontrar durante el desarrollo seguramente ya hayan sido resueltos por algún otro usuario y, si no, será más fácil encontrar ayuda. Además dispone de una gran cantidad de información de calidad, accesible gratuitamente, como por ejemplo [Django-project] y [Django-book]
- Compatible con distintos tipos de bases de datos, haciendo simple el cambio entre una y otra.

Versión 1.6.2

2.2.2. Sphinx & LaTeX

Herramienta de composición de textos con la que se ha redactado la memoria, usando la plantilla proporcionada por el profesor Roberto Cortiñas.

[1] Concretamente la versión 1.6.2 de Django (https://www.djangoproject.com/)

2.2.3. Dropbox

Herramienta para mantener los contenidos almacenados en la nube, usada como sistema de copia de seguridad.

2.2.4. Dia, GIMP & LibreOffice Draw

Se han utilizado estos tres programas para la generación de todos los diagramas utilizados a lo largo de esta memoria.

2.2.5. Git

Software de control de versiones para la gestión de los cambios sobre el código fuente del proyecto.

Versión: 1.7.9.5

2.2.6. Eclipse

Utilizado como entorno de desarrollo para todo el proyecto, con el plugin PyDev para el desarrollo en Python y el plugin ReST Editor para redactar esta memoria.

2.2.7. Python

Se trata de un lenguaje de programación interpretado, con el que se ha desarrollado la parte del lado del servidor del proyecto.

Versión: 2.7.3

2.2.8. Javascript & jQuery

Al igual que Python, es un lenguaje de programación interpretado. Este ha sido usado en el lado del cliente.

Ha sido combinado con la biblioteca jQuery, que simplifica la manera de interactuar con los documentos HTML y provee unos diseños que ayudan a mejorar la estética de la web con poco esfuerzo.

2.2.9. Apache

Servidor web HTTP para utilizado para servir la página web desarrollada en el proyecto. A pesar de existir otras opciones como *nginx* o *lighthttpd*, se ha elegido utilizar *Apache* porque ya se ha trabajado con él anteriormente. Esto ayuda a poder centrarse en otros apartados del proyecto en lugar del sistema, que posteriormente se podrá utilizar otro que se adapte a las necesidades sin mayor problema.

Versión: 2.2.22

2.2.10. MySQL

Es uno de los sistemas de gestión de bases de datos con los que es compatible Django. Si bien proporciona otras alternativas cómo SQLite, PostgresSQL u Oracle, se ha optado por MySQL debido al mayor conocimiento del sistema por parte del desarrollador, lo que reducirá considerablemente el tiempo de aprendizaje. Una de las desventajas frente a SQLite es que requiere más recursos del sistema, sin embargo, nos proporciona un mayor control de los datos, al permitirnos indicar el tipo de datos que se esperan. Esto es útil por si los datos almacenados en el sink son incorrectos, el sistema los detecta antes de introducirlos, en lugar de almacenar datos incorrectos. Además, nos permite acceso concurrente. SQLite, a pesar de ser un sistema de gestión de bases de datos, almacena los datos en un único fichero, de manera que no se pueden realizar varios accesos de escritura de manera simultanea.

Versión: 5.5.35

2.2.11. Chromium

Navegador web. Al tratarse de una aplicación web, necesitaremos un navegador web para poder visualizar la aplicación y movernos por los distintos apartados de la misma. Se ha elegido este navegador por varias razones:

- Según Wikipedia [2] Chrome cuenta con más de un 40 % de uso entre los distintos navegadores utilizados, y el código fuente de Google Chrome se basa en este navegador.
- Google Chrome está disponible en gran cantidad de plataformas, como Windows, Mac OS X, Ubuntu, Android e iOS.

Versión: 32.0.1700.107

2.2.12. Google Maps

Google Maps es un servidor de aplicaciones de mapas en la web que pertenece a Google. Ofrece imágenes de mapas desplazables, perfecto para mostrar las localizaciones de las motas.

Existen distintas opciones para el sistema dependiendo de la licencia [3] La licencia gratuita obliga a que la aplicación final sea pública y de acceso gratuito. Para la licencia privada habría que consultar las opciones que ofrece *Google Maps for Business*.

Para el prototipo se ha optado por este sistema por su uso extendido y la calidad de la documentación disponible, que hará más sencillo su implementación.

Versión: Google Maps JavaScript API v3

[2] Wikipedia - Navegador Web (http://es.wikipedia.org/wiki/Navegador_web)
[3] Licencia Google Maps (https://developers.google.com/maps/licensing?hl=es)

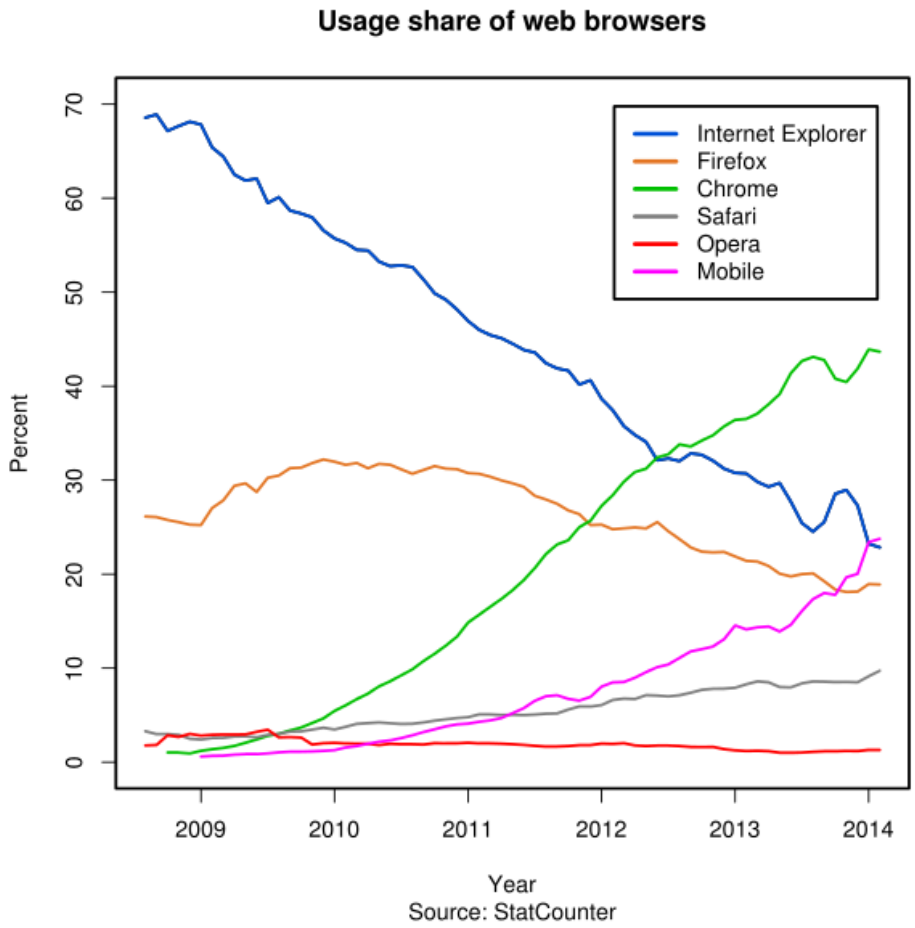

Figura 2.6. Uso de los navegadores

2.3. Formación

La tarea más importante del proyecto, ya que sin una formación adecuada no se puede completar el mismo de manera correcta.

Al estar familiarizado con algunos componentes utilizados en este proyecto, como MySQL, Apache, Javascript, HTML, CSS... se ha dedicado inicialmente un mes de aprendizaje, por el poco conocimiento del lenguaje de programación Python y ninguno del framework Django ni de la biblioteca jQuery. Para el aprendizaje de Python se han utilizado los manuales *Python para todos* [Python-para-todos] y *Programming the Raspberry Pi: Getting Started with Python* [Prog-Rasp], indicados en la bibliografia. Con Django se ha utilizado la página oficial, y el libro *The Definitive Guide to Django* [Guide-to-Django], y su versión web, más actualizada, djangobook, también señalados en la bibliografia. Para jQuery se ha utilizado la web oficial.

También ha sido necesario aprender a usar la API v3 de Google Maps, para mostrar correctamente de la forma deseada los datos solicitados. Para ello, se ha hecho uso de la documentación ofrecida por Google [Google-Maps-Api].

Además se ha necesitado leer la documentación de Sphinx para la creación de esta memoria.

2.4. Motor del sistema y BBDD

En esta tarea iniciaremos la estructura del sistema, con la instalación de Django y la creación de la estructura de la base de datos que se usará en el proyecto.

Al crear el proyecto con Django tendremos una estructura inicial, basada en el patrón MVC, que nos permitirá separar la lógica, los datos y la capa de presentación. Habrá que indicar una serie de parámetros, como el tipo de la base de datos y, en caso de que sea necesario, el usuario y contraseña de la misma. Esto facilita enormemente la posibilidad de elegir entre las distintas bases de datos sin necesidad de cambiar el tipo en distintos ficheros.

Al crear la base de datos inicial, Django nos permite añadir el sistema de administración ofrecido, permitiendo manejar grupos, usuarios y permisos, y administrarlos, de una manera increíblemente sencilla.

Como ya se ha comentado, el proyecto principal del sistema de localizaciones se encuentra en desarrollo. Al no disponer de datos reales, las localizaciones de las motas han sido generadas manualmente. Se ha partido de la base de que una localización cuenta con varios datos clave como son: identificador de la mota, fecha en la que se ha registrado la localización y posición, definida por la altitud, la longitud y la latitud.

Django ofrece un sistema de creación de la estructura de la base de datos a partir de una definición del modelo. Para ello, nos proporciona un fichero, en el que, con Python, le indicamos el formato, que posteriormente Django se encargará de traducir al sistema SQL indicado en la configuración, en nuestro caso, MySQL:

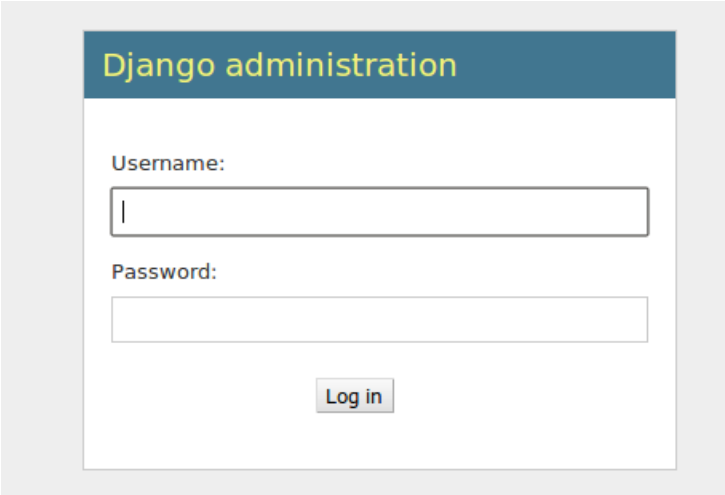

Figura 2.7. Administración de Django

```
from django.db import models

class mota(models.Model):
   '''Informacion de la mota'''
   mota_id = models.CharField(max_length=50, primary_key=True)
   mota_nombre = models.CharField(max_length=100)
   def __unicode__(self):
      return unicode(self.mota_nombre)

class mota_localizacion(models.Model):
   '''Localizaciones de las motas'''
   mota_id = models.ForeignKey(mota)
   mota_localizacion_fecha = models.DateTimeField('location time')
   mota_localizacion_latitud = models.FloatField(max_length=100)
   mota_localizacion_longitud = models.FloatField(max_length=100)
   mota_localizacion_altitud = models.FloatField(max_length=100)
   def __unicode__(self):
      return unicode(self.mota_id)

class mota_configuracion(models.Model):
   '''Parametros de configuracion de las motas'''
   mota_configuracion_parametro = models.CharField(max_length=100, primary_key=True)
   mota_configuracion_descripcion = models.CharField(max_length=100)
   mota_configuracion_valor = models.CharField(max_length=100)
   def __unicode__(self):
      return unicode(self.mota_configuracion_descripcion)
```

Tras actualizar la base de datos con este modelo, tendremos la base de datos especificada con los tipos indicados, como claves, claves foraneas, y sus tipos adecuados.

Una vez realizada la actualización, Django nos permite manejar las bases de datos desde el sistema de administración comentado anteriormente. Con este paso tenemos listo el modelo de datos que se utilizará a lo largo del desarrollo.

Django nos ofrece otra potente funcionalidad, que ahorrará repetir el mismo codigo en distintos

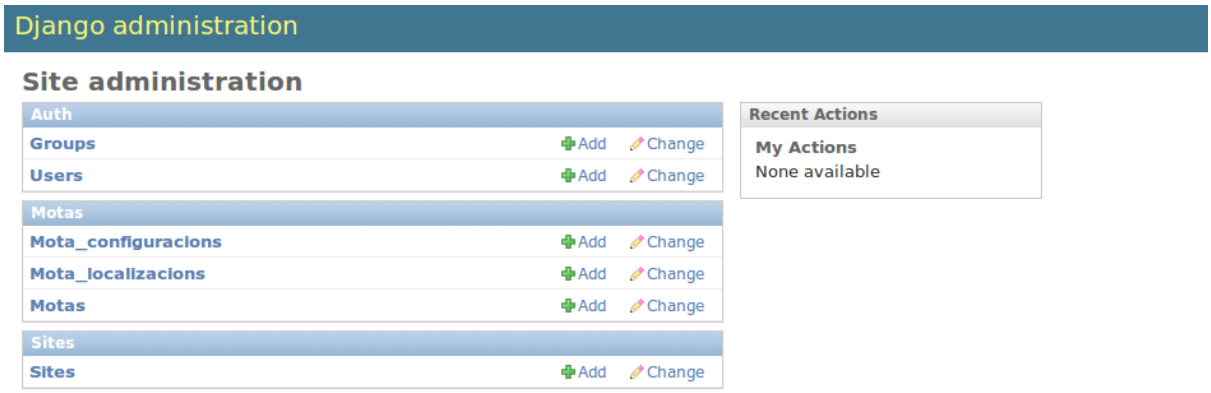

Figura 2.8. Sistema de administración de Django

lugares. Se trata de la herencia de plantillas, que nos permitirá introducir los elementos comunes de las distintas plantillas en un único lugar, en lugar de hacerlo en cada una de las plantillas usadas. Esto también hará más sencillo la extensión del sistema, al no tener que reeditar el menú en todas las plantillas en caso de añadir nuevas secciones.

Se definirá una plantilla base en la que se incluirá el sistema de navegación por las distintas secciones de la página web. Para esto se ha optado por una barra superior que nos redirigirá a los distintos apartados del sistema. Se trata de una barra sencilla, que facilitará el uso y la navegación como se muestra a continuación

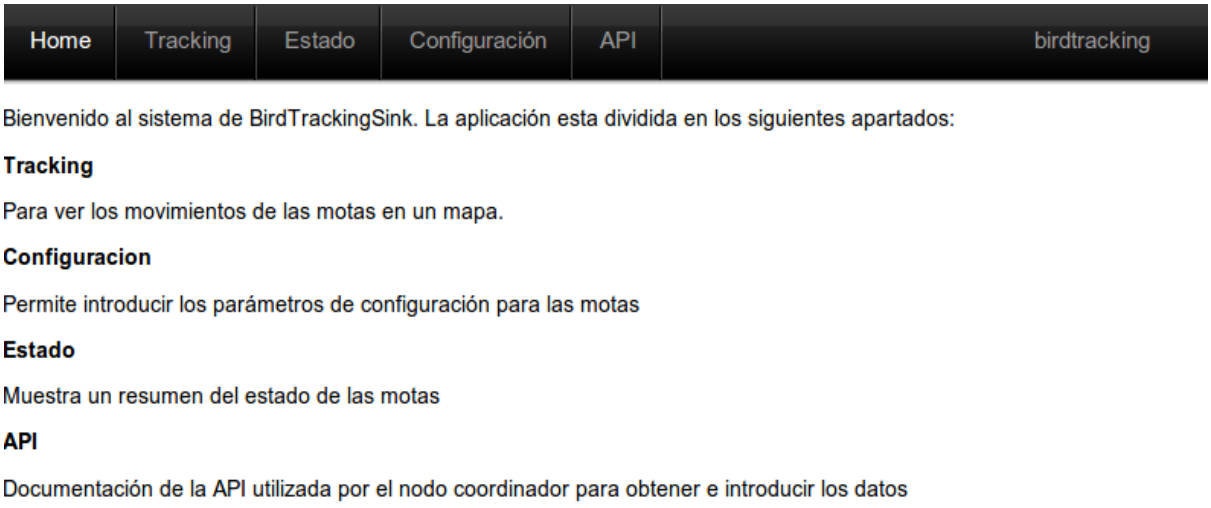

Figura 2.9. Interfaz del sistema BTS

2.5. Tracking

En esta pantalla podremos consultar las localizaciones de las motas.

Se mostrará el recorrido realizado por las motas en un mapa, que estará representado por una flecha indicando la dirección de los desplazamientos. Esto se consigue gracias al registro de hora de las

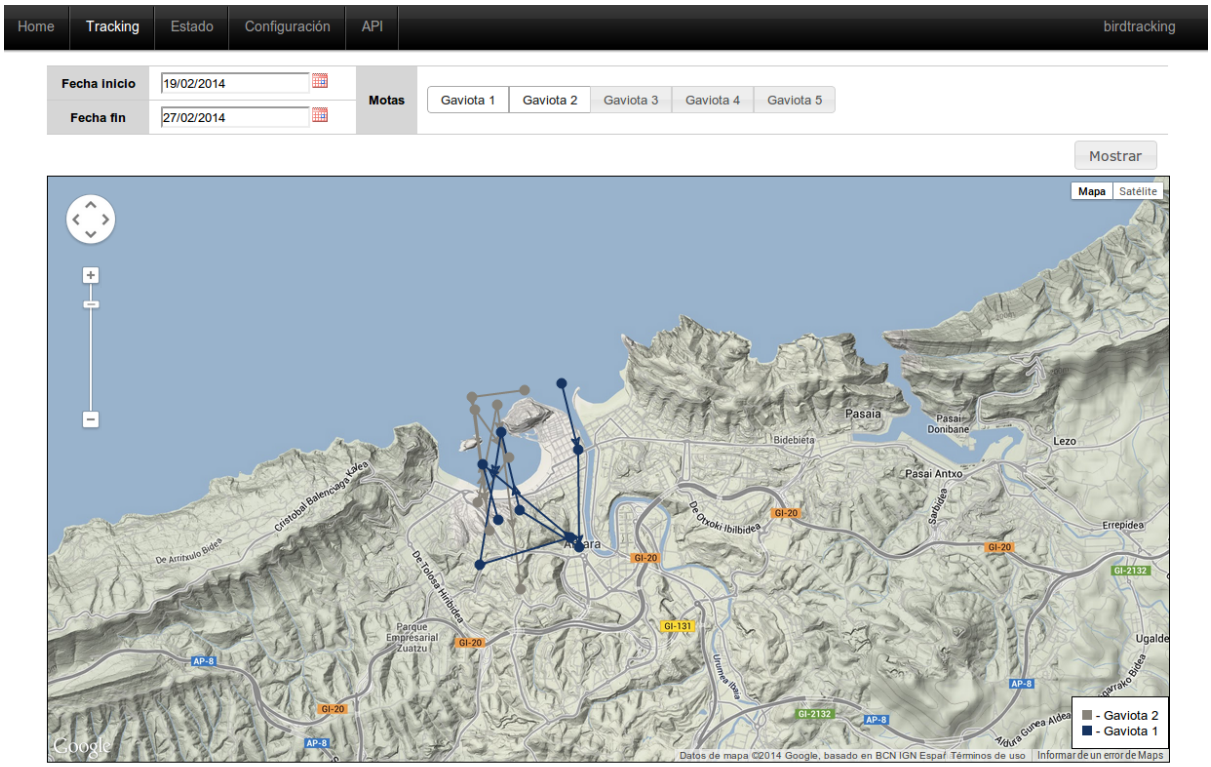

Figura 2.10. Sistema de tracking

localizaciones. Cada registro está señalado individualmente, distinguiendo claramente el recorrido de cada mota mediante el uso de colores para las distintas trazas.

Se puede seleccionar un rango de fechas y unas motas concretas, que facilitará la lectura de los datos para su análisis. Una vez se seleccionan los filtros deseados se mostrará la cantidad de datos disponibles con los datos seleccionados, de manera que se puedan acotar más los filtros en caso de que haya excesivos resultados.

Tras seleccionar los filtros habrá que darle a mostrar para que muestre los datos. Esto se realiza así para evitar la saturación del servidor cuando se están modificando los filtros. De este modo conseguimos que tanto el servidor cómo el cliente no procesen los datos hasta que se tengan bien seleccionados los filtros deseados. Esta opción es importante para cuando la web se consulta desde dispositivos móviles, que suelen tener un límite de uso de datos.

Además, esta pantalla también permite consultar un dato específico sobre una localización de las mostradas.

2.6. Identificación

Para poder acceder a ciertas funcionalidades, primero habrá que identificarse en el sistema. Para ello, bastará con seleccionar la opción de Login en la barra de navegación superior.

Esto nos llevará a una nueva pantalla en la que se nos pedirá un usuario y contraseña.

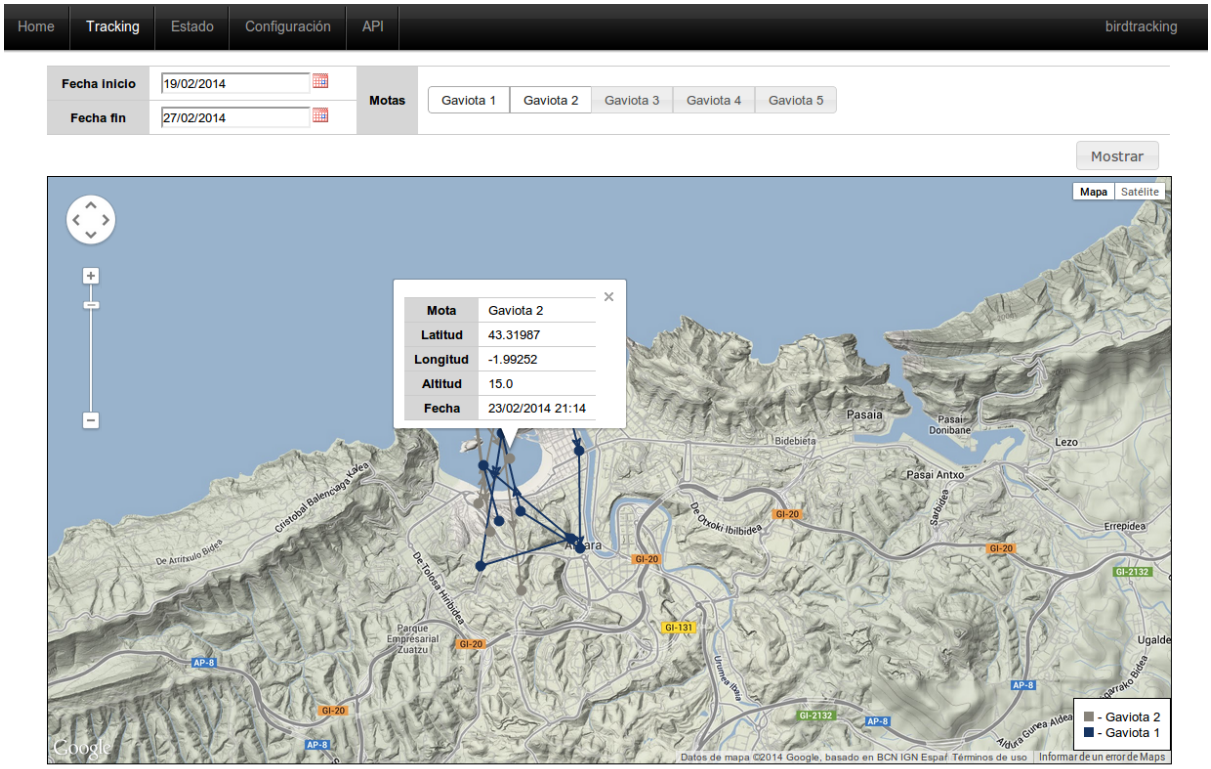

Figura 2.11. Detalles de las localizaciones en el tracking

Figura 2.12. Barra de navegación

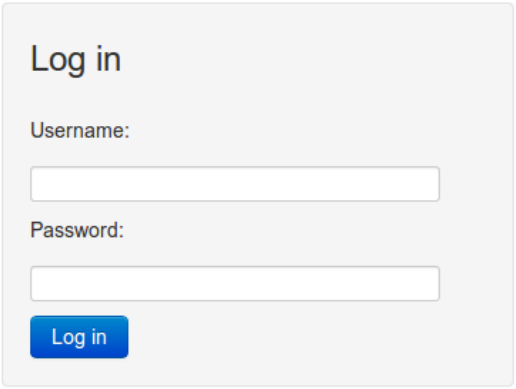

Figura 2.13. Pantalla de Login

Los datos del administrador serán creados durante la instalación del sistema. Con ese usuario se podrán crear nuevos usuarios y dar distintos permisos mediante el sistema de administración comentado anteriormente. Para facilitar el acceso a la pantalla de administración, los administradores dispondrán de la opción de acceso en el menú de login.

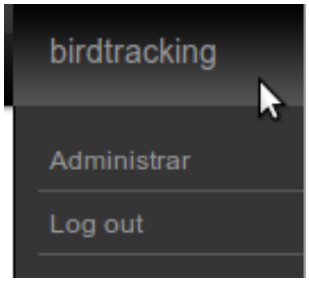

Figura 2.14. Menú de Login

2.7. Parámetros de configuración

En la pantalla de configuración se podrán consultar los parámetros que serán enviados desde el servidor al sink.

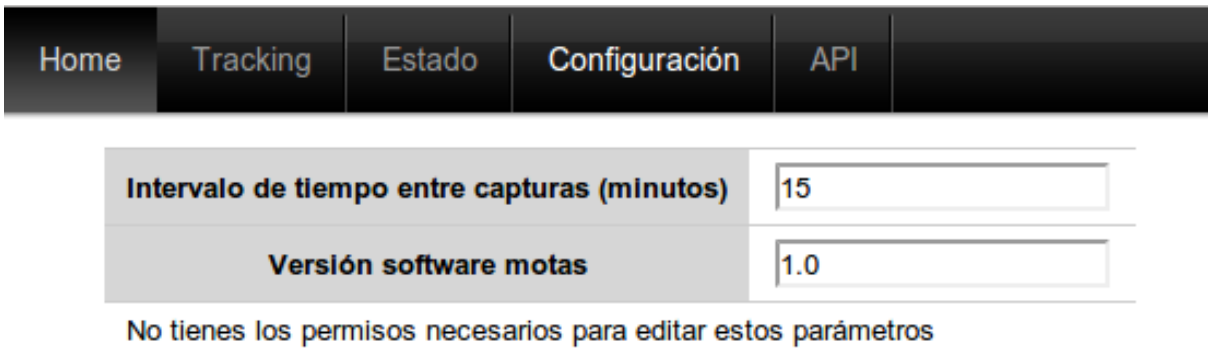

Figura 2.15. Pantalla de configuración para un usuario

Los tipos de parámetros serán introducidos por un administrador, mientras que un investigador podrá modificar los valores. Para ello, en la misma pantalla, en caso de estar identificado con un usuario con los permisos necesarios, permitirá guardar los valores introducidos.

2.8. API REST para Configuración y Localizaciones

En la pantalla de API podremos consultar la documentación del uso de las APIs disponibles.

Gracias a la API se podrán introducir nuevas localizaciones o consultarlas, además de poder consultar también los parámetros de configuración mediante el uso de JSON. Esto nos permite introducir de manera sencilla e instantánea localizaciones en el sistema, lo que nos ofrece la posibilidad de

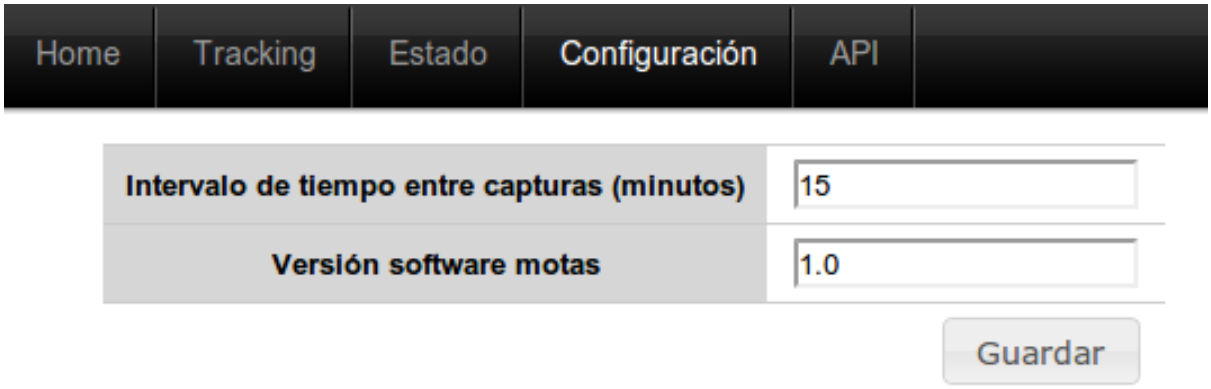

Figura 2.16. Pantalla de configuración para un investigador

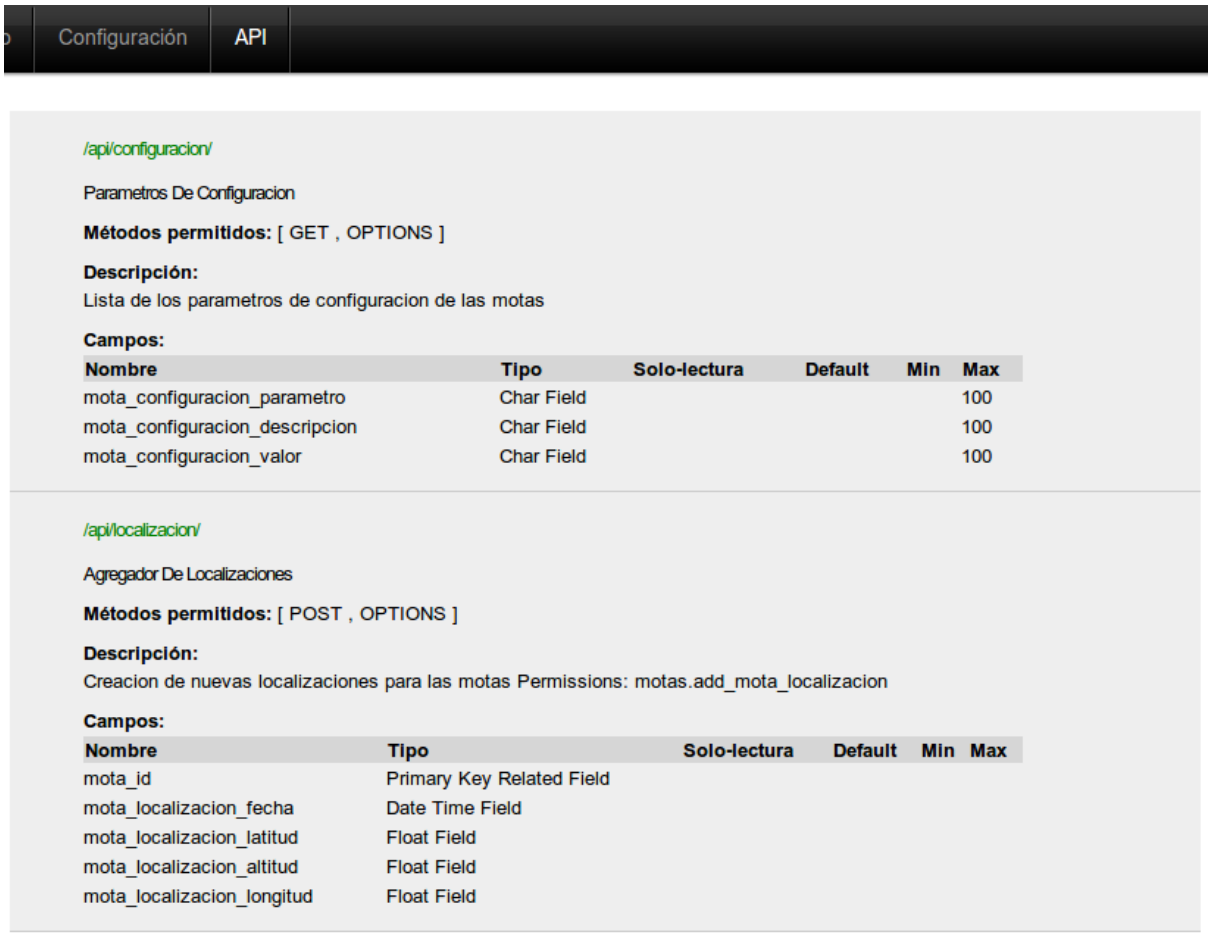

Figura 2.17. Documentación de la API

interactuar con sistemas distintos al propio. Bastará con que el usuario disponga de un usuario con los permisos necesarios y que consulte la documentación ofrecida en ésta página.

El formato JSON se trata de un formato ligero para el intercambio de datos, lo que permite reducir el uso de ancho de banda. Esto, combinado con la simplicidad del formato ha hecho que se generalice mucho su uso.

Ejemplo de localización:

```
{
   "mota_id": "1",
   "mota_localizacion_fecha": "2014-03-01 06:00",
   "mota_localizacion_latitud": "1",
   "mota_localizacion_altitud": "1",
   "mota_localizacion_longitud": "1"
}
```

Ejemplo de parámetro:

```
[
   {
      "mota_configuracion_parametro": "intervalo_minutos_captura_posicion",
      "mota_configuracion_descripcion": "Intervalo de tiempo entre capturas (minutos)",
      "mota_configuracion_valor": "15"
   },
   {
      "mota_configuracion_parametro": "parametro_prueba",
      "mota_configuracion_descripcion": "Parametro prueba",
      "mota_configuracion_valor": "11"
   }
]
```

También nos ofrece la posibilidad de descargar los datos de localizaciones en el formato comentado, de manera que se pueda emplear para el análisis de los mismos de maneras distintas o utilizarlos en otras aplicaciones con acceso al sistema.

2.9. Estado

En la pantalla de estado se muestra un resumen de los problemas aparecidos.

Nos muestra la fecha de la última localización recibida, de manera que se pueda detectar rápidamente si el sink está defectuoso o se recogen pocos datos de las motas, indicando que tal vez habría que desplazar la base a un lugar más concurrido por los ejemplares que se están analizando.

Se muestran también las motas que hace más de una semana que no han enviado datos, que puede ser porque no han pasado cerca del sink, se ha roto o se ha perdido el dispositivo de seguimiento.

También se muestran las motas que no contienen ninguna localización todavía. Esto puede ser porque el administrador ha dado de alta una mota recientemente y todavía no ha recibido localizaciones, se ha introducido un identificador erróneo o no ha visitado el sink en ningún momento.

Figura 2.18. Pantalla de Estado de las motas

Estos datos son útiles si se quiere mantener una cantidad mínima de ejemplares en observación, nos ofrece un sistema rápido de detección de posibles problemas.

2.10. Nodo coordinador

Para el nodo coordinador se han configurado dos tareas periódicas de tipo *cron*, una para el envío de las localizaciones almacenadas en el dispositivo, y otra para obtener los parámetros de configuración de las motas.

Queda fuera del alcance del proyecto el sistema de comunicación entre el nodo y las motas, pero se especifican los requisitos para que esta aplicación funcione correctamente:

Las localizaciones de las motas se almacenarán en un fichero tipo CSV [4] con las siguientes columnas:

1. Identificador de la mota
2. Fecha de la localización, en formato `YYYY-MM-DD HH:mm`
3. Latitud
4. Longitud
5. Altitud

Ejemplo del CSV:

[4] Wikipedia - CSV (http://es.wikipedia.org/wiki/CSV)

```
1,2014-03-08 10:54,43.31829,-1.98338,2
3,2014-03-20 17:28,43.32439,-1.98309,67
3,2014-03-10 23:20,43.30508,-1.99901,28
2,2014-03-18 01:03,43.32352,-1.98932,74
2,2014-03-17 20:47,43.3262,-1.98636,71
3,2014-03-14 17:43,43.30511,-1.98884,21
1,2014-03-17 20:49,43.32845,-1.99718,16
1,2014-03-23 10:53,43.31336,-1.98963,95
2,2014-03-27 16:37,43.31657,-1.99093,43
2,2014-03-25 03:54,43.31055,-1.99705,64
3,2014-03-09 20:30,43.32673,-1.99431,33
3,2014-03-17 11:57,43.31151,-1.99824,85
3,2014-03-12 18:03,43.31022,-1.98939,70
1,2014-03-01 02:19,43.32641,-1.98506,23
1,2014-03-03 20:45,43.31198,-1.99412,26
2,2014-03-12 13:07,43.32988,-1.98122,9
3,2014-03-28 22:12,43.31099,-1.99443,27
1,2014-03-06 22:36,43.3062,-1.98154,92
3,2014-03-10 03:19,43.31306,-1.98013,8
1,2014-03-20 07:07,43.30403,-1.99829,26
```

Los parámetros de configuración se almacenarán en un fichero. Se almacenarán como variable de python para que se pueda hacer un uso directo de estas variables mediante un import, el nombre de estas variables será el del identificador del parámetro en el servidor.

Ejemplo del fichero obtenido:

```
intervalo_minutos_captura_posicion = 15 version_firmware_motas = 1.0
```

Ejemplo de uso en un script Python:

```
#!/usr/bin/env python
import config
print config.intervalo_minutos_captura_posicion
```

Esto imprimirá el valor 15 en nuestro script.

Tanto los ficheros de entrada y de salida, como los datos del servidor (URL, usuario, etc.) serán definidos en un fichero de configuración que se encontrara en el mismo nivel de los scripts que se encargarán de enviar y recibir los datos mencionados.

En la Raspberry Pi contamos con el sistema Cron[5] que nos permite ejecutar unas tareas de manera periódica. Esto nos permitirá ejecutar las tareas de envío de localizaciones y recepción de parámetros de configuración de una manera sencilla.

Ejemplo de tarea cron:

```
* */6  * * * /usr/bin/python /path/to/tareaEnvioLocalizaciones.py
* */12 * * * /usr/bin/python /path/to/tareaObtenerConfiguracion.py
```

[5] Wikipedia - Cron (http://es.wikipedia.org/wiki/Cron_(Unix))

Estos valores habrá que adaptarlos según las necesidades, para enviar los datos cada menos tiempo si fuera necesario. Esto tendrá repercusiones en el consumo de energía del nodo coordinador, por lo que habrá que analizar previamente las posibilidades del sistema de manera que no se interrumpa el servicio ofrecido por el nodo. Este análisis queda fuera del alcance de este proyecto.

El estado de las comunicaciones entre el nodo y el servidor quedarán almacenados en un fichero de log en el nodo coordinador, de manera que en caso de que haya problemas en la comunicación y sea necesario recuperar el dispositivo, se pueda observar el comportamiento que ha tenido y a que se ha debido el fallo de los envíos. El fichero de log tendrá el siguiente formato:

```
2014-02-25 20:52:14,893 - birdtracking - INFO - Iniciando tarea de envio de localizaciones
2014-02-25 20:52:14,969 - birdtracking - INFO - {"detail": "Invalid username/password"}
2014-02-25 20:52:14,976 - birdtracking - INFO - Tarea de envio de localizaciones finalizada correctamente
2014-03-02 10:47:41,008 - birdtracking - INFO - Iniciando tarea de envio de localizaciones
2014-03-02 10:47:41,297 - birdtracking - ERROR - Error en la tarea de envio de localizaciones
2014-03-02 11:02:19,562 - birdtracking - INFO - Tarea de envio de localizaciones finalizada correctamente
2014-03-02 11:07:51,338 - birdtracking - INFO - Iniciando tarea de envio de localizaciones
2014-03-02 11:08:12,151 - birdtracking - INFO - {"detail": "Authentication credentials were not provided."}
```

3. CAPÍTULO

Resultados, Conclusiones y Líneas Futuras

3.1. Resultados

Una vez finalizado el proyecto, podemos dar por completados los objetivos propuestos inicialmente.

- Se ha desarrollado un servicio web que ofrece la información recogida por las motas. Permite analizar los resultados de manera cómoda gracias a la opción de establecer filtros, para obtener datos de algunas motas en concreto y unas fechas concretas.
- Se proporciona información sobre posibles fallos en la recogida de datos cuando se detecta un tiempo de inactividad de alguna de las motas.
- Los datos se pueden obtener de una manera rápida, sin necesidad de intervención humana: no se necesita recoger los dispositivos ni transferir manualmente los datos del sink al servidor.
- Las localizaciones pueden ubicarse en cualquier parte del mundo gracias a la tecnología de Google Maps.
- Se pueden enviar datos a los sink desde el servidor.

3.2. Conclusiones

Se han alcanzado todos los objetivos planteados en el alcance del proyecto correctamente.

Han surgido problemas durante el desarrollo que han impedido que la planificación inicial se haya podido cumplir. La dedicación constante propuesta inicialmente no se ha podido mantener durante todo el desarrollo. Al haber planificado con tiempo suficiente y unos márgenes adecuados, esto no ha supuesto ningún problema para terminar el proyecto antes de la fecha límite.

Se han planteado funcionalidades que inicialmente no estaban pensadas. Al no tratarse de grandes cambios y, gracias a las metodologias ágiles, estos cambios se han podido incluir en el desarrollo del proyecto sin afectar a la fecha de entrega.

3.3. Líneas Futuras

3.3.1. Añadir mapa propio

Una de las alternativas estudiadas al comienzo de este proyecto ha sido la posibilidad de crear un mapa propio. Debido a la limitación temporal no era viable incluir esta opción en este desarrollo. Debería considerarse hacer este cambio, por si en un futuro Google decidiera dejar de ofrecer el servicio de Gooogle Maps, poder seguir utilizando el sistema de tracking, y no tener una dependencia externa en una parte tan importante de la aplicación.

3.3.2. Añadir otros nodos coordinadores

Ahora mismo, el sistema permite que haya varios nodos coordinadores, pero está diseñado para monitorizar el estado como si solo existiera uno. Al añadir mas nodos coordinadores se abarcarían más zonas, por lo que habría que pensar en dividir el mapa por zonas, o permitir seleccionar por cercania de los nodos coordinadores las partes del mapa. Además, también se podría añadir información del nodo que ha enviado cada localización, para conocer la zona en la que se encuentran las motas en cada momento.

3.3.3. Analizar la información recibida

Se podrían añadir funcionalidades para que la información se muestre más conclusiva, como puede ser colorear las zonas más o menos visitadas, momentos en los que no se han tomado valores en los intervalos esperados, por ejemplo, porque no tenía batería el dispositivo, y así analizar automáticamente la eficiencia de los dispositivos.

Glosario

GSM El sistema global para las comunicaciones móviles es un sistema estándar de telefonía móvil digital.

CSS Es un lenguaje de hojas de estilos usado para describir la presentación de un documento escrito en lenguaje de marcas como HTML, XHTML o XML.

Sink Se trata del nodo que recopila los datos, generalmente un PC

GUI Graphical User Interface: Interfaz gráfica de usuario

HTML (HyperText Markup Language) hace referencia al lenguaje de marcado para la elaboración de páginas web. Es un estándar que, en sus diferentes versiones, define una estructura básica y un código (denominado código HTML) para la definición de contenido de una página web, como texto, imágenes, etc.

Mota Son los sensores que recopilan información del entorno, que posteriormente es descargada sobre el Sink

MVC Patrón de programación Modelo-Vista-Controlador, es un patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario.

Python Es un lenguaje de programación interpretado. Posee una licencia de código abierto y es multiplataforma.

Web API (Application Programming Interface) Lista de métodos y propiedades de una implementación para permitir realizar alguna acción o acceder a alguna característica o contenido que proveen los sitios.

WSN Wireless Sensor Network (Redes de Sensores). Son redes que están formadas por una serie de pequeños dispositivos electrónicos (Motas).

4. CAPÍTULO

Anexo A: Configuración del servidor

En este anexo se detallan las instrucciones para poner en marcha la aplicación sobre un servidor.

4.1. Versiones probadas

Aunque no es necesario que el sistema utilice las versiones de los programas aquí indicadas, se detallan por razones de compatibilidad.

- **Sistema operativo**: La aplicación ha sido probada sobre entornos Linux con las distribuciones Raspbian [1] , basada en Debian Wheezy (Debian 7.0) y Ubuntu 12.04 [2] . Ambas en su versión estable.
- **Servidor web**: Apache v2.2.22
- **Servidor BBDD**: MySQL v5.5.31 / Sqlite 3.8.2
- **Navegador web**: Chromium v34.0.1847.116 / Firefox v24.0

4.2. Instalación

4.2.1. Software necesario

- Ubuntu 12.04
- Python 2.7.3
- Django 1.6.2

[1] Raspbian (http://www.raspbian.org/)
[2] Ubuntu (http://www.ubuntu.com/)

```
$ sudo apt-get install python python-pip
$ sudo pip install django
```

Dependencias del proyecto:

```
$ sudo pip install djangorestframework
$ sudo pip install django-rest-framework-docs
```

En caso de usar mysql

```
$ sudo apt-get install mysql-server python-mysqldb
```

Crear una base de datos y un usuario con todos los privilegios sobre ella

---

Por hacer

mysql -h localhost -u root -p

```
CREATE DATABASE bird_tracking_sink CHARACTER SET UTF8;
GRANT ALL PRIVILEGES ON bird_tracking_sink.* TO 'birdtracking'@'localhost' IDENTIFIED BY 'password' WITH GRANT OPTI
```

Añadir los datos al fichero `settings.py` en la carpeta bts dentro del proyecto:

```
DATABASES = {
    'default': {
      'ENGINE': 'django.db.backends.mysql',
      'NAME': 'bird_tracking_sink',
      'USER': 'birdtracking',
      'PASSWORD': 'password',
      'HOST': '',
      'PORT': '',
    }
}
```

En la carpeta del proyecto para inicializar la base de datos:

```
python manage.py syncdb
```

Con este comando crearemos un superusuario en la aplicación que nos servirá para crear nuevos usuarios, dar de alta motas, etc.

Modificar en `settings.py` los parametros:

- MEDIA_ROOT
- STATIC_ROOT
- STATICFILES_DIRS
- TEMPLATE_DIRS

Para usar el servidor proporcionado por Django, en `settings.py` poner los parámetros:

```
DEBUG          = True
TEMPLATE_DEBUG= DEBUG
```

En la carpeta del proyecto:

```
python manage.py runserver 0.0.0.0:8080
```

Configuración con apache:

```
sudo apt-get install apache2 libapache2-mod-wsgi
```

En `settings.py` poner los parámetros:

- DEBUG = False
- Eliminar parametro TEMPLATE_DEBUG

Editar path del script WSGI que se encuentra en la carpeta apache del proyecto bts:

```
import os
import sys

path = '/home/pi/bts/birdtrackingsink/birdtrackingsink/bts'
if path not in sys.path:
   sys.path.append(path)

os.environ['DJANGO_SETTINGS_MODULE'] = 'bts.settings'

import django.core.handlers.wsgi
application = django.core.handlers.wsgi.WSGIHandler()
```

Modificar configuracion apache editando el fichero `/etc/apache2/sites-available/bts`:

1. Modificar `DocumentRoot` para que apunte a la carpeta del proyecto *bts*
2. Modificar path de `WSGIScriptAlias` para que apunte al script del punto anterior
3. Modificar path del `Alias /static` para que apunte a la carpeta `static` del proyecto *bts*

```
<VirtualHost *:8080>
  ServerName bts
  DocumentRoot /home/pi/bts/birdtrackingsink/birdtrackingsink/bts

  WSGIScriptAlias / /home/pi/bts/birdtrackingsink/birdtrackingsink/bts/apache/django.wsgi
  WSGIPassAuthorization On

  <Directory /home/pi/bts/birdtrackingsink/birdtrackingsink/bts>
      Order allow,deny
      Allow from all
  </Directory>

  Alias /static /home/pi/bts/birdtrackingsink/birdtrackingsink/bts/static/
</VirtualHost>
```

```
$ sudo a2ensite bts
$ sudo echo "Listen 8080" >> /etc/apache2/ports.conf
$ sudo service apache2 restart
```

Entrando en http://localhost:8080/admin con el usuario creado anteriormente podremos gestionar la base de datos.

4.3. Tarea cliente

4.3.1. Envío localizaciones

Modificar los parametros servidor, usuario, password, path_directorio_localizaciones y nombre_fichero_localizaciones del archivo de la carpeta cliente settings.py

Ejemplo:

```
servidor = 'http://xacosta.servequake.com:8080'
usuario = 'birdtracking'
password = '1234'
path_directorio_localizaciones = '/home/acosta/git/bts/birdtrackingsink/cliente/'
nombre_fichero_localizaciones = 'prueba'
```

En path_directorio_localizaciones indicar la carpeta donde se encuentra el CSV de las localizaciones y en nombre_fichero_localizaciones el nombre del fichero, que debe tener las siguientes columnas:

1. Identificador de la mota
2. Fecha de la localización, en formato YYYY-MM-DD HH:mm
3. Latitud
4. Longitud
5. Altitud

Ejemplo: **1,2014-01-05 12:00,43.32165,-1.98497,15**

Ejecución manual:

```
python /path/to/tareaEnvioLocalizaciones.py
```

Cron para enviar los datos cada 6 horas:

```
* */6  * * * /usr/bin/python /path/to/tareaEnvioLocalizaciones.py
```

Bibliografía

[Django-book] Adrian Holovaty, Jacob Kaplan-Moss, et al.(2012). *The Django book*. http://www.djangobook.com/

[Guide-to-Django] Adrian Holovaty, Jacob Kaplan-Moss *The Definitive Guide to Django: Web Development Done Right*.

[Sphinx-Docu] Georg Brandl. *Sphinx documentation*. http://sphinx-doc.org/. The Pocoo Team, Python community.

[Google-Maps-Api] Google Team. *Google Maps API v3.0*. https://developers.google.com/maps/?hl=es. Google Inc.

[Python-para-todos] Raúl González Duque. *Python para todos*

[Django-REST-framework] Django REST framework http://www.django-rest-framework.org/

[Django-project] Django Project https://www.djangoproject.com/

[Prog-Rasp] Simon Monk *Programming the Raspberry Pi: Getting Started with Python*.

[jQuery] jQuery API http://api.jquery.com/

[BoutenBSC13] Bouten, E. Baaij, J. Shamoun-Baranes, K. C. J. Camphuysen. *Wireless mesh networks: a survey*. Ornithology, 2013

[UvA] *UvA Bird Tracking System*. Universidad de Amsterdam. http://www.uva-bits.nl

[Telemetry] Microwave Telemetry Inc. http://microwavetelemetry.com

[TecnicasMarcaje] Juan Bécares, Beneharo Rodríguez, José Manuel Arcos, Asunción Ruiz *Técnicas de marcaje de aves marinas para el seguimiento remoto*. Revista de anillamiento 12/2010; 25-26:29-40.

Índice