Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Kune – Calzado conectado

Trabajo Fin de Máster

Máster Universitario en Ingeniería Informática

Autor: Juan Iscar Martinez

Tutor: Vicente Pelechano Ferragud

Tutor: Sergio Milla Montero

Curso académico 2016/2017

Agradecimientos

Deseo agradecer la importante labor dedicada por mi tutor de empresa Sergio Milla, el cual ha estado apoyándome en todo momento y enseñándome a realizar trabajos profesionales. También destaco la oportunidad que me ha ofrecido Vicente Pelechano por ser mi tutor docente y haber confiado en mi proyecto.

El proyecto Kune ha sido desarrollado gracias a los programas de ayuda de innovación social del Ayuntamiento de Valencia.

Resumen

El presente proyecto de final de master es una memoria del trabajo realizado en la empresa Wonkacenter, el cual pretende cubrir una necesidad en el mercado del cuidado de pacientes. La memoria abarca el trabajo realizado por el departamento software, el cual ha diseñado, programado y probado todo el sistema.

El proyecto, llamado “Kune - Calzado conectado”, tiene como objetivo el desarrollo de una aplicación que facilite el cuidado de los pacientes por parte de sus familiares, especialmente los afectados por el Alzheimer. En el caso de que un paciente salga solo a la calle, los familiares podrán conocer su ubicación a través de la aplicación desarrollada e ir a buscarle de ser necesario.

Desde el primer momento se ha pensado utilizar tecnologías avanzadas y profesionales, como lo son Nodejs y Meteor, almacenando los datos en Mongodb, y adaptándose a distintos soportes, como dispositivos móviles y aplicaciones por navegador.

La empresa ya cuenta con experiencia en productos similares y se espera que el producto resultante sea un referente en el ámbito de la localización y del desarrollo software.

Palabras clave: Kune, Nodejs, Meteor, Mongodb, React, Sigfox, Android.

Abstract

This final master project is a memoir of the work done at Wonkacenter, which aims to cover a need in the patient care market. The memory covers the work done by the software department, which has designed, programmed and tested the entire system.

The project, called "Kune - Connected Footwear", aims to develop an application that facilitates the care of patients by their relatives, especially those affected by Alzheimer's. In the event that a patient leaves the street alone, the family members will be able to know their location through the application developed and go to look for him if necessary.

From the very beginning, it has been thought to use advanced and professional technologies, such as Nodejs and Meteor, storing the data in Mongodb, and adapting to different media, such as mobile devices and browser applications.

The company already has experience in similar products and the resulting product is expected to be a benchmark in the field of localization and software development.

Keywords: Kune, Nodejs, Meteor, Mongodb, React, React, Sigfox, Android.

Tabla de contenidos

Índice de Ilustraciones.

Índice de tablas.

1. Introducción

El presente trabajo de final de master describe el sistema producido por el alumno para la empresa Wonkacenter, detallando sobre el producto, las funciones de que dispone, requisitos y todo lo relacionado con su desarrollo. El resultado será un producto comercializable y abierto a cambios.

1.1. Motivación

El cuidado de los pacientes con Alzheimer es un problema actual, en el que lo familiares tienen un importante papel activo. Los pacientes con esta enfermedad, concretamente en su segunda etapa, mantienen su movilidad, pero tiene casos de perdida de orientación. Si esta pérdida de orientación ocurre en la calle, a el cuidador le resultaría extremadamente complicado localizarlo de nuevo.

Según la revista 'Neurology' los casos de Alzheimer pueden triplicare en 40 años a causa del envejecimiento de la población. Por lo que se pretende que esta aplicación sea un referente en el campo del cuidado de pacientes, y ayude a todas aquellas personas que se vean afectadas.

Los cuidadores de este perfil de pacientes no siempre pueden permanecer constantemente con estos, por lo que existen casos en el que los pacientes deban de quedarse solos.

Esta problemática crea un mercado de productos que faciliten la vida tanto de cuidadores como de los pacientes, dándole más libertad a cada uno, la empresa Wonkacenter[1] se adelanta a cubrir esta necesidad con el producto "Kune- Calzado conectado".

El sistema de "Kune – Calzado conectado" pretende cubrir la necesidad de que el cuidador pueda encontrar a los pacientes que tiene a cargo remotamente, ofreciéndole más libertad para otras actividades, librándole de una presencia constante con sus pacientes. La aplicación también puede ser usada para pacientes con síntomas similares.

En la actualidad, el uso de un teléfono móvil personal es una práctica generalizada que ofrece una gran cantidad de oportunidades comerciales y de gestión, "Kune – Calzado conectado" también dispondrá de una aplicación móvil con la que el cuidador será capaz de monitorizar sus pacientes más cómodamente.

[1] http://wonkacenter.com/

Este producto puede, en un futuro extrapolarse a otros ámbitos y cubrir otros problemas con necesidades similares.

1.2. Estructura

El presente trabajo se encuentra dividido en 8 apartados, los cuales se describen a continuación, la estructura está basada en el estándar IEEE 830-1985, al a que se le han incluido nuevos puntos para describir otros aspectos relevantes al producto.

1. **Introducción.**
   Consiste en el primer apartado de la memoria y será destinado a introducir al lector en el contexto del mismo, se expondrán los objetivos del mismo y su estructura.

2. **Descripción del sistema.**
   En este apartado se describe superficialmente el sistema de "Kune – Calzado conectado" con el fin de comprender más adelante especificaciones más técnicas.

3. **Descripción general.**
   Este capítulo es una continuación más detallada del anterior, en él se muestran las dependencias de "Kune – Calzado conectado" con otros sistemas y productos, los perfiles de los usuarios, las funciones que realiza, restricciones que presenta y posibles cambios futuros.

4. **Requisitos específicos.**
   A continuación, se detalla muy detenidamente los requisitos de diseño, las funciones que realiza y todos los demás aspectos para el correcto funcionamiento del sistema.

5. **Estudio de mercado.**
   En este punto se indica que productos similares existen en el mercado y cuáles son los elementos diferenciadores de "Kune – Calzado conectado" que le aportan valor.

6. **Diseño técnico.**
   En este punto se describe detalladamente la estructura del sistema, que capas lo componen y como funciona cada capa.

7. **Pruebas.**
   Una vez finalizado el desarrollo, se procede a describir como se han realizado las pruebas, en que orden y los resultados obtenidos con el fin de mejorar la calidad del software.

8. **Conclusión y trabajo futuro.**

Finalmente se expondrán las conclusiones extraídas del análisis y desarrollo del proyecto, también se realizarían propuestas para la continuación del mismo y otras mejoras.

Al final de este documento se muestra una serie de anexos con el fin de concretar aspectos del desarrollo, toma de decisiones y otras anotaciones.

Finalmente se incluye una bibliografía detallando autores, las fuentes y fechas de consulta según el estándar IEEE.

1.3. Objetivos

El objetivo final de todo el trabajo es completar un producto de calidad para la empresa Wonkacenter, el proyecto se encuentra dividido en una serie de capas que serán producidas secuencialmente, cada una será considerada como un hito y un objetivo. El último objetivo es verificar la correcta funcionalidad del sistema ejecutando todas sus capas.

Los objetivos principales consisten en:

- Seleccionar la tecnología adecuada para cada elemento.
- Desarrollar una API REST junto con su base de datos.
- Desarrollar una Web App.
- Desarrollar una aplicación móvil.
- Probar el conjunto del sistema.

Durante el desarrollo se debe de dotar al sistema de los siguientes requisitos de funcionalidad:

- Funcionamiento intuitivo, con el fin de llegar al máximo publico posible sin necesidad de formación.
- Simplicidad, porque un sistema simple reduce el coste de mantenimiento y sus errores.
- Fiable, el cliente cofia en nuestro producto y proporcionar ubicaciones erróneas de pacientes puede conllevar problemas y pérdida de confianza.

Adicionalmente es necesario para el futuro del sistema redactar una correcta y completa documentación del sistema para que sea posible realizar cambios por algún otro desarrollador futuro.

Como objetivo secundario:

- Desarrollar documentación técnica.

2. Descripción del sistema

Las características, requerimientos y limitaciones que debe cumplir la aplicación, se desarrollarán en este apartado siguiendo el estándar IEEE 830-1998[2], que indica como redactar un documento ERS, proporcionando un esquema cercano y fácilmente comprensible de cara al usuario.

2.1. Propósito

El fin del presente ERS es definir al lector todos los datos relacionados con el desarrollo de la aplicación, el uso por parte de todos los roles de usuarios y futuras ampliaciones. Se expresará claramente a quien va dirigido, con que fin, que es lo que el sistema será capaz de hacer y de lo que no.

2.2. Ámbito del sistema

La aplicación descrita en el presente documento lleva el nombre de “Kune – Calzado conectado”, se le ha bautizado así debido a la similitud léxica de la palabra “Kune” con “une” con el objetivo de trasmitir una sensación de unión al usuario, en este caso, de los cuidadores con sus pacientes. El subtítulo de “Calzado conectado” hace referencia al producto físico que acompaña al sistema, en este caso un calzado con chips de localización incorporados.

El fin del sistema es otorgar a los cuidadores la posibilidad de conocer la ubicación de los pacientes en tiempo real, ya sea desde un ordenador o desde un teléfono móvil. El sistema también tendrá la capacidad de enviar notificaciones de actividad del paciente hacia el móvil del cuidador si este lleva la aplicación de nuestro sistema instalado.

Cada cuidador solo puede localizar a sus propios pacientes.

Los pacientes serán registrados en el sistema por parte de sus cuidadores, pero lo dispositivos serán entregados y registrados por la empresa propietaria de este sistema.

Adicionalmente, la aplicación móvil dispondrá de un pequeño entretenimiento para que cuidadores y pacientes puedan compartir una actividad.

La aplicación no será capaz de interactuar de ninguna forma con los pacientes ni de limitar su actividad

Los beneficios de la aplicación hacia sus usuarios son la tranquilidad de conocer la ubicación de los pacientes debido a que muchas veces estos son familiares y no pueden estar presencialmente con ellos constantemente.

[2] https://standards.ieee.org/findstds/standard/830-1998.html

El beneficio que obtendrán los pacientes será, además de poder ser encontrados si estos se extravían y la seguridad que ellos mismos pueden sentir, es la comodidad de llevar un dispositivo localizador no visible. Las personas que portan dispositivos de este tipo como identificadores, botones de alarma, localizadores…; suelen desprenderse de ellos por ser grandes, vistosos y no los reconocen como elementos propios, en parte porque vienen en forma de pulseras y collares, el dispositivo localizador de "Kune – Calzado conectado" viene en forma de calzado, por lo que resulta muy difícil que el paciente lo rechace.

2.3. Definiciones, Acrónimos y Abreviaturas

Web App: Abreviatura de "Aplicación Web", representa un panel de gestión el cuan se accede a través de un navegador Web, en el caso de "Kune- calzado conectado" es un panel capaz de listar los pacientes del usuario y localizarlos.

Paciente: Rol de "Kune- calzado conectado", representa a un paciente parcialmente dependiente, el cual es capaz de desplazarse, pero necesita supervisión. Es el usuario del calzado localizador.

Cuidador: Rol de "Kune- calzado conectado", es el encargado de cuidar a los pacientes y por tanto capaz de conocer su ubicación a través de la Web App o de la aplicación móvil.

Aplicación móvil: Programa que es ejecutado desde un teléfono móvil, en el caso de "Kune- calzado conectado", permite autenticar al cuidador, listar y ubicar a los pacientes del cuidador autenticado. Además, es capaz de recibir notificaciones informando de las acciones de los pacientes.

Calzado localizador: Prenda usada por los pacientes, son capaces de enviar al sistema la ubicación de estos.

Nodejs: Entorno de ejecución usado para la API REST del sistema.

API REST: Arquitectura para la transferencia de información, en "Kune- calzado conectado" es el servicio que recibe los mensajes de la red de Sigfox y se comunica con la Aplicación móvil.

Red Sigfox: Red de comunicaciones propiedad de Sigfox a través de la cual, el Calzado localizador envía los mensajes de posición a la API REST.

PHP: Lenguaje de programación usado para emular la red de Sigfox y realizar pruebas.

MySQL: Base de datos relacional utilizado para emular la red de Sigfox.

Mongodb: Base de datos no relacional utilizado para la API REST y la Web App. Contiene toda la información del sistema.

3. Descripción General

Para el correcto funcionamiento del sistema y maximizar la experiencia del usuario es necesario utilizar servicios externos como Sigfox o Google, y se adecue el perfil de los usuarios al que va destinado, descritos en el presente apartado.

3.1. Perspectiva del producto

En el presente apartado del ERS define las dependencias de "Kune – Calzado conectado" con otros productos y sistemas.

La aplicación "Kune – Calzado conectado" consta de diversas capas, una de ellas es la red de comunicaciones Sigfox, Sigfox es una compañía emergente con una red de conectividad presente ya en muchos países de Europa.

Otra capa presente en el sistema es el calzado localizador, el cual lleva integrado un chip programable con capacidades de comunicación y localización fabricado por la compañía Telecom Design[3], presente en Francia y España. El software del chip es tecnología propia de la empresa Wonkacenter.

También es usado los servicios de Google, en concreto la API de Google maps[4] y Firebase[5].

Google maps es un servicio de librerías para interfaces, que muestra un mapa interactivo al usuario, el cual está disponible en varias plataformas: Navegadores, dispositivos móviles Android e IOs. En "Kune – Calzado conectado" es usado para mostrar los mapas con las posiciones de los pacientes a los usuarios.

Firebase es un servicio de mensajería entre aplicaciones usado ampliamente para difusión y notificaciones en dispositivos móviles. En nuestro sistema es empleado para el envío de alertas entre la API REST con la aplicación móvil.

[3] http://www.telecom-design.com/
[4] https://developers.google.com/maps
[5] https://firebase.google.com/

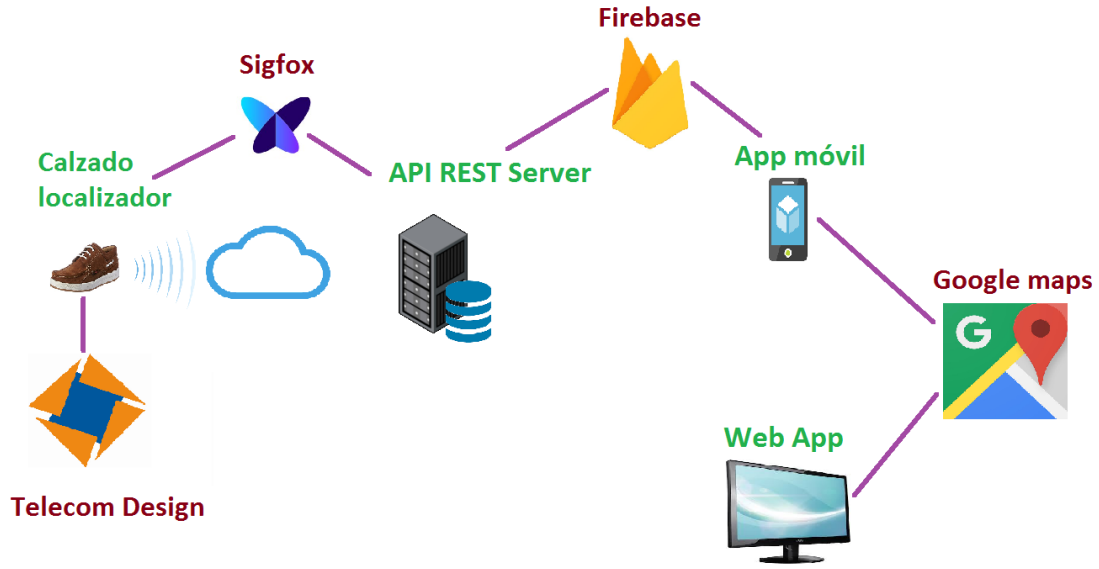

Ilustración 1. Productos externos en el sistema.

3.2. Funciones del producto

Las actividades principales que realiza el cuidador sobre la aplicación son 3, tanto para la aplicación móvil como para la Web App:

Listar los pacientes, que mostrará al cuidador una lista con los pacientes a su cargo, sobre cada paciente de la lista se pueden realizar las operaciones de ver la ficha del paciente y ver su posición. De cada paciente se muestra su nombre y el resto de opciones disponibles.

Posición del paciente, acción que consulta la ubicación del paciente seleccionado y es mostrado al cuidador a través de un mapa.

Ficha del paciente, es la opción que permite visualizar los detalles de un paciente seleccionado.

Adicionalmente, la Web App tiene funciones adicionales, como la creación de un nuevo paciente, eliminar paciente ya creado y editar sus datos.

Crear nuevo paciente, Se crea un nuevo paciente a partir del nombre y DNI, el resto de datos se mantiene a su valor por defecto.

Eliminar paciente, a partir de la lista de pacientes, cada uno tiene la opción de eliminar que lo borrará permanentemente de la base de datos y no se podrá volver a localizar.

Editar paciente, lo campos mostrados en la ficha de paciente pueden ser editados, estos cambios son enviados al servidor. En este punto se indica que Calzado localizador se asigna a cada paciente.

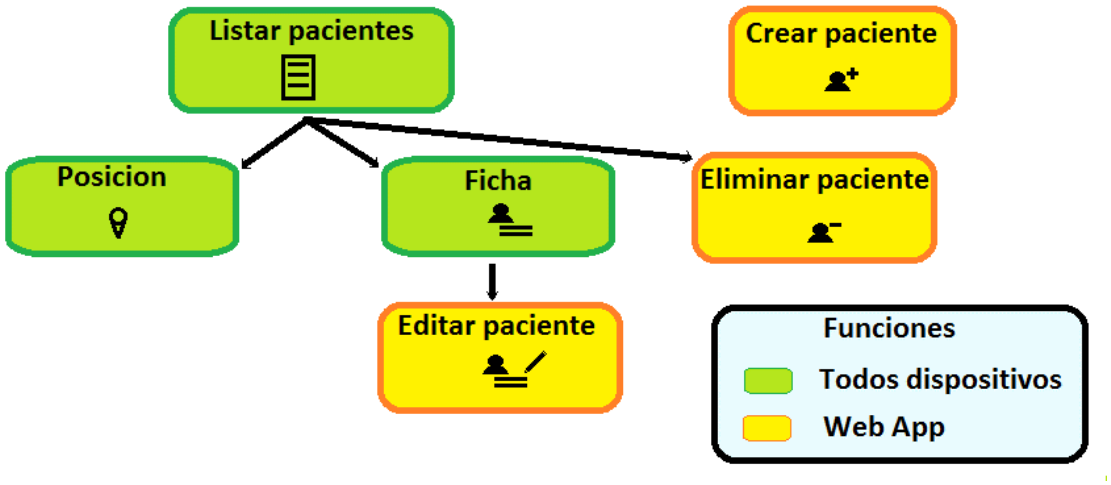

Ilustración 2. Jerarquía de funciones.

3.3. Características de los usuarios

El sistema mantiene dos roles de usuarios bien diferenciados, pacientes y cuidadores:

Cuidadores: Corresponde a un sector de personas que cuiden o sean responsables de personas con cierto grado de dependencia, en muchos casos son familiares de estos. No es necesario una alta formación académica para el uso de las herramientas del sistema, pero deben de estar familiarizados con el uso de aplicaciones móviles y un nivel de informática básico.

Pacientes: Participan en el sistema por parte de sus cuidadores, el perfil esperado son personas con movilidad, pero con cierto nivel dependencia y posible pérdida de orientación. Estos pacientes no interactúan directamente con el sistema, por lo que no es necesario ninguna restricción de formación ni experiencia.

3.4. Restricciones

Las restricciones presentes en el proyecto vienen dadas principalmente por la tecnología empleada (Nodejs, Meteor, Mongodb), la cual es a elección del desarrollador, la red de comunicaciones y la mensajería de Firebase.

Restricciones de la red Sigfox: Esta red de comunicaciones permite un envío de información máximo de 24 caracteres, junto con otras restricciones, disponibles desde https://www.sigfox.com/en/sigfox-global-iot-network. También se destaca de que Sigfox todavía no dispone de cobertura en ciertas zonas rulares.

Restricciones Firebase: Esta nube de mensajería ofrece un tamaño de mensajes de hasta 2KB, el resto de propiedades se encuentran publicadas en https://firebase.google.com/docs/cloud-messaging/concept-options.

3.5. Supuestos y dependencias

"Kune – Calzado conectado" emplea servicios y hardware de terceros, un ejemplo es la red Sigfox, en el caso de que esta cambiase los requisitos, sería necesaria una revisión de una pequeña parte de la aplicación.

El calzado localizador dispone de hardware de Telecom Design, si este cambiase su hardware es posible que alterase el formato de los mensajes enviados al servidor, teniendo que revisar este último.

La aplicación móvil ha sido desarrollada para el nivel de API 25 de Android, si las futuras versiones presentan incompatibilidades, será necesaria una revisión de la aplicación.

La Web App no mantiene muchas dependencias, es publicada por un servidor web bajo el protocolo HTTP y HTML, con un importante componente en JavaScript, no se espera que estos componentes caigan en desuso o sufran cambios importantes que alteren la aplicación, en cuanto a la tecnología empleada, que es Meteor, es relativamente reciente y cabe la posibilidad de que deje de dársele soporte.

3.6. Requisitos futuros

La aplicación es muy básica, por lo que ha sido diseñada para una reciente revisión y ampliación de esta, basándose en la experiencia de los usuarios.

En cuando la aplicación sea un poco más madura, se le incorporarán características como la capacidad de incluir una imagen del paciente, la cual será mostrada en la lista de pacientes del usuario en lugar de su nombre y será capaz de editarse desde la Web App. La aplicación móvil también hará uso de esta imagen.

Cabe la posibilidad de aumentar los detalles de los pacientes, incluyendo más datos en caso de que estos sean necesarios.

Finalmente, la cobertura de la red Sigfox y de GPS se ve reducida en el caso de emitirse desde el interior de un edificio, llegando a perder la cobertura. Para solucionar esta pérdida de cobertura, se estudiará un posible balizamiento por bluetooth[6].

[6] https://en.wikipedia.org/wiki/Bluetooth_low_energy_beacon.

4. Requisitos específicos

En este aparatado se exponen los requerimientos de la interfaz de usuario, la funcionalidad del sistema necesaria para el correcto funcionamiento del sistema.

4.1. Interfaces externas

Son dos los elementos de la aplicación que requieren de interfaces, la aplicación móvil y la Web App.

En la **Web App**, existe un formato compartido para todas las pantallas, donde el cuerpo de la aplicación mantiene amplios márgenes laterales de igual medida, en el caso de presentarse en pantallas de un ancho reducido, este margen se auto dimensiona hasta desaparecer si la pantalla es menor a 600 px, en toda la aplicación se muestra el mismo tema con los mismos colores.

A este diseño general se le suma una cabecera, esta presenta un diseño ancho que abarca todo el cuerpo de la pantalla y de poca altura. En la que se contiene en nombre de la aplicación y un botón para entrar en la pantalla de autenticación y registro, si el usuario ya se encuentra autenticado, se muta el nombre del usuario y un botón para desautenticarse.

El primer elemento en mostrarse al usuario es la pantalla de bienvenida, la cual es una presentación de la aplicación a cualquier usuario visitante, esta página no tiene ninguna interfaz definitiva hasta su puesta en producción.

La pantalla de autenticación y registro consta de dos secciones separadas horizontalmente, en el apartado superior se encuentra un formulario de dos campos y un botón para insertar credenciales, en el apartado inferior se muestra un formulario de 6 campos y un botón. En ambos formularios los campos y los botones se encuentran apilados uno debajo de otro con el nombre del campo mostrado en el lateral izquierdo.

La lista de pacientes se divide en dos apartados, un formulario de creación de nuevo paciente y a continuación, una lista con los pacientes insertados, los elementos de la lista ocupan todo el ancho del cuerpo dela pantalla mostrando de izquierda a derecha, el nombre del paciente, un botón para abrir su ficha, un botón para seguirlo con el mapa y otro botón para eliminarlo, la lista muestra una cabecera indicando el nombre de los campos. El formulario de nuevo paciente presenta el mismo formato que el de registro con dos campos.

La pantalla de ficha de paciente contiene un botón en la parte superior izquierda para volver a la lista, y elementos clave-valor mostrando los datos del paciente en el centro de la pantalla, estos elementos son mostrados verticalmente igual que los formularios.

La pantalla del mapa del paciente es mantiene un botón de volver a la lista con el mismo formato que en la ficha de paciente, debajo de este aparece un mapa interactivo ocupando todo el ancho del cuerpo con un marcador mostrando la ubicación del paciente.

En la **Aplicación móvil**, se ilustra un formato sencillo vertical y centrado, se muestra un botón en la parte superior en todas las pantallas salvo en el menú y la pantalla de entrada para volver a la pantalla anterior. Si el usuario no está autenticado, únicamente podrá acceder a la pantalla de entrada, que muestra dos campos con el nombre del campo mostrado en un placeholder y un botón que autenticará al usuario con los valores de los campos.

El menú de la aplicación presenta tres botones, para desautenticar al usuario, para acceder a la lista de pacientes y para ir a los juegos.

La pantalla de lista de pacientes tiene una lista de pacientes con el nombre del paciente y dos botones, para acceder a la ficha y al mapa.

En la pantalla de la ficha se listan verticalmente los datos del paciente con el título del campo en la parte izquierda. La pantalla del mapa muestra un mapa interactivo ocupando toda la pantalla con la posición del paciente, la forma de volver atrás es mediante el botón hardware del dispositivo móvil.

El menú de juegos mantiene cuatro botones apilados con el nombre de cada minijuego con el que enlaza, cada juego mantiene un diseño adaptado.

Por último, es necesario que los servidores de la API REST y de la Web App mantengan IPs estáticas, debido a la necesidad de comunicación entre ellos, de la Aplicación Móvil, y de la red Sigfox. Puesto que la conexión de red también se toma como una interfaz de comunicación.

4.2. Requisitos funcionales

El sistema únicamente ofrece funciones a los cuidadores si se encuentran registrados, no se definen más roles con acciones, las funciones se pueden clasificar según la plataforma desde la que se ejecuten.

Funciones realizables por cualquier plataforma.

Función	**Listar pacientes.**
Propósito	Mostrar una lista en la interfaz con todos los pacientes a los cuales el cuidador tiene autorización.
Entrada	Credenciales del cuidador (Usuario y contraseña).
Proceso	Se realiza una consulta sobre la base de datos.
Respuesta	Se muestra una lista de los pacientes con opciones para administrarlos.

Función	**Ficha de paciente.**
Propósito	Muestra al usuario los detalles de un paciente.
Entrada	Pulsar el botón de ficha del paciente.
Proceso	Se realiza una consulta sobre la base de datos con el dni del paciente.
Respuesta	Se listan en pantalla todos los datos del paciente.

Función	**Localizar paciente.**
Propósito	Conocer la ubicación actual del paciente.
Entrada	Pulsar el botón del mapa del paciente.
Proceso	Se consulta en la base de datos que "Calzado localizador" está asignado al paciente y la última ubicación de este.
Respuesta	Un mapa con una marca en la última ubicación del paciente.

Funciones realizables desde la Web App

Función	**Editar paciente**
Propósito	Cambiar los datos mostrados en la ficha del paciente.
Entrada	Todos los campos mostrados en la ficha, el botón de cambiar.
Proceso	Se genera una consulta que se ejecuta en la base de datos y sustituye lo datos del paciente, que es buscado por su dni.
Respuesta	Se actualizan los datos insertados.

Función	**Crear paciente**
Propósito	Insertar un nuevo paciente en la lista.
Entrada	Nombre y dni del paciente, botón de crear paciente.
Proceso	Se inserta un nuevo documento en la base de datos con todos los datos en su valor por defecto.
Respuesta	Aparece el nuevo paciente en la lista de pacientes y aparece en la base de datos.

Función	**Eliminar paciente**
Propósito	Eliminar uno de los pacientes de la lista.
Entrada	Pulsar el botón de eliminar del paciente.
Proceso	Se ejecuta una operación de eliminar sobre la base de datos con el dni del paciente.
Respuesta	Deja de listare el paciente y desaparece de la base de datos.

La aplicación debe de tener cuatro juegos simples, con ellos se pretende que tanto cuidador como paciente comparta una actividad. Estos juegos cumplen todos el mismo formato: Una pregunta y cuatro respuestas. De seleccionarse la respuesta correcta se felicita al usuario y se ofrece la opción de cambiar de juego o continuar el actual, de ser incorrecta el botón se vuelve de color rojo permitiendo seleccionar otra respuesta.

Primer juego, Banderas: Este juego presenta una imagen de una bandera nacional, debajo de esta aparecen cuatro botones con nombres de países, siendo estos las respuestas de las cuales una es la correcta.

Segundo juego, Números: Juego matemático, se propone al usuario una operación de suma o resta con números muy sencillos. La respuesta para seleccionar es el resultado de la operación.

Tercer juego, Objetos: Se formula una pregunta al usuario del estilo "Que hay en la cocina", entre las respuestas aparecen objetos, siendo uno de estos un objeto habitual en el área indicada, como "sartén".

Cuarto juego, Antónimos: Se muestra una palabra al usuario, en las respuestas se muestran otras cuatro palabras, siendo una de ellas un antónimo de la primera palabra, la respuesta a seleccionar es el antónimo.

4.3. Requisitos de rendimiento

La aplicación debe de soportar miles de usuarios y pacientes activos, el sistema con más carga es la API REST, puesto que recibe las posiciones del Calzado localizador y las peticiones de la Aplicación móvil, adicionalmente el servidor de la Web App realiza consultas sobre la misma maquina a la base de datos.

Por lo tanto, el servidor de la API REST debe de contar con espacio de disco para almacenar miles de usuarios, pacientes y sus posiciones. Los requisitos de memoria RAM son altos, aunque Nodejs requiere poco de este recurso; Mongodb necesita grandes cantidades de memoria principal, de hecho, en un post de la página[7] de Mongodb se recomienda usar sistemas de 64 bits.

El servidor de la Web App no requiere de muchos recursos pues Meteor es basado en Nodejs, pero si es imprescindible que el cliente disponga de un navegador capaz de ejecutar JavaScript.

4.4. Atributos del sistema

Es imprescindible que el sistema disponga de las siguientes características:

Fiabilidad: Los cuidadores confían en el funcionamiento de la aplicación para encontrar a los pacientes que se encuentren extraviados, por ello, se han invertido esfuerzos en garantizar que todas las posiciones mostradas son correctas y no dan valores nulos o de codificación errónea.

Mantenibilidad: El sistema se encuentra en una fase inicial y se someterá a futuros cambios, por ello, se ha estructurado y comentado el código para que el futuro desarrollador sea capaz de realizar cambios eficazmente.

Portabilidad: Las capas de la API REST y del servidor de la Web App no han sido diseñadas para que funcionen en distintos sistemas, aunque pueden instalarse en cualquier sistema Linux.

Sin embargo, la aplicación móvil puede ser instalada en cualquier teléfono Android con un nivel de API superior 14. Y el cliente de la Web App ejecutado en cualquier navegador.

[7] https://www.mongodb.com/blog/post/mongo-db-memory-usage

Seguridad: "Kune – Calzado conectado" nuestra datos personales y ubicación de personas, estos datos sensibles deben de encontrarse protegido y evitar ser visto por terceros, es por esto por lo que se han tomado las siguientes medidas de protección:

- Permisos.

Solo los cuidadores tienen permisos para monitorizar pacientes, los pacientes no pueden entrar en el sistema puesto que no tiene credenciales. Un cuidador solo puede gestionar los pacientes que el mismo ha creado.

- Autenticación.

Para cualquier operación es necesario la autenticación del usuario, para autenticarse, el usuario dispone de una pareja de credenciales de usuario y contraseña.

- Contraseñas.

Las contraseñas usadas por el sistema nunca se operan ni almacenan en claro, desde el primer momento se realiza una operación de hash SHA 256 y se empieza a operar con ella.

- API REST.

La API no registra sesiones, para realizar operaciones de forma segura, es necesario enviar en los argumentos las credenciales del cuidador.

URL: La primera línea de entrada es la URL, el sistema tan restrictivo implementado solo acepta las direcciones válidas y no muestra la estructura ni el tipo de ficheros. El usuario ordinario no puede encontrar ningún inconveniente en el diseño debido a que este no tiene necesidad directa de comunicarse con el servidor más que empleando la aplicación móvil, la falta de algún campo o con distinto formato implica que el usuario trata de entrar por otros medios.

Argumentos: Node.js tiene el inconveniente de que el mal procesamiento de un campo a nulo puede colgar el sistema, algo critico si no existen servicios que relancen el servidor. Por ello el primer paso es limitar las entradas validando si estas están a nulo, indefinidas o vacías (en caso de ser obligatorias) en ese orden. A continuación, se ejecuta la función "trim[8]" sobre cada variable. Aun si el servidor se detiene, existen medios para relanzarlo automáticamente.

Integridad de base de datos: La base de datos es el sistema más crítico del servidor, todos los datos que se inserten deben de cumplir siempre el mismo formato (DNI en el mismo orden y en mayúsculas…), por lo tanto, inmediatamente después de comprobar que los argumentos no pueden generar errores, se han de validar y/o convertir a un formato estándar para todo el sistema. Obligando al usuario que los datos insertados sean siempre los esperados.

8 https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/String/Trim

4.5. Casos de uso

En el presente punto se dispone a detallar todas las acciones de todos los actores que interviene en el sistema, para una mejor comprensión se utilizará el lenguaje UML. Con el lenguaje UML se representarán los casos de uso y diagramas de secuencias.

En el primer diagrama representa los actores que intervienen en "Kune – Calzado conectado", en los siguientes diagramas se muestra que el principal actor es el cuidador, pues es el propietario de la mayoría de las acciones, el paciente es un elemento pasivo en la aplicación debido a que no tiene control ninguno. Por último, Wonkacenter realiza dos acciones principales, la creación de dispositivos y su entrega a los cuidadores para su reasignación a los pacientes, aunque no es mostrado, Wonkacenter posee un control total sobre el sistema en caso de requerir algún cambio.

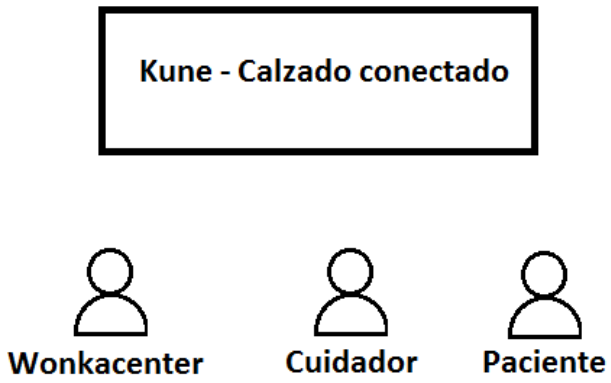

Ilustración 3. Actores.

En el diagrama del paciente, el dispositivo que tiene es el Calzado localizador, el cual es asignado por el cuidador, esté también es el que envía las posiciones GPS. Para que la ubicación sea correcta, el paciente no solo tiene que tener asignado el dispositivo en el sistema, sino que también debe de llevarlo físicamente, que es el calzado.

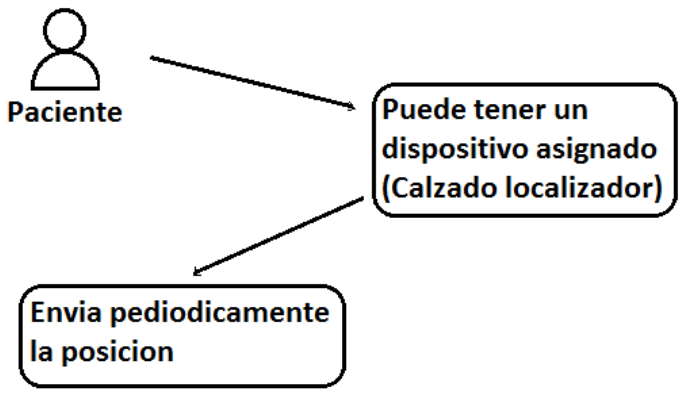

Ilustración 4. Diagrama de acciones del paciente.

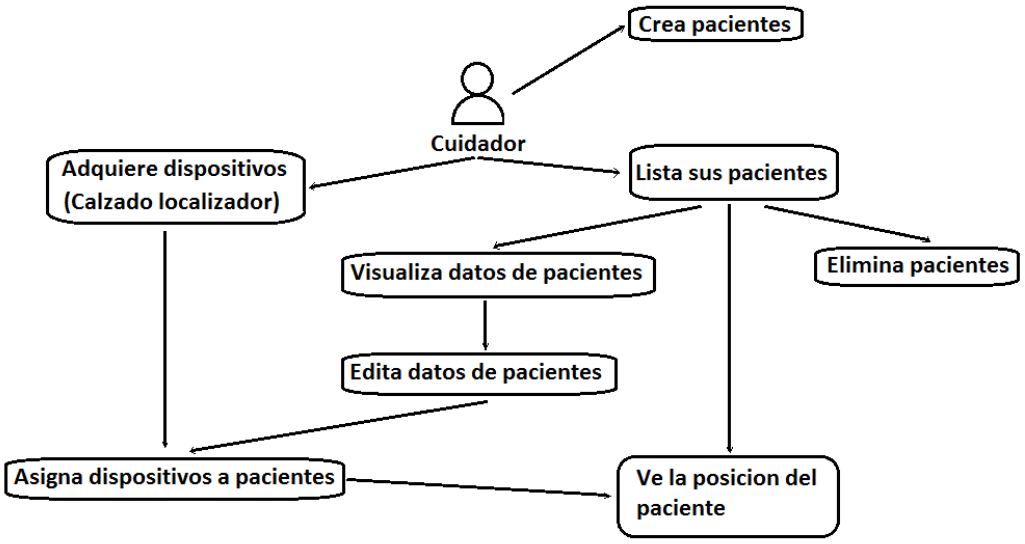

Ilustración 5. Diagrama de acciones del Cuidador.

Además del control total de la aplicación, Wonkacenter realiza principalmente las acciones de crear dispositivos, entregarlos a los cuidadores que los solicitan y permitirles a estos su asignación.

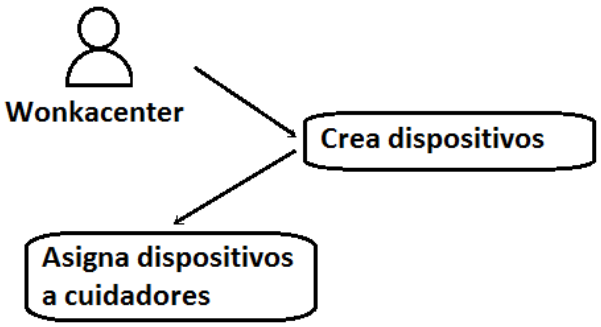

Ilustración 6. Diagrama de acciones de Wonkacenter.

5. Estudio de mercado

En el campo de la localización, especialmente de pacientes con demencia, existen otros productos que realizan funcione similares, para que el éxito de "Kune – Calzado conectado" sea posible es vital que este se diferencie del resto.

5.1. Productos similares.

A continuación, se describen productos similares:

Tweri Alzheimer Caregiver: Consiste en una aplicación móvil que tiene el paciente y el cuidador, al activase, envía la ubicación del paciente al teléfono del cuidador, esta aplicación también dispone de un botón de auxilio para el paciente.

Esta aplicación conlleva importantes inconvenientes, es necesario de que el paciente disponga de un teléfono móvil, el cual puede perder o no tener conocimientos de su uso.

Geo.band: Es un sistema de localización similar donde la emisión de la ubicación se realiza desde un dispositivo en una pulsera o llavero, el cual envía la posición a una aplicación móvil.

Este los dispositivos localizadores portados por los pacientes son fácilmente desprendibles, especialmente las pulseras, que pueden ser sentidas como elementos ajenos.

Keruve: Esta aplicación consiste en un reloj que es portado por el paciente y es emisor de su posición, el receptor de la señal es un dispositivo con apariencia de Tablet. Esta solución implica que el cuidador lleve consigo el dispositivo receptor, de olvidarse, o podrá localizar al paciente.

iTraq: Es un simpe sistema de geolocalización mediante unos dispositivos en forma de tarjetas, la cuales no utilizan el servicio de GPS, sino que triangula la posición a través de torres de telefonía. Finalmente, el cuidador puede ver la localización a través de su teléfono móvil.

Simplemente consiste en un sistema de rastreo de propósito general extrapolado al cuidado de pacientes, sin emitir alertas de alejamiento. Las tarjetas de localización pueden perderse.

Nock Senior: Sistema de localización a través de un reloj, el cual requiere de una tarjeta SIM ofrecida por la compañía, la ubicación se muestra en una aplicación en el teléfono del cuidador.

También se dan otras soluciones en fase experimental, como un brazalete localizador de la CEAFA y otro dispositivo del centro de terapéutico de día de AFA Bierzo.

5.2. Valor diferenciador

“Kune – Calzado conectado” se diferencia de los otros productos en que el dispositivo que registra la posición se encuentra en el calzado, un elemento del cual el paciente no se desprende con facilidad, de modo que siempre lo llevara consigo.

El cuidador puede observar la posición del paciente mediante dos vías: la Aplicación móvil y la Web App. Con este modo, el cuidador puede conocer la ubicación del paciente aun sin disponer de un teléfono móvil.

La aplicación es capaz de enviar alertas al teléfono del cuidador indicando eventos como que un paciente ha salido de casa o que ha permanecido demasiado tiempo en un solo lugar.

Todas estas características contribuyen a destacar el producto de Wonkacenter.

6. Diseño técnico

La aplicación se encuentra dividida en 4 capas, cada capa cumple con una función específica: Calzado localizador, API REST, Web App y Aplicación móvil.

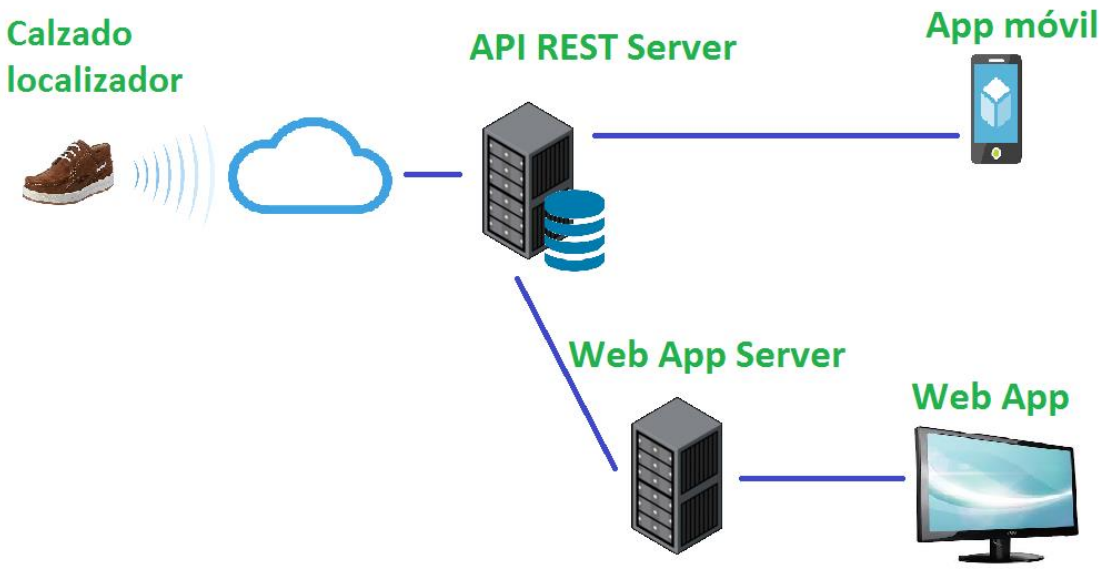

Ilustración 7. Componentes de la aplicación.

6.1. Componentes de la aplicación

6.1.1. Calzado localizador

El calzado localizador son dispositivos hardware que lo pacientes llevan consigo, estos dispositivos consisten en placas de Telecom Design [9] programadas para envían periódicamente mensajes al servidor con las coordenadas actuales a través de la red de telecomunicaciones Sigfox[10].

Normalmente, este tipo de dispositivos vienen forma de collares y pulseras, pero los pacientes tienden a deshacerse de estos elementos por no reconocerlos como propios. Para solventar este problema, el dispositivo es incorporado en el calzado, el cual es rara vez retirado.

Las placas se han programado en un departamento aparte de la empresa.

[9] http://www.telecom-design.com/
[10] https://www.sigfox.com/

6.1.2. API REST

Es el componente principal de la aplicación, el cual recibe los datos de la nube de comunicaciones Sigfox y sirve información a la aplicación móvil. Cuando se recibe un mensaje de posición de un dispositivo localizador, se almacena directamente en la base de datos, se comprueba si ha entrado o salido de la zona designada como casa y de darse uno de estos casos, se notifica a la aplicación móvil.

La aplicación móvil también obtiene datos a través de la API, autenticando todas las consultas mediante usuario y contraseña, solo se servirán los datos a los que el usuario este autorizado. Un ejemplo de las funciones de la API REST son:

- Consulta de usuario y contraseña valido.
- Listado de pacientes del usuario
- Posición del paciente.

Dirección	Campos	Función	Usado para
insert_token	Usuario, contraseña, token.	Inserta un token de FCM en el usuario indicado.	Aplicación móvil.
receive_position	Identificador de dispositivo, mensaje.	Registra una nueva posición indicada en el mensaje al dispositivo indicado. Puede enviar una notificación a la aplicación móvil del cuidador si se reúnen las condiciones.	Recepción de red Sigfox.
send_fcm		Envía un mensaje de prueba de FCM.	Implementación, usado para realizar pruebas.
user_new	Nombre, DNI, email, usuario, contraseña.	Crea un nuevo usuario.	Implementación, usado para realizar pruebas.
user_change_pass	DNI usuario, antigua contraseña, nueva contraseña,	Cambia la contraseña del usuario indicado.	Implementación, usado para realizar pruebas.
user_login	Usuario, contraseña, token,	Comprueba las credenciales	Aplicación móvil.

		indicadas y almacena un token FCM.	
patient_list_by	Usuario, contraseña.	Lista los pacientes de un usuario.	Aplicación móvil.
patient_new	Nombre, DNI.	Crea un nuevo paciente.	Implementación, usado para realizar pruebas.
link_patient_device	DNI paciente, identificador dispositivo.	Vincula un dispositivo con un paciente.	Implementación, usado para realizar pruebas.
link_patient_user	DNI paciente, DNI usuario.	Vincula un paciente a un usuario.	Implementación, usado para realizar pruebas.
dev_new	Identificador, usuario.	Crea un nuevo dispositivo.	Implementación, usado para realizar pruebas.
position_new	Dispositivo, latitud, longitud.	Crea una nueva posición.	Implementación, usado para realizar pruebas.
position_last_by	DNI de paciente, usuario, contraseña.	Muestra la última posición de un paciente.	Aplicación móvil.

Estructura del servidor de API REST

El servidor ha sido construido en Node.js, y consta de 3 archivos principales:

- **Index.js**. Es el archivo que indica que peticiones se atienden y llama a server.js para arrancar el servicio.
  Las peticiones permitidas se almacenan en un diccionario, donde la clave es una cadena con la petición y el valor es una llamada a la función que tratará la petición.

  Si en el diccionario existe “diccionario["/listar"] = función_listar;”
  la URL “localhost/listar” será atendida por la función “funcion_listar();”.

  Esta estructura tan restrictiva oculta los nombres de los ficheros y sus funciones, proporcionando una mayor seguridad, adicionalmente protege de intentos de ataque por directorio transversal como: “../etc/passwd”.

- **Server.js**. Contiene el servidor que escucha en puerto indicado, gestiona las conexiones seguras y envía las peticiones recibidas a router.js.
  Los certificados para las conexiones seguras se almacenan en el directorio "certificates", que son leídos a cada conexión, permitiendo cambiar el certificado sin necesidad de reiniciar el servicio.

- **Router.js.** Es llamado por server.js cada vez que recibe una petición, este la recoge y en el caso de la petición se encuentre en index.js se llama al manejador de petición, de lo contrario de devuelve el error 404.

Construcción de los manejadores de petición.

Los manejadores de petición son los encargados de realizar las peticiones de los clientes y de devolver resultados, para ello, se realizan tres pasos importantes:

- **Recepción de argumentos.** Los argumentos son recibos por los manejadores a través de la URL. Al principio, para facilitar el desarrollo, se recibían los argumentos a través de una URL no semántica, recogiendo los datos mediante su clave, finalmente se recogen con una URL semántica para dar un mejor acabado a la API. Para recoger las variables no semánticamente se ha usado la librería "querystring".
- **Validación**. Los datos insertados han de cumplir un formato para mantener la integridad de la base de datos y garantizar que el servidor no se detenga por valores nulos, por ello, las validaciones intentan ser muy estrictas y/o transformar los datos insertados. Para facilitar este proceso se ha construido la librería valídate, que contiene funciones para validar strings, números, DNI, valores nulos…
- **Ejecución.** Generalmente, la ejecución es una instrucción a la base de datos, para protegerla mejor, se realizan librerías de acceso a la BBDD y solo mediante ellas se opera.

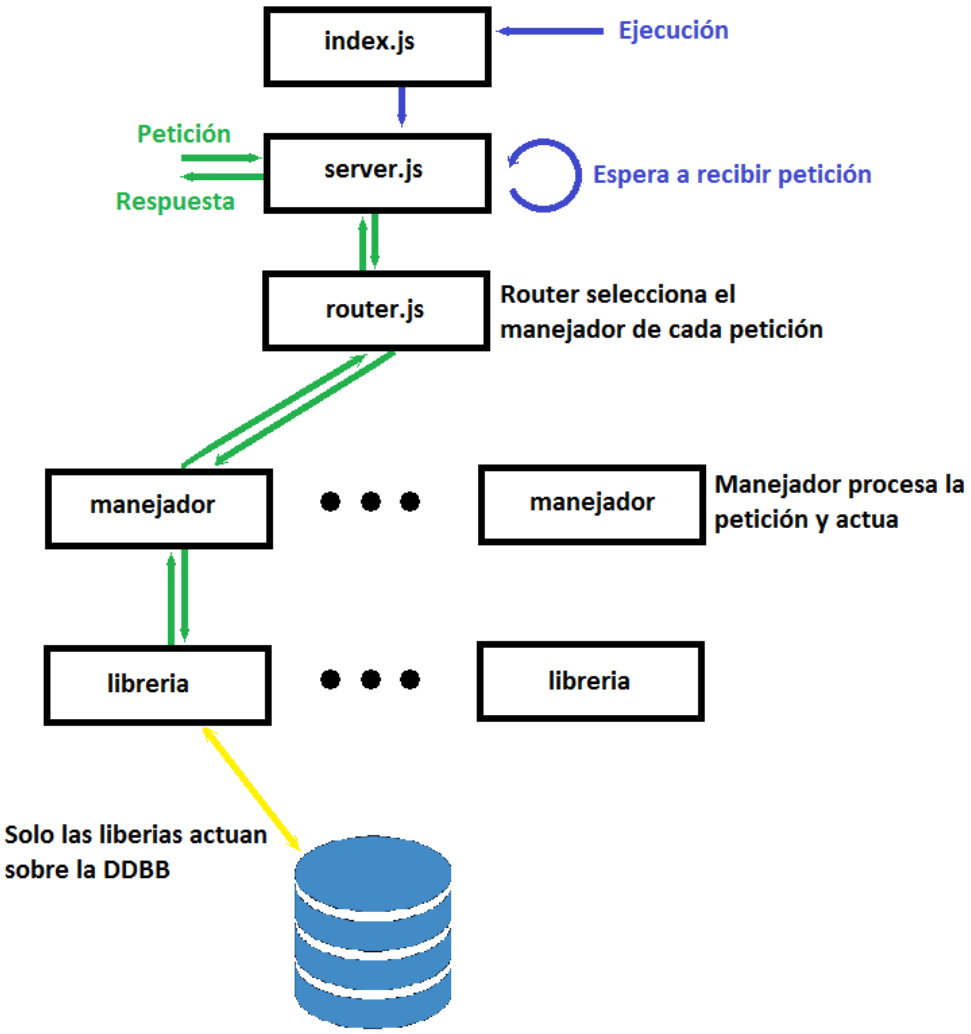

Ilustración 8. Estructura de la API REST

Formato de respuesta de la API REST.

Las respuestas enviadas por la API REST conservan siempre el mismo formato, consiste en un diccionario de dos campos, "error" y "data".

- Error: Campo que indica si la petición ha sido satisfecha con éxito o si existe algún problema, es similar al Status Code del protocolo HTTP, normalmente, si error es distinto de 1 el campo de data está vacío.

ERROR	Significado
0	Error de campos
1	Todo OK
2	Error DDBB

Tabla 1. Campo "error" de respuesta de la API REST.

- Data: Campo sobre el cual la API inserta los datos a recibir por el cliente en forma de array o diccionario.

Ilustración 9. Ejemplo de respuesta de la API REST.

Notas sobre seguridad:

- **URL:** La primera línea de entrada del usuario es la URL, el sistema tan restrictivo implementado que solo acepta las direcciones válidas y no muestra la estructura ni el tipo de ficheros. El usuario ordinario no puede encontrar ningún inconveniente en el diseño debido a que este no tiene necesidad directa de comunicarse con el servidor más que empleando apps.
- **Argumentos:** Node.js tiene el inconveniente de que el mal procesamiento de un campo a nulo puede colgar el sistema, algo critico si no existen servicios que relancen el servidor. Por ello el primer paso es limitar las entradas validado si estas están a nulo, indefinidas o vacías (en caso de ser obligatorias) en ese orden. A continuación, se ejecuta la función "trim" sobre cada variable.

- **Integridad de base de datos:** La base de datos es el sistema más crítico del servidor, todos los datos que se inserten deben de cumplir siempre el mismo formato (Contraseñas hasheadas, DNI en el mismo orden y en mayúsculas…), por lo tanto, inmediatamente después de comprobar que los argumentos no pueden generar errores, se han de validar y/o convertir a un formato estándar. Obligando al usuario que los datos insertados sean siempre los esperados.

Arranque del servidor.

Node.js se ejecuta con un comando como si de un simple script se tratase, pero el arranque del servidor a mano cada vez que la maquina se inicia no es una opción viable, un reinicio de la maquina por un corte de luz implicaría no haber servicio hasta que se ejecute el servicio a mano. Por lo que ha decidido utilizar un gestor de procesos de Node.js llamado **pm2**, el cual convierte las ejecuciones indicadas de Node.js en demonios del sistema y monitorizar su actividad.

Este gestor de procesos es especialmente útil, debido a la alta probabilidad de que el servicio de detenga por un error de programación, ya que como nuestro servidor consiste en un único hilo en ejecución, un simple valor nulo insertado por un usuario y no validado puede detener la ejecución. Pm2 es capaz de relanzar los procesos de Node.js caídos.

Para instalar pm2 utilizamos el gestor de paquetes NPM.

```
npm install -g pm2
```

pm2 fue desarrollado para versiones antiguas de Node.js de forma que trata de llamar a Node.js utilizando el comando “node” en lugar de “nodejs”. La mayoría de soluciones propuestas por la comunidad tratan de que instalemos “node-legacy” con “apt-get install node-legacy”, acción errónea en mi opinión ya que se ejecuta versiones obsoletas de Node.js. Una solución mejor es crear un enlace simbólico con el siguiente comando.

```
ln -s /usr/bin/nodejs /usr/bin/node
```

A continuación, se puede arrancar el servidor y mantener el servicio con:

```
pm2 start index.js --name 'kune'
pm2 save
```

Es importante que se arranque desde el propio directorio donde se encuentra el fichero “index.js”, de lo contrario, las rutas relativas que contenga no serán encontradas.

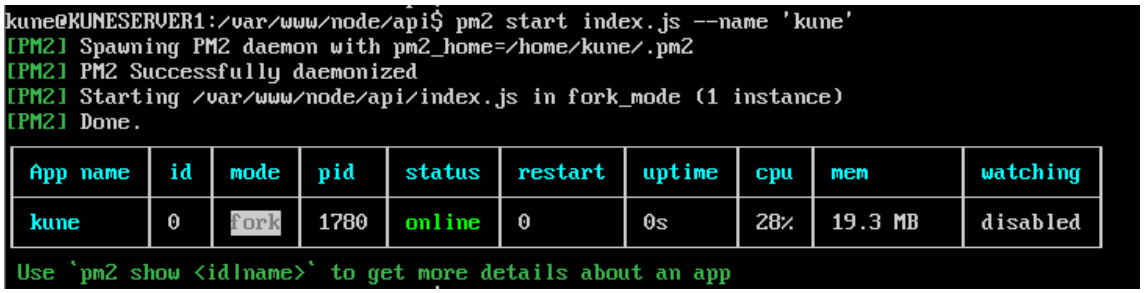

Ilustración 10. Arranque del servidor de la API REST.

Envío de notificaciones a la aplicación móvil.

La API REST se encarga de las comunicaciones con la aplicación móvil, esto incluye las notificaciones, estas están muy bien documentadas para implementarse en PHP por Google, pero la documentación para ser enviada a través de JavaScript y de más concretamente por Node.js es más escasa, por ello se ha valido de la librería fcm-node[11], que facilita el envío de mensajes a través de Firebase, utilizado para enviar alertas a los dispositivos móviles.

Esta librería genera el siguiente error al ser ejecutada:

```
SyntaxError: Block-scoped declarations (let, const, function, class) not yet
supported outside strict mode
```

[11] https://www.npmjs.com/package/fcm-node

Para corregirlo[12], en el fichero “fcm-node/lib/topic_request.js” ha sido necesario incluir en la línea 6 la instrucción:

```
use strict;
```

Conexión con Mongodb

Para la inserción de datos en Mongodb se ha utilizado la librería mongodb[13], instalada a través de NPM.

6.1.3. Web App

Consiste en una aplicación accesible desde el navegador a través de la cual el usuario consulta la lista de pacientes, gestionarlos y visualizar sus posiciones. Para acceder a la gestión, el usuario debe de disponer de las credenciales. Esta Web App accede directamente a la base de datos sin necesidad de acceder a la API REST, con motivo de evitar que el bloqueo de un sistema impida el acceso a los datos.

Consiste en una página simple con cuatro apartados:

- Login, a través del cual el usuario se autentica.
- Listado, de los usuarios autorizados.
- Ficha de paciente, con los datos almacenados de él.
- Mapa, con la localización del paciente.

También permite cambiar datos de los pacientes.

Para el portal web de Kune, se ha escogido utilizar Meteor, que es un entorno de desarrollo en JavaScript que permite construir páginas web sin diferenciar cliente de servidor, es decir, el programador construye la página como si de un programa local se tratara y Meteor se encarga de separar código de cliente y de servidor preprocesador el código construido. Adicionalmente, el entorno garantiza una actualización automática de los datos insertados en el servidor en los demás clientes gracias a que estos se subscriben automáticamente este. Ofreciendo con todo esto un portal web de calidad con muy poco esfuerzo.

[12] https://github.com/jlcvp/fcm-node/issues/26
[13] https://www.npmjs.com/package/mongodb

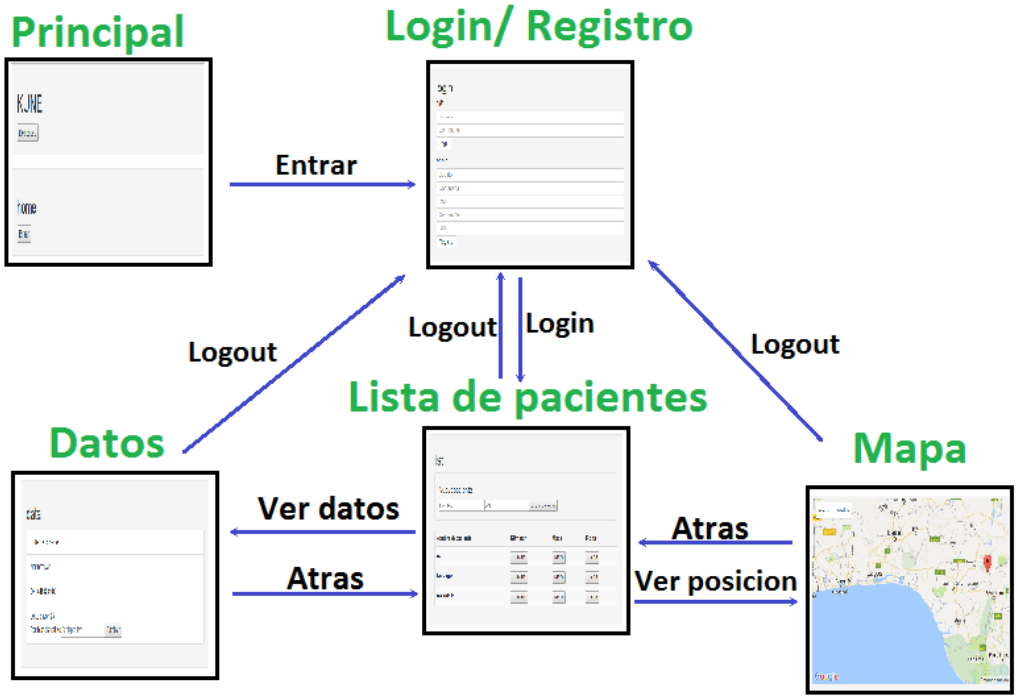

Ilustración 11. Esquema de la Web App.

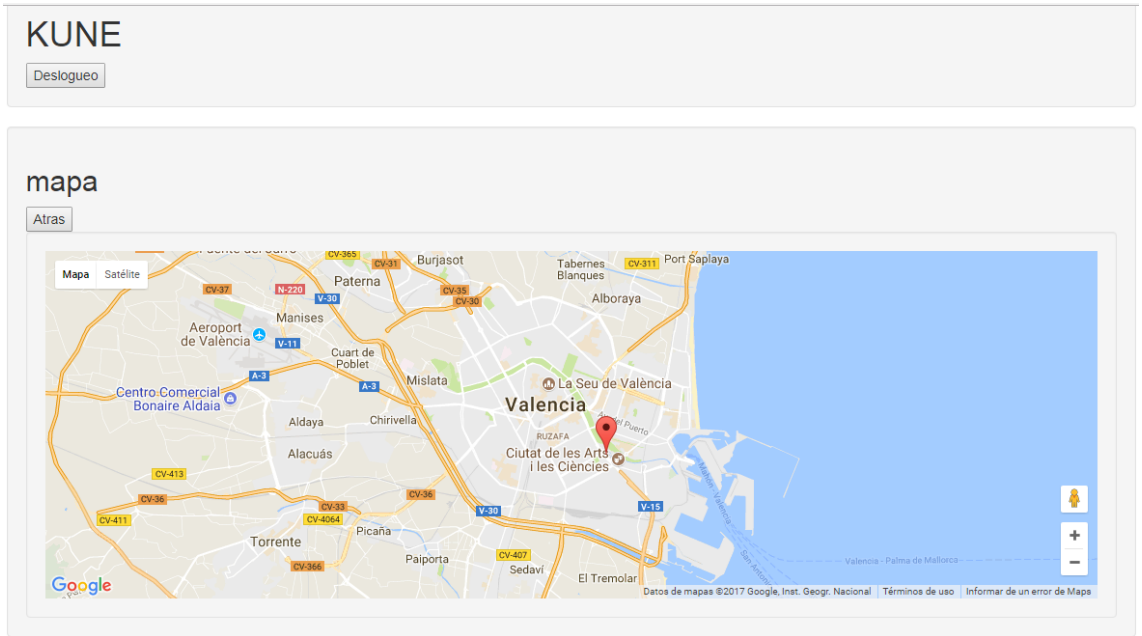

Ilustración 12. Mapa motrado en la Web App.

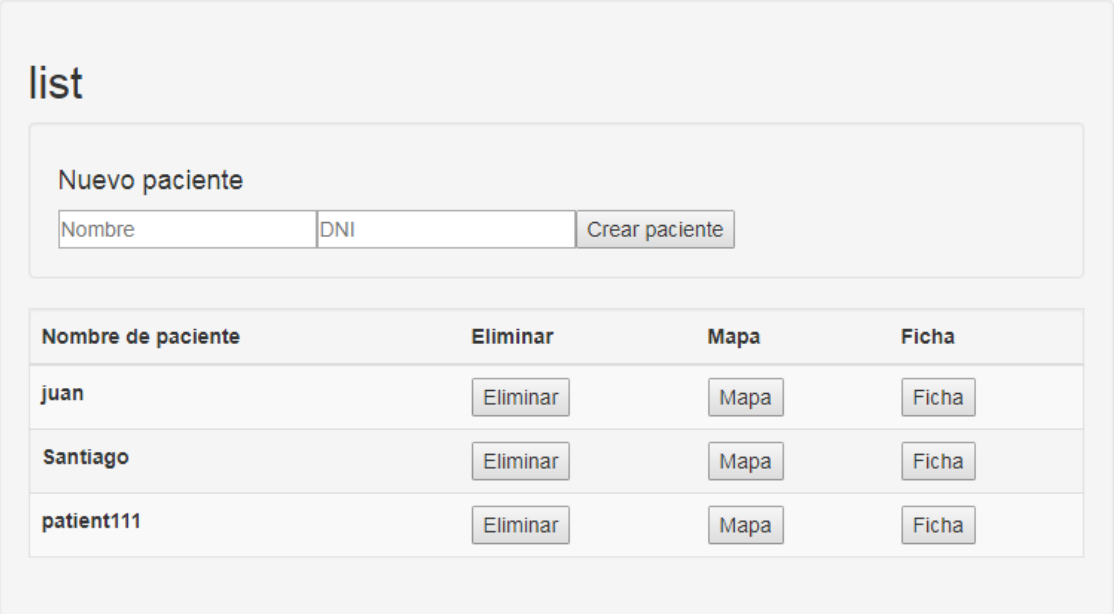

Ilustración 13. Lista de pacientes de la Web App.

6.1.4. Aplicación móvil

El usuario puede instalarse la aplicación móvil para poder consultar la localización de los pacientes y recibir las alertas que el servidor envía, la aplicación no tiene capacidad de inserción de datos, solo de consulta.

Adicionalmente dispone de cuatro juegos simples con el objeto de entretenerse con los pacientes.

Para el funcionamiento de la aplicación es necesaria la conexión a internet y que la API REST se encuentre funcionando.

El diseño gráfico es simple a espera de un diseño final en cuanto las funcionalidades de la aplicación haya sido acabadas, La mayoría de la experiencia del usuario viene dada por el aspecto de la interfaz, y generalmente, no se reciben instrucciones del funcionamiento, por lo que el usuario aprenderá a utilizarla por su aspecto y prueba y error. Es fundamental que cada elemento sea intuitivo.

Se ha usado como color predominante el azul claro complementado con el verde, escogidos por ser colores que transmiten confianza con la aplicación.

La mayoría de elementos presentan bordes redondeados y bien definidos para poder diferenciarse con facilidad. Los botones y textos presentan todos los mismos formatos para que estos sean intuitivos.

La interfaz de la aplicación comunica al usuario el estado de sus mediante mensajes emergentes (ejemplo: No hay conexión, o registro incorrecto).

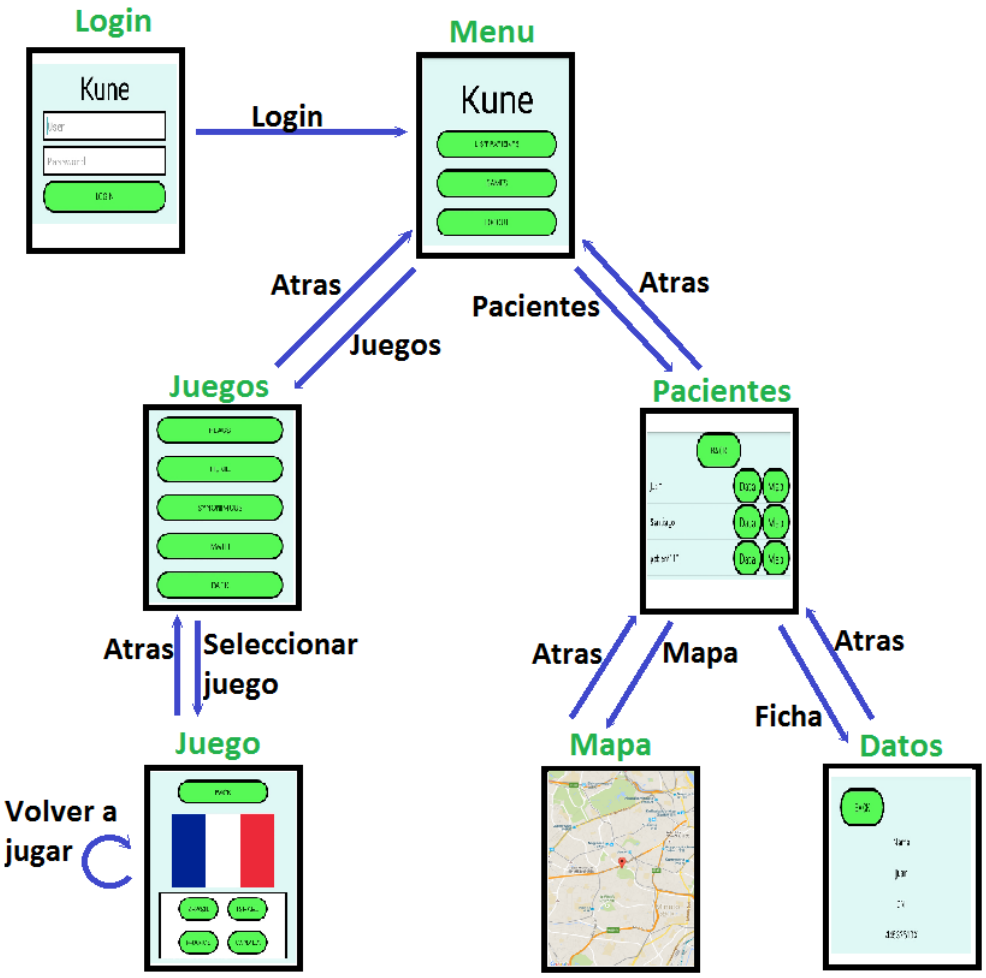

Ilustración 14. Esquema de la aplicación móvil.

La aplicación arranca siempre en una pantalla inicial con únicamente el título de la aplicación, con motivo de comprobar si existe un usuario registrado, debido a la rapidez de la comprobación, esta pantalla no es apenas visible. Se salta automáticamente a la pantalla de Login o de menú dependiendo de si existe un usuario.

La pantalla de Login es un formulario sencillo de dos campos y un botón, destinados a insertar las credenciales del usuario y comprobarlas a través el servidor.

La pantalla de menú muestra tres botones con los que ir a los juegos, ver el listado de pacientes o desloguearse.

El listado de pacientes muestra una lisa de los pacientes a los cuales el usuario tiene permisos de listar, junto con botone para ver el reto de la información de cada uno e ir al mapa con su ubicación.

Las herramientas para la construcción de la aplicación son principalmente Android Studio, junto con los recursos de Google para implementar la actividad del mapa.

La ilustración 15 expone un ejemplo de la interfaz de la aplicación móvil, a la izquierda se muestra la interfaz de un pequeño minijuego diseñado para que tanto el paciente como el cuidador ejerciten la memoria, a la derecha vemos la lista de pacientes y su botones.

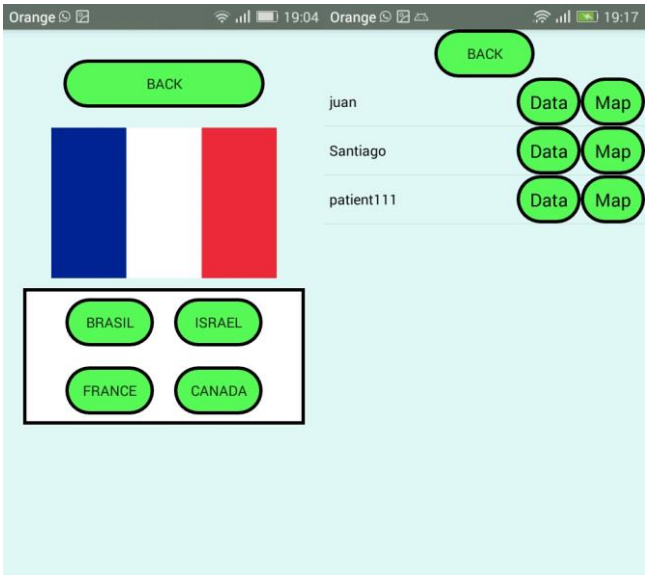

Ilustración 15. Juego y lista de la aplicación móvil.

Idiomas

La aplicación ha sido preparada para poder traducirse a varios idiomas, actualmente está disponible el inglés (por defecto) y en español, esto idiomas se seleccionan automáticamente por la configuración de idioma del teléfono. Todas las cadenas de palabras mostradas al usuario se almacenan en el fichero string.xml, del cual se extraen para ser mostradas, este fichero tiene la propiedad de ser traducido fácilmente mediante Android Studio, el cuan también ofrece servicios de traducción.

Conexiones

Todas las conexiones a internet (por ejemplo, a la API, se gestionan desde una sola clase, este sistema permite centralizar las conexiones con motivo facilitar la programación y ampliaciones futuras.

Documentación

El código de la aplicación se encuentra debidamente documentada a nivel de código para futuros cambios. Adicionalmente se redactará un documento indicando en todas sus pantallas la funcionalidad de cada elemento en estas.

Notificaciones

La aplicación es capaz de recibir notificaciones, las cuales son enviadas desde la API REST a través de los servicios de Firebase. Estas notificaciones indican cambios en el desplazamiento de los pacientes, en concreto, que han salido de casa. Al ser abiertas llevan al mapa donde se encuentra la ubicación del paciente.

6.2. Base de datos

El sistema trabaja con una base de datos Mongodb, en el anexo 9.1 se explican el análisis y conclusiones realizadas para elegir usar este tipo de base de datos.

Para facilitar la compresión, se incluye una tabla de equivalencias entre conceptos MySQL y Mongo.

Equivalencias MySQL con Mongodb.	
MySQL	Mongodb
Tabla	Colección
Fila	Documento
Columna	Campo

Tabla 2. Equivalencias MySQL con Mongodb.

Las relaciones entre las tablas (creadas por la aplicación, no por la base de datos debido a que no es relacional), se pueden observar en la siguiente ilustración:

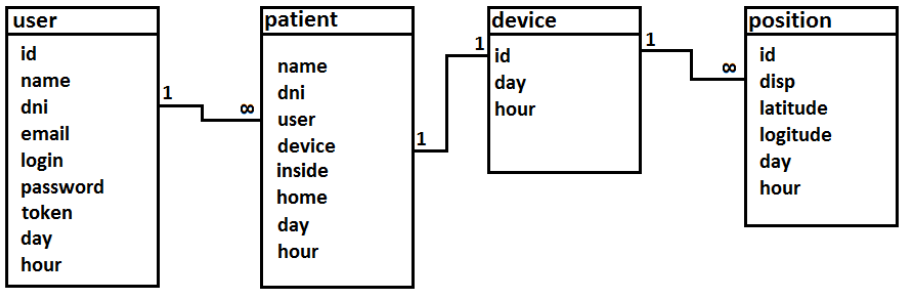

Ilustración 16. Diseño de la base de datos.

Las descripciones de cada colección vienen definidas en las siguientes tablas:

users_2	
Campo	Descripción
id_	Clave autogenerada en cada documento.
name	Nombre real del usuario.
dni	Número de DNI del usuario.
email	Dirección de correo del usuario
day	Fecha de creación del usuario.
hour	Hora de creación del usuario.
login	Seudónimo del usuario usado para entrar en el sistema.
password	Contraseña del usuario, se almacena con formato hash.

Tabla 3. Diseño de la tabla de usuarios.

Los usuarios registrados en la aplicación, lo cuales llevan el rol de cuidadores son almacenados en la colección “users_2”. La colección users_2 originalmente era llamada “users”, el motivo del cambio es que Meteor crea una colección también llamada users, con el cambio de nombre se resuelve el conflicto de colecciones. El campo empleado como búsqueda entre cuidadores es el dni.

La colección “users” creada por Meteor es empleada para sus funciones de autenticación de usuarios y es gestionada por él mismo.

patients	
Campo	Descripción
id_	Clave autogenerada en cada documento.
user	Número de DNI del usuario cuidador.
disp	Identificador del dispositivo vinculado al usuario.
name	Nombre del paciente.
dni	Número de DNI del paciente.
day	Fecha de creación del paciente.
hour	Hora de creación del paciente.

Tabla 4. Diseño de la tabla de pacientes.

Los pacientes son registrados por los cuidadores y almacenados en la tabla “patients”, esta a su vez, vincula los pacientes con sus cuidadores y sus dispositivos asignados. El campo empleado para la búsqueda de pacientes es el dni.

devices	
Campo	Descripción
id_	Clave autogenerada en cada documento.
id	Código identificador del dispositivo.
day	Fecha de creación del dispositivo.
hour	Hora de creación del dispositivo.

Tabla 5. Diseño de la tabla de dispositivos.

Los dispositivos hacen referencia al Calzado localizador, almacenados en el sistema en la colección “devices”. El campo id es empleado para la búsqueda y asignación de dispositivos.

positions	
Campo	Descripción
id_	Clave autogenerada en cada documento.
disp	Código identificador del dispositivo emisor.
latitude	Coordenada de latitud.
longitude	Coordenada de longitud.
day	Fecha de creación de las coordenadas.
hour	Hora de creación de las coordenadas.

Tabla 6. Diseño de la tabla de posiciones.

La colección “positions” almacena las posiciones enviadas por los dispositivos (Calzado localizador).

6.3. Mensajes Sigfox

La red de comunicaciones Sigfox envía los mensajes de posición desde el Calado localizador hasta la API REST. Se ha optimizado el tamaño de mensaje a 24 caracteres, en los que se envía los datos de coordenadas y una posible ampliación para la altura.

En la ilustración 17 se desglosa cara carácter enviado en un mensaje Sigfox. EL Calzado localizador envía el mensaje a la nube a través de su propia red de comunicaciones, la nube devuelve el mensaje al servidor a través de una URL.

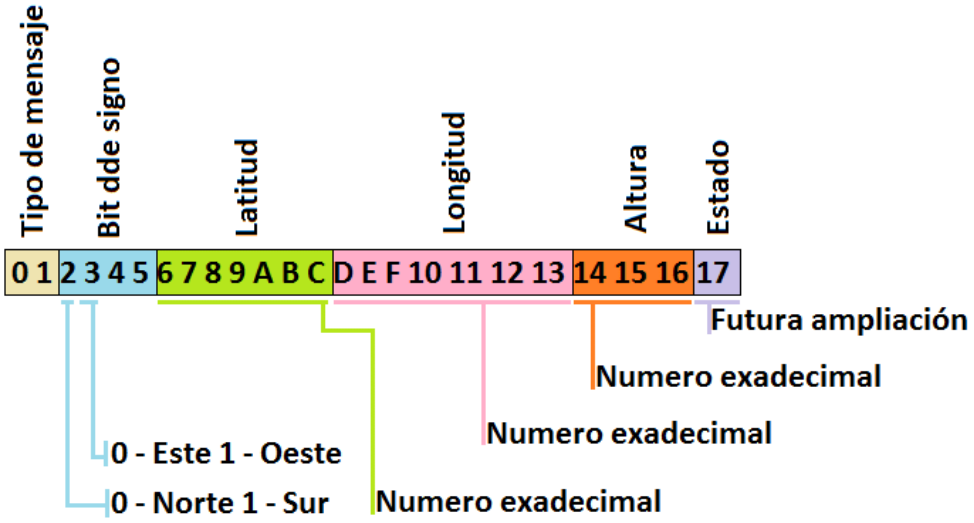

Ilustración 17. Formato de mensaje Sigfox.

Los campos de Latitud, Longitud y Altura se han compactado en hexadecimal para que ocupen un menor tamaño. Los campos de Altura y Estado no son empleados actualmente y se reservan como una posible futura ampliación.

El campo Bit de signo define el signo de la Latitud y Longitud, que supone que la coordenada apunta al Norte, Sur, Este u Oeste según la ilustración, también se reservan dos campos para futuros cambios.

6.4. Sistema de logs

Los logs son una herramienta importante para resolver futuras incidencias, por lo que es una buena decisión dotar de una buena gestión de estos y prever que datos son los necesarios.

Como la API REST ha sido construido en Node.js, se ha escogido el paquete de gestión de logs Winston, por su versatilidad, sencillez y documentación. Permite almacenar eventos en distintos ficheros y ordenarlos por niveles, además los datos son almacenados en formato JSON, permitiendo tanto ser leído fácilmente por JavaScript que por una búsqueda por comandos.

Se prevé que los logs ayuden a resolver incidencias del tipo:

- Reclamaciones de usuarios: Los usuarios pueden reclamar información que ellos mismos han alterado o eliminado y se debe de demostrar que el sistema ha obrado según lo planeado.
- Esclarecer ataques: Muchos de los ataques realizados a organizaciones comienzan con un comportamiento sospechoso con un mes de antelación antes de que estos se produzcan.
- Búsqueda de errores: Si se ha producido un fallo en el sistema, los logs pueden indicar el origen.

Sabiendo estos datos, se ha decidido almacenar los logs en 3 categorías:

- Log de servidor. Almacenado en /var/log/kune/log_server.log
  Recoge los eventos internos realizados por el servidor (Servidor lanzado, conexión iniciada, petición sin resultado…).
- Log de base de datos. Almacenado en /var/log/kune/log_ddbb.log
  Registra las acciones realizadas por lavase de datos (Inserciones, búsquedas…).
- Log de usuarios. Almacenado en /var/log/kune/log_user.log
  Registra las acciones realizadas por los usuarios y sus errores (Usuario implicado, dirección IP, petición, errores de validación…).

Los niveles de los mensajes se han clasificado por su nivel de criticidad, y ordenados en la siguiente tabla.

Nivel de mensaje de log	Descripción
FATAL	Mensaje indicando que el servidor se detenido, este tipo de mensajes no han sido implementados en la aplicación.
ERROR	Estos mensajes indican que el servidor no puede continuar su ejecución, estos mensajes deben de ser corregidos rápidamente y suelen ser escasos.
WARNING	Indica comportamientos anómalos en el servidor, los cuales no suponen un peligro para su funcionamiento, son poco comunes y son usados para indicar campos con tipo incorrecto, no existen datos y similares.
INFO	Indica eventos recibidos por el servidor como: peticiones de clientes, servidor lanzado/detenido, usuario invalido. Se acumulan muy rápidamente.
DEBUG	Este tipo de mensajes es utilizado para seguir el progreso de la aplicación y facilitar su desarrollo, serán comentados en el entorno de producción, contiene resultado de variables e indica el lugar donde se ejecuta.
TRACE	Mensajes que indican en que lugar se encuéntrala ejecución del programa, se acumulan muy rápidamente y no han ido usados para el proyecto.

Tabla 7. Nivel de los mensajes de log.

6.5. Problemas encontrados con Meteor:

Meteor, ha presentado los siguientes problemas programáticos:

- Registro de sesión:

Meteor gestiona automáticamente las sesiones, creando cuentas y validando usuarios, restando mucho trabajo de programación, pero resulta imposible llevar un control de las cuentas debido a que el programador no puede acceder al proceso de creación de estas. Cada cuenta es creada por el cliente y almacenada en la colección "users" en Mongodb, únicamente con los campos usuario, email y contraseña.

Si se dispone de una colección de usuarios con otros campos, sería recomendable realizar una tarea que sincronice la tabla de usuarios propia con la de Meteor.

- Conexión a base de datos:

Por defecto, los datos y colecciones grabados se realizan en una base de datos propia de Meteor, para poder usar una existente, hay que añadir en el fichero *"/etc/enviorement"* la siguiente línea que conectará Meteor con la base de datos ya creada:

```
export MONGO_URL='mongodb://localhost:26016/KuneDB'
```

- Consultas en base de datos.

Este es un problema doble, por un lado, los datos que han demostrarse al cliente deben de realizarse mediante un sistema de suscripción, el otro inconveniente es no poder realizarse consultas relacionales.

El sistema de suscrición consiste en que el cliente mantiene una base de datos propia que garantiza una mayor velocidad, esta base de datos se actualiza automáticamente con el servidor, el cual envía al cliente solo los datos que se le autorice, y el cliente selecciona los datos a mostrar. Al realizar un cambio de usuario, el servidor publica los nuevos datos. Este sistema fuerza a que en cada colección exista un campo indicando el usuario al que pertenece o a realizar consultas encadenadas.

En cuanto a las consultas relacionales, se resuelve mediante diversas consultas encadenadas, convirtiendo consultas sencillas en muchas líneas de código recursivo. Este problema es propio de Mongodb, no de Meteor.

- Ámbito de variables:

Generalmente, las variables globales son accesibles tanto desde el cliente como del servidor, pero cuando se ejecutan instrucciones únicamente desde el servidor (por ejemplo, recuperar un elemento de la base de datos) el valor de una variable global puede ser diferente para el servidor no siendo recuperable por el cliente.

- Navegación:

La navegación en la App es recursiva en mi primera implementación, es decir, la sección del mapa se abre dentro de la sección de la lista, ocultando el contenido de esta. El comportamiento ideal sería poder abrir las secciones en pantallas separadas, igual

que con una página tradicional, este sistema es posible implementarlo mediante la librería “iron:router” [14] [15], que es fácilmente instalable desde el comando:

```
meteor add iron:router
```

Su utilización es simple y facilita el uso de las plantillas “templates” junto con React, adicionalmente incorpora la captura de variables con URL amigables. Por el contrario, resulta más complicado la inserción de propiedades en cada nueva pantalla, este problema se puede cubrir con el uso de variables globales o mediante la librería session, explicada a continuación.

- Librería de Mapas:

Finamente, se ha intentado utilizar la api de Google para la implementación de un mapa utilizando los servicios de Google Maps, pero debido a la reactividad de Meteor, no es fácilmente implementarla y se deben de utilizar librería de terceros, como dburles:google-maps, cuya documentación está parcialmente completa y requiere de varias pruebas para dominarla.

6.5.1. Otras herramientas

La librería session es propia de Meteor que crea un diccionario global, ideal para conservar todas las variables globales, pero es manipulable desde la consola del cliente. No apto para sesiones de usuario. Naturalmente, al refrescar el navegador, el contenido de este diccionario se pierde a igual que del resto de variables.

```
meteor add session
```

6.5.2. URL semántica

El siguiente paso es que la URL que se comunique con nuestros servidores mantenga el formato de una URL semántica, lo que quiere decir que lo datos enviados a través de las variables de una URL pasan a ser enviados como si de una ruta se tratase, con el fin de aumentar la legibilidad de ésta por un humano y trasmiten más confianza y profesionalidad.

URL no semántica:

http://localhost/ usuarios?nombre=juan

URL semántica:

http:// localhost /usuarios/juan

[14] http://iron-meteor.github.io/iron-router/
[15] https://atmospherejs.com/iron/router

El formato de estas URLs debe de ser “dirección del servidor “/”operacion”/”variable”, cada variable debe de estar separada por una barra, similar a una dirección de directorios y estas deben de estar siempre en el mismo orden.

Las URL presentadas muestran las diferencias explicadas, pero para nuestro proyecto, la API o ser accedida mediante un navegador, sino mediante la aplicación a desarrollar.

Por otro lado, una URL semántica permite una mejor indexación por los buscadores.

7. Pruebas

Debido a que el proyecto se compone de varias capas, es posible probar cada capa individualmente para facilitar la detección de posibles errores. Posteriormente, se procede a probar todo el conjunto del sistema.

Las pruebas y la detección de errores deben de ser registradas y reproducibles, por ello se han registrado el estado previo de la prueba, las entradas, los resultados y las observaciones. En el apartado solo se muestran algunas de las pruebas exitosas, puesto que las pruebas con resultados anómalos se han corregido durante el proceso.

Las maquinas sobre las que se han ejecutado los servidores son máquinas virtuales VirtualBox, a excepción de la aplicación móvil, que se ha emulado utilizando BlueStacks.

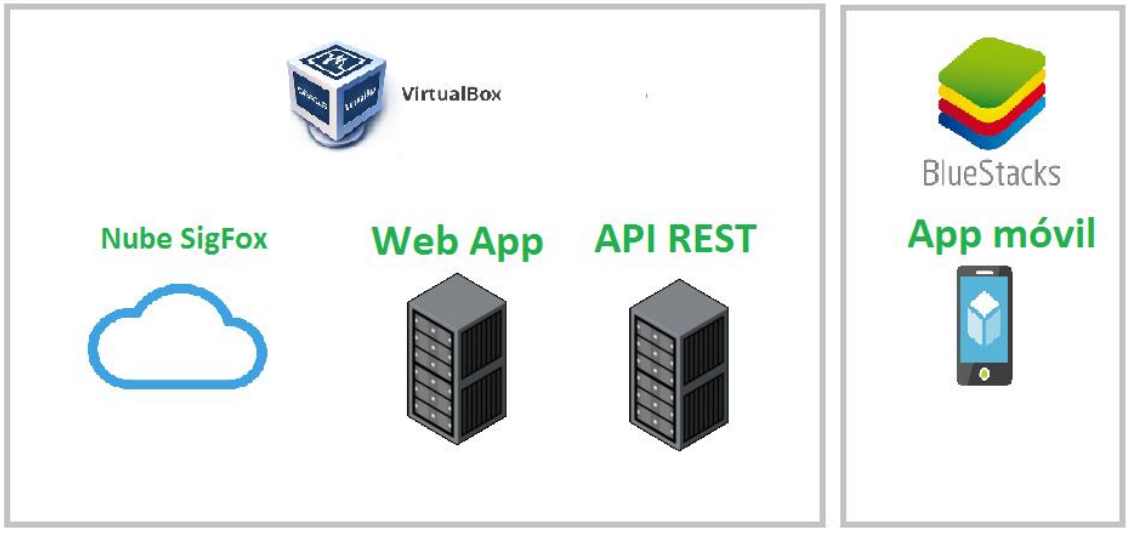

Ilustración 18. Herramientas de virtualización utilizadas.

7.1. Recursos para las pruebas

Algunas de las capas, son desarrolladas por otros departamentos o servido por terceros, como es la nube Sigfox. Esta nube de telecomunicaciones no se encontraba disponible en el momento de realizar las pruebas, por lo tanto, se ha desarrollado un sistema que emula en envío de mensajes de dicha red.

Consiste en una simple aplicación web de una página donde se inserta en un formulario un identificador de dispositivo y sus coordenadas, el sistema recoge los datos, los almacena y los envía a la API REST tal y como lo realizaría el Calzado localizador y la red de comunicaciones Sigfox.

Las tecnologías empleadas para esta capa de pruebas es un sistema LAMP. Una simple base de datos mantiene un registro de los mensajes enviados

7.2. Prueba API REST

La API REST es una capa cuya única interfaz es puerto 8888, este hecho simplifica las pruebas, siendo únicamente necesario un navegador o la herramienta Postman[16]. En las capturas siguientes se muestran operaciones de consulta y sus respuestas.

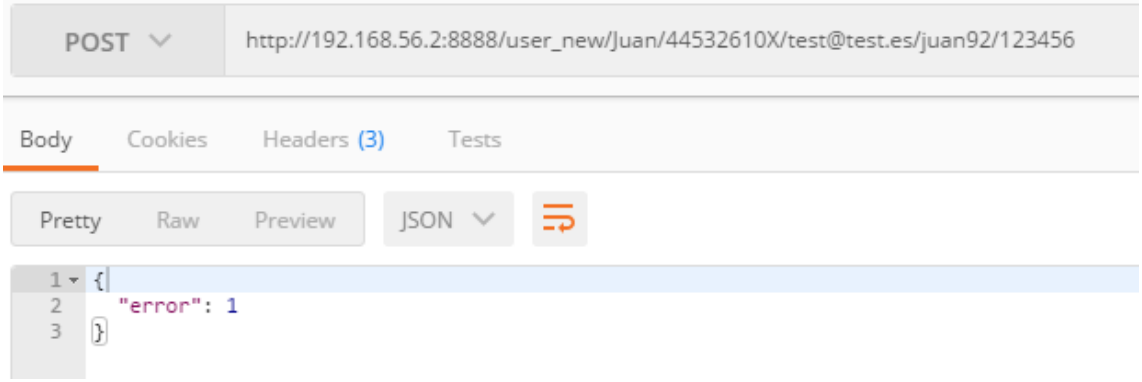

Ilustración 19. Prueba 1 de la API REST.

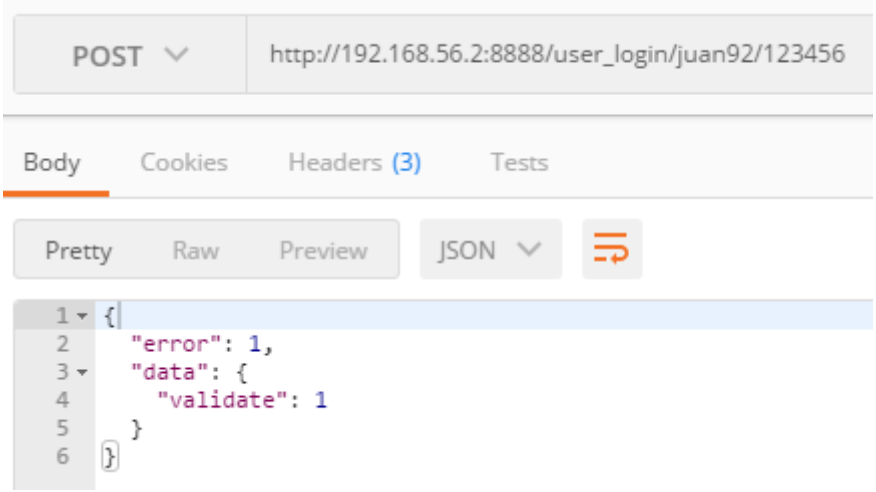

Ilustración 20. Prueba 2 de la API REST.

[16] https://www.getpostman.com/

Caso de prueba.	Creación de nuevo usuario.
Descripción.	Se comprobará la correcta inserción de un nuevo usuario (cuidador) en el sistema.
Estado inicial.	No existe el usuario en la base de datos.
Entrada.	Se envía la siguiente URL con la instrucción de creación de nuevo usuario y sus datos en el formato correcto: http://192.168.56.2:8888/user_new/Juan/44532610X/test@test.es/juan92/123456
Respuesta.	{“error”:1}
Observaciones.	Hay un nuevo documento en la base de datos con los datos del nuevo usuario. La respuesta devuelta es correcta.

Caso de prueba.	Comprobación de credenciales.
Descripción.	Comprobación de la validación de las credenciales de los usuarios, para autenticarlos en las aplicaciones.
Estado inicial.	Existe el usuario de pruebas con todos los datos correctos.
Entrada.	Se envía la URL con la instrucción de validación y las credenciales en el formato correcto: http://192.168.56.2:8888/user_login/juan92/123456
Respuesta.	{“error”:1,”data”: {“validate”:1}}
Observaciones.	La respuesta devuelta es correcta. Los campos del usuario no han sido alteraros.

Caso de prueba.	Recepción de coordenadas de la red Sigfox.
Descripción.	Prueba de recepción de los mensajes enviados de Sigfox, extrayendo y almacenando las coordenadas que en ellos se codifica.
Estado inicial.	Existe el dispositivo “123456” en la base de datos, asignado a un paciente y este un cuidador con un token de Firebase. El dispositivo dispone de otras coordenadas.
Entrada.	Se envía la URL con el mensaje en hexadecimal: http://192.168.57.2:8888/receive_position/123456/312e3132332f312e313233
Respuesta.	No se devuelve respuesta, se envía una alerta al teléfono del paciente.
Observaciones.	El resultado es el esperado, la red Sigfox no requiere de respuesta, y la alerta se ha enviado al usuario por detectar que el dispositivo ha salido de su área.

Las pruebas realizadas sobre la API REST verifican el correcto funcionamiento, siendo muy restrictivo con el formato de entrada de los argumentos.

7.3. Prueba Web App

La Web App es un elemento complejo con una gran cantidad de entradas: La URL, la propia pantalla con los formularios y botones que muestra y la consola del navegador.

El usuario ordinario solo dispone de conocimientos para el uso de la entrada normal de la interfaz y de la URL, de modo que solo se mostrarán las pruebas sobre estos elementos.

Se le otorga tanta importancia a la consola del navegador por que Meteor tiene un importante componente JavaScript en el cliente, el cual, un usuario avanzado puede manipular, aun así, no se pueden realizar acciones hostiles sobre el servidor a través de esta entrada.

Caso de prueba.	Autenticarse y listar usuarios.
Descripción.	
Estado inicial.	La Web App e encuentra en la pantalla de Login, con los campos vacíos, existe en la base de datos un usuario con credenciales y pacientes asignados a él.
Entrada.	Se indican solamente los datos de usuario y contraseña correctos en el formulario de Login y se pulsa Entrar.
Respuesta.	Se ha cambiado página a una con la lista de pacientes y sus opciones, los pacientes son únicamente los del usuario.
Observaciones.	Se confirma en la consola que el usuario se encuentra autenticado en el cliente.

Caso de prueba.	Navegación por URL, lista a lista.
Descripción.	Prueba de navegación de la Web App a través de la dirección URL, en este caso es un refresco de la página de la lista de pacientes con un usuario autenticado.
Estado inicial.	El usuario “juan92” se encuentra autenticado y en la pantalla de lista de pacientes.
Entrada.	Se introduce la siguiente URL en la barra de búsqueda, que es la misma en la que se encuentra: http://192.168.57.3:3000/list
Respuesta.	Se carga la página de nuevo con los mismos pacientes en la lista.
Observaciones.	La Web App ha mantenido al usuario autenticado, pero las variables internas del cliente se han reiniciado.

Caso de prueba.	Navegación por URL, ficha a ficha.
Descripción.	Prueba de navegación de la Web App a través de la dirección URL, en este caso es un refresco de la página de la ficha de un paciente.
Estado inicial.	El usuario "juan92" se encuentra autenticado y en la pantalla de la ficha de datos de un paciente.
Entrada.	Se introduce la siguiente URL en la barra de búsqueda, que es la misma en la que se encuentra: http://192.168.57.3:3000/data
Respuesta.	Se carga la página de nuevo, pero los datos se encuentran vacíos, la pagina sigue reconociendo la sesión del usuario.
Observaciones.	La Web App ha mantenido al usuario autenticado, pero las variables internas del cliente se han reiniciado, es por esto que la chicha se muestra vacía.

Dadas las pruebas, para el correcto funcionamiento de la App el usuario solo hade navegar con la utilización de los botones de navegación de la interfaz. Esta ha de ser diseñada para ser los suficientemente cómoda e intuitiva para que el usuario no acceda a la barra de búsqueda para navegar.

7.4. Prueba Aplicación Móvil

Se han realizado pruebas sobre en esta capa tanto en dispositivo simulados como reales para garantizar que en todos los casos el comportamiento es el esperado.

Caso de prueba.	Cambio de usuario.
Descripción.	Prueba de a aplicación para desautenticar a un usuario, autenticar a otro y verificar que el cambio es correcto.
Estado inicial.	El usuario "juan92" se encuentra autenticado y en la pantalla de la lista de pacientes. Debe de existir un segundo usuario en la base de datos de la API REST.
Entrada.	Se vuelve a la pantalla de menú y se desloguea el usuario, seguidamente se insertan las credenciales del segundo usuario y se accede.
Respuesta.	En la lista de pacientes aparecen los pacientes del segundo usuario.
Observaciones.	La aplicación móvil ha respondido correctamente, de no disponer de conexión con la API no sería posible autenticar al nuevo usuario.

Caso de prueba.	Recepción de notificaciones.
Descripción.	Comprobación de la recepción de las notificaciones de la aplicación y comprobación de mostrar el mapa al abrirla.
Estado inicial.	La aplicación se encuentra cerrada, previamente se ha autenticado al usuario “juan92”.
Entrada.	Se envía dese el servidor una notificación de alejamiento de paciente.
Respuesta.	Se abre una notificación en el teléfono con el nombre del paciente, al abrirse lleva al mapa de la aplicación indicando la posición de este.
Observaciones.	Esta prueba se ha realizado en un dispositivo real, el icono de la notificación aparece por defecto, un cambio futuro indicará la imagen del paciente. Si se pulsa el botón hardware volver del teléfono, cierra la aplicación puesto que no hay pantalla anterior.

7.5. Prueba del sistema completo

Las pruebas con el sistema completo han sido difíciles debido a la potencia necesaria para la emulación de los dos servidores y de la nube Sigfox emulada y del navegador para acceder a la Web App.

Caso de prueba.	Inserción de nueva posición.
Descripción.	Prueba general del sistema completo y test de refresco de la Web App.
Estado inicial.	Existe en la base de datos un usuario con un paciente y este con el dispositivo “123456”, el usuario se encuentra autenticado en la Web App y se encuentra visualizando el mapa del paciente.
Entrada.	Se inserta una nueva posición (11.222, 33.444) para el dispositivo “123456”.
Respuesta.	La nueva posición se inserta exitosamente en la base de datos, pero en la Web App no se muestra la nueva posición como la última inmediatamente.
Observaciones.	El sistema de suscripción de Meteor que envía los datos a una base de datos local del cliente no reacciona directamente si los datos en la base de datos cambian.

Esta prueba no ha sido todo lo exitosa que se esperaba, pero se puede considerar como una limitación de Meteor. El resultado esperado era que, viendo el mapa de un paciente a través de la Web App, los puntos se actualizarían a la vez que el calzado localizador enviaba nuevas posiciones. Se suponía que se realizaría este refresco automático debido que Meteor renderiza la interfaz de los clientes conforme los datos y variables cambian, no es así debido a que Meteor solo detecta los cambios realizados por los clientes, no directamente por la base de datos.

8. Conclusiones y trabajo futuro

8.1. Conclusiones

Gracias al presente proyecto y a Wonkacenter he aprendido la importancia de la correcta planificación de un proyecto antes de su desarrollo, siendo este un paso vital en el desarrollo de un producto software.

También he aprendido la importancia de la correcta elección de una tecnología para un determinado producto. Durante el desarrollo, se me ha propuesto una serie de lenguajes y frameworks a elección para desarrollar "Kune – Calzado conectado", el análisis y justificación de esas propuestas de entornos ha sido parte del desarrollo y por tanto parte de mi aprendizaje, obteniendo una mejor visión crítica de cada lenguaje y en que casos usarlos.

El presente proyecto también ha mejorado mis habilidades de comunicación escrita y a plasmar en un proyecto y de forma estructurada mis ideas y trabajos. Con esto también he percibido la importancia de documentar los trabajos realizados, de forma que estos aumentan de valor con una correcta documentación y permiten repetirlos y resolver problemas similares en el futuro.

La colaboración entre departamentos es un punto importante en el desarrollo a gran escala, siendo necesaria una fluida y contante comunicación para que el resulto sea el esperado. Las reuniones semanales son un recurso muy importante para que el proyecto lleve el rumbo previsto y para una rápida solución de incidencias programáticas.

En último lugar, he comprendido y participado en los pasos desde que surge una idea de negocio hasta que esta se publica, siendo un proceso que lleva una gran planificación y lleno de imprevistos.

8.2. Futuras propuestas

Durante el proceso de desarrollo han surgido distintas propuestas de mejora, las cuales se listan a continuación:

Autenticación en Meteor.

En cuanto a Meteor, las sesiones es un aspecto muy poco desarrollado y que se debe mejorar en el futuro, en su estado actual es desaconsejable construir sesiones a través de esta herramienta.

El programador no tiene control de en que colección se almacenan las cuentas, que datos se pueden almacenar ni impedir que se creen, un cliente puede crear cuentas a través de la consola JavaScript sin limitación alguna. Los únicos datos que se pueden almacenar de cada sesión son usuario, email y contraseña.

Un futuro trabajo puede analizar más profundamente la seguridad de los usuarios en Meteor y cómo administrar los usuarios por parte el programador.

Usuarios en Mongodb

Mongodb no ha sido diseñado para ser usado mediante usuarios, aunque estos pueden implementarse, es posible manipular la base de datos sin autenticarse.

El trabajo debiera realizar un estudio de la dificultad de creación de usuarios, su seguridad y pruebas de conexión a esta.

Restricciones en los elementos de Mongodb

En Mongodb sería de gran utilidad un estándar para limitar los tipos de los campos insertados (por ejemplo, el campo edad solo admite enteros), esto podría lograrse con un campo de metadatos indicando de que tipo debe ser cada elemento, si es obligatorio y otras propiedades.

El posible trabajo era analizar la utilidad y viabilidad de ese estándar, incluyendo consejos para su utilización y pruebas con diversos lenguajes de programación.

9. Apéndices

9.1. Base de datos seleccionada

9.1.1. Problemática:

Para el almacenamiento de la información, se dispone a evaluar dos tipos de base de datos, MySQL y Mongodb:

MySQL es un gestor de base de datos relacional (RDBMS) muy utilizado, el cual estructura sus datos en tablas.

Por otro lado, Mongodb es una base de datos más reciente que almacena los datos en formato JSON, ofreciendo más libertad en los tipos de campos, pero requiere que el programador gestione adecuadamente los campos que se guardan, al guardar elementos con reglas tan poco restrictivas, no se garantiza que los datos devueltos tras una consulta son los esperados si el programador los ha insertado con otro formato, por lo que obliga al programador a cuidar que los datos insertados están en el formato adecuado y validar los obtenidos por una consulta por si la inserción no fue correcta.

Una gran diferencia entre ambos es el método de consulta, MySQL analiza una string con la sintaxis SQL, mientras que Mongodb emplea objetos para la consulta, haciendo a este último robusto frente a inyecciones de código. En contrapartida, también requiere de aprender de una nueva sintaxis.

Como pequeña diferencia, Mongodb no dispone de instrucciones similares al JOIN presente en SQL, complicando así la combinación de tablas.

Finamente, Mongodb no ha sido enfocado a ser operado con usuarios, por lo tanto, configurar estos de forma segura no es sencillo.

9.1.2. Conclusión:

Considerando que el proyecto está realizado en Node.js, la inserción y recuperación de datos en JSON es una ventaja en cuanto a simplicidad de código, comprensión y tiempo de CPU, por lo que finalmente se decide realizar la base datos en Mongodb, aunque requiere un esfuerzo de formación aprende a utilizar la nueva sintaxis.

Otro aliciente para usar Mongodb es la inserción de datos en JSON, dado que los datos son insertados por el usuario, ejecutar las instrucciones en objetos y no en cadenas protege en gran medida la base de datos de la inserción de código (aun validando estrictamente los campos insertados).

Las pruebas realizadas para conectarse a Mongodb mediante autenticación no han sido satisfactorias, pues la configuración de un usuario requiere especificar sus roles por

cada colección, siendo esta una operación laboriosa, además de que es posible operar la base de datos sin autenticarse.

9.2. Meteor y comparativa de frameworks

Una vez realizada la API y la base de datos, se necesita una página web que permita a los usuarios controlar a los pacientes sin la necesidad de utilizar la App. Esta página ofrecerá un control de sesión simple con un litado de pacientes, los cuales el usuario puede monitorizar y los datos básicos de ellos.

Para implementar esta página de administración, se decide emplear Meteor por dos razones: Ha sido diseñado especialmente para aplicaciones de una sola página, como es nuestro caso y también por su facilidad para crear prototipos páginas.

Una característica, que puede ser tomado como ventaja es la metodología de programación, se debe de programar como si se realizara un programa local con interfaz en HTML, en la ejecución, Meteor separa los lados de cliente y servidor, adicionalmente añade una actualización casi inmediata de datos en todos los servidores mediante la técnica suscripción. Por el contrario, dificulta levemente el diseño de las páginas, que es compensado por Bootstrap.

FRAMEWORK

Como ya se ha comentado, para programar en Meteor se necesita uno de los frameworks Blaze, React o angular:

- React:

La inserción de código HTML en la página es muy sencilla, en un constructor de JavaScript se escribe el código HTML con una sintaxis muy parecida (Ejemplo sintaxis: Class => className), y se inserta en un elemento del fichero HTML principal.

React tiene propiedades y estados, los cuales actúan como variables, las propiedades actúan como argumentos cuando una clase es llamada y los estados reconstruyen la página en cuanto estos han sido cambiados (Sin refrescar la página).

React tiene la facilidad de poder insertar código en HTML en cualquier método de JavaScript muy fácilmente,

- Angular:

Al contrario que React, si se desea insertar código HTML en un constructor de JavaScript, es necesario llamar a un fichero que contenga el código deseado, e insertarlo en el fichero HTML del cliente. También existen pequeños cambios en la sintaxis (ejemplo: etiquetas que iteran elementos).

El sistema que presenta Angular crea un código con una gran cantidad de ficheros, siendo necesario muchas llamadas entre ellos y dificultando seguir el código.

- Blaze:

Blaze fue diseñado para Meteor, por lo que es el framework más simple. Cuando se inserta código HTML, es recogido de “templates” (plantillas) que se encuentran en el fichero HTML principal del cliente. En los “templates” también se aceptan etiquetas de control de flujo como condicionales y bucles. Blaze permite tener todo el código HTML en un solo fichero y por otro el JavaScript.

Conclusión:
Blaze y Angular son muy similares en cuanto al uso de clases, variables e inserción de código HTML, que es muy parecido a la programación ordinaria de JavaScript. Por el contrario, React tiene otros tipos de variables como las propiedades y estados, las cuales cuesta comprender inicialmente pero muy potentes.

10. Bibliografía

Aunque se han visitado más referencias, solo se citan aquellas que han tenido relevancia.

Navarro C, Natalia S (2015). "Cómo citar la bibliografía en los trabajos académicos". <http://mpison.webs.upv.es/investigacion_aplicada/textos/como_citar_upv.pdf> [Consulta: 6 de julio de 2017].

Velasco R (2017). "VMware vs VirtualBox. ¿Qué herramienta de virtualización elegir?". <https://www.softzone.es/2017/03/14/comparativa-vmware-virtualbox/> [Consulta: 6 de junio de 2017].

W3Schools. <https://www.w3schools.com/>, <https://www.w3schools.com/php/default.asp>, <https://www.w3schools.com/js/default.asp>, <https://www.w3schools.com/jquery/default.asp>, <https://www.w3schools.com/nodejs/default.asp> [Consulta: 15 de junio de 2017].

Wikipedia. *Bienvenidos a Wikipedia*. <https://es.wikipedia.org/wiki/>, <https://es.wikipedia.org/wiki/URL_sem%C3%A1ntica>, <https://es.wikipedia.org/wiki/React>, <https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador> [Consulta: 2 de junio de 2017].

Meteor. *THE FASTEST WAY TO BUILD JAVASCRIPT APPS*. <https://www.meteor.com/> [Consulta: 2 de junio de 2017].

Mongodb. *MongoDB Atlas Database as a Service*. <https://www.mongodb.com/> [Consulta: 30 de julio de 2017].

MySQL. *The world's most popular open source database*. <https://www.mysql.com/> [Consulta: 15 de julio de 2017].

Brian L (2016). "How to emulate a network using VirtualBox". <http://www.brianlinkletter.com/how-to-use-virtualbox-to-emulate-a-network/> [Consulta: 16 de julio de 2017].

Christian L (2013). "In VirtualBox, how do I set up host-only virtual machines that can access the Internet?". <https://askubuntu.com/questions/293816/in-virtualbox-how-do-i-set-up-host-only-virtual-machines-that-can-access-the-in> [Consulta: 16 de julio de 2017].

Eugeniya K. "MongoDB vs MySQL Comparison: Which Database is Better?". <https://hackernoon.com/mongodb-vs-mysql-comparison-which-database-is-better-e714b699c38b> [Consulta: 30 de julio de 2017].

Macool (2013). "MySQL vs PostgreSQL vs MongoDB (velocidad)". <http://macool.me/mysql-vs-postgresql-vs-mongodb-velocidad/04> [Consulta: 30 de julio de 2017].

JSX. *A faster, safer, easier JavaScript*. <https://jsx.github.io/> [Consulta: 3 de junio de 2017].

Carlos A (2016). "JSX para novatos". <https://carlosazaustre.es/blog/jsx-para-novatos/> [Consulta: 3 de junio de 2017].

Risk Programmer (2016). Lista de reproducción de 8 tutoriales. <https://www.youtube.com/watch?v=lvziXejmJ5g&list=PLQ7rrCblsgaNI73ge-63QQdd0Bln7ZJlq> [Consulta: 3 de junio de 2017].

VMWare. <https://www.vmware.com/es.html> [Consulta: 06 de junio de 2017].

VirtualBox. *Welcome to VirtualBox.org!* <https://www.virtualbox.org/> [Consulta: 6 de junio de 2017].

Manuel K, Herman A. "El Libro para Principiantes en Node.js". <https://www.nodebeginner.org/index-es.html> [Consulta: 23 de julio de 2017].

Ubuntu. <https://www.ubuntu.com/>, <https://help.ubuntu.com/lts/serverguide/git.html> [Consulta: 15 de julio de 2017].

PM2. *Advanced, production process manager for Node.js*. <http://pm2.keymetrics.io/> [Consulta: 31 de julio de 2017].

Ruben V (2015). "VMware Player vs VirtualBox. ¿Quién ofrece mejor rendimento?". <https://www.redeszone.net/2015/08/22/vmware-player-vs-virtualbox-quien-ofrece-mejor-rendimento/> [Consulta: 6 de julio de 2017].

Jose L (2011). "Comparacion entre un linux y windows server (migrar a linux)". <http://www.ubuntu-es.org/node/156452#.WTba3OuLTIU> [Consulta: 6 de julio de 2017].

Git *--distributed-even-if-your-workflow-isnt*. <https://git-scm.com/> [Consulta: 6 de julio de 2017].

Tomas T (2017). "Todo lo que Necesitas Saber Sobre Direcciones URL Amigables". <http://deteresa.com/url-amigables/> [Consulta: 6 de julio de 2017].

Postman. *Developing APIs is hard. Postman makes it easy*. <https://www.getpostman.com/> [Consulta: 6 de julio de 2017].

Angularjs. <https://angularjs.org/> [Consulta: 7 de julio de 2017].

Sacha G (2015). "What's Going On With Blaze, React, and Meteor?". <https://www.discovermeteor.com/blog/blaze-react-meteor/> [Consulta: 7 de julio de 2017].

David Burles (2015). "How to create a reactive Google map". <http://meteorcapture.com/how-to-create-a-reactive-google-map/> [Consulta: 27 de junio de 2017].

Jorge Mahecha Durango. (2017). Título del TFG (MeteorJS: Un framework full-stack para la creación de aplicaciones web real-time.) UPC.

Francis' Verlag (1999). "*JavaScript Praxisbuch*". Zaragoza, Marconbo.

Mike Cantelon, Marc Harter, T.J.Holowaychuk, Nathan Rajlich (2014). "*Node.js inAction*". New York, Manning Publications.