Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

SISTEMA PARA RASTREO DE CONTAGIOS EN ENFERMEDADES INFECCIOSAS

Alumno: **Pedro Ruiz Valverde**

Tutor: Antonio Jesús Rueda Ruiz
Dpto: Informática

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

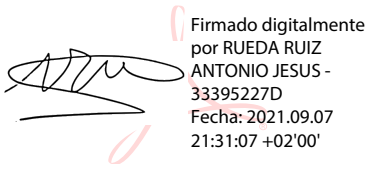
Don **Antonio Jesús Rueda Ruiz** , tutor del Proyecto Fin de Carrera titulado: **Sistema para rastreo de contagios en enfermedades infecciosas**, que presenta **Pedro Ruiz Valverde**, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Septiembre de 2021

El alumno:

Pedro Ruiz Valverde

Los tutores:



Antonio Jesús Rueda Ruiz

Indice

1. Introducción

1.1. Introducción

Durante la pandemia por Covid-19, se han explorado diversas formas para rastrear y aislar posibles contagios. El rastreo permite localizar e informar a las personas que puedan haber estado expuestas. De esta manera, se consigue aislar los círculos cercanos de posibles contagiados desacelerando de manera significativa la propagación del virus.

Este rastreo lo realizan los rastreadores. Estos son operarios humanos que se encargan, por una parte, de buscar contactos cercanos a los infectados y, por otra, de contactar con cada uno de ellos para tomar las medidas oportunas. Esta tarea puede tomar bastante tiempo porque es posible que el infectado se haya olvidado de algún contacto, también pueden producirse equivocaciones por parte del informador y si la carga de trabajo es muy alta es posible que no se pueda contactar con todos en un tiempo razonable. [1]

Para conseguir afrontar el proceso de rastreo y notificación a los usuarios se han diseñado y puesto en marcha sistemas informáticos que permiten, por una parte, detectar todos los contactos entre personas y por otro, determinar el grado de riesgo de forma automática. Esto facilita en parte el trabajo de los rastreadores.

Vamos a investigar cómo funcionan estos sistemas, qué protocolos han utilizado y la forma para determinar quiénes son los posibles contagiados. Desarrollaremos una aplicación móvil que monitorice los contactos cercanos y el servidor que registre y muestre estos datos a los rastreadores.

1.2. Motivación

El desarrollo de una herramienta que sea útil para la monitorización de la propagación puede ser crucial si se usa adecuadamente. Ayudaría de manera muy significativa en el descenso de contagios. No solo hablamos del virus Covid-19 sino que podría ser cualquier otro tipo de virus o infección que esté por venir.

Es por ello que disponer de herramientas funcionales es indispensable para enfrentarnos a posibles nuevas situaciones epidemiológicas.

1.3. Objetivos

El objetivo de este trabajo de fin de grado es obtener una herramienta final con la que poder rastrear y detectar posibles contactos cercanos, facilitando el trabajo al personal sanitario.

A través de esta herramienta no se busca reemplazar a los rastreadores, ya que son una parte esencial. Pero sí facilitar la información que un rastreador necesita, sin necesidad de que tenga que buscarla ni obtenerla a través de terceras personas. Con esto conseguimos que el rastreo sea eficaz, porque una persona puede olvidar o no querer nombrar a ciertos individuos por diversos motivos.

2. Sistemas de rastreo

2.1. Rastreo de contagios durante la crisis del Covid-19

A lo largo de este tiempo, se ha visto como algunos países han utilizado sistemas masivos de rastreo, los cuales han conseguido de manera radical terminar con la propagación del coronavirus, por ejemplo, Corea del Sur.

Corea del Sur solicitó a los principales operadores móviles las listas de los usuarios que hubieran estado en la zona de un brote. Consiguió ubicar a 10.905 usuarios a los cuales se les notificó por mensaje de texto que debían presentarse a un test de Covid-19. También se utilizaron registros de pagos con tarjeta bancaria para encontrar a más ciudadanos [2].

Esta decisión no fue bienvenida debido a los problemas de privacidad que implican. Por ello, los expertos siempre han apoyado una solución que ofreciera privacidad y a la vez permitiera notificar a los usuarios de un posible contagio.

Se han apostado por sistemas menos intrusivos. España ha puesto a disposición del ciudadano la aplicación RadarCovid, esta app intercambia códigos mediante *Bluetooth* con dispositivos cercanos. Los códigos son anónimos y regenerados cada 10 minutos, de esta manera la información intercambiada es anónima y no está asociada a ningún perfil. En el caso de que alguien dé positivo las autoridades de salud le proporcionan un código de verificación que debe introducir en la aplicación [3].

Estas aplicaciones presentan otros tipos de inconvenientes. Al ser voluntarias, no todo el mundo las utiliza. Los expertos avisan que es necesario que más de un 66% de la población la tenga activa para que tenga un efecto significativo en el

descenso de contagios. Lejos de la realidad, la mayoría de países se mantienen muy por debajo de ese rango [4].

2.2. Compromiso entre privacidad y funcionalidad.

Al final terminamos en un gran debate. Es necesario que la mayor parte de la población utilice la aplicación para que pueda ser útil, pero, al no ser obligatoria se debe de intentar preservar de la mejor forma posible la privacidad, para que los usuarios confíen en ella y se la descarguen.

A mayor nivel de privacidad más complicado es centralizar el uso de la app para extraer el listado de personas que pudieran estar expuestas. Es por esto que al final, la alerta de la aplicación es simplemente un indicativo y que el usuario puede ignorar desplazando la notificación hacia la derecha en su móvil, siendo a veces inútil el esfuerzo puesto en este tipo de aplicación.

Es por esto que sistemas que comprometen más nuestra privacidad funcionan mejor. El control es centralizado y se pueden tomar acciones en aquellas personas que puedan estar contagiadas.

Habría que llegar a un acuerdo final con el usuario, por el cual se consiga sacar el máximo provecho a una app de este tipo y a la vez de confianza al mismo.

2.3. Estado del Arte

Existen multitud de arquitecturas que resuelven este problema de distintas formas. La mayoría de protocolos apuestan por un sistema descentralizado en donde el aviso solo lo puede recibir el cliente y no es procesado por el servidor. En cambio, otros apuestan por un sistema centralizado en el que la administración de salud pueda obtener datos sobre contactos.

2.3.1 Conceptos de un sistema de rastreo

Vamos a describir las partes clave de una aplicación de rastreo desde un punto de vista amplio. La mayoría de protocolos están divididos en dos tareas.

- Por un lado tenemos la detección de dispositivos (llamado Device Handshake). Esta tecnología se encarga en detectar la distancia y tiempo de exposición con alguien. Para ello se pueden utilizar varias técnicas entre las que entran tecnologías GPS, WiFi, Bluetooth…

  Actualmente, la mayoría de protocolos utilizan BLE (Bluetooth Low Energy) por el poco impacto que tiene sobre la batería. Otros sistemas como por

ejemplo GPS puede tener un alto gasto de batería ademas de ser impreciso si estamos en zonas sin cobertura.

Utilizar BLE tiene como ventajas poder contactar con dispositivos cercanos sin necesidad de saber la localización del dispositivo, respetando la privacidad.

- Por otro lado tenemos el encargado de reportar los casos positivos. Puede ser un sistema centralizado: cuando un servidor central es el encargado de procesar y notificar los posibles contactos, o descentralizado: cuando es el cliente el que tiene que comprobar si el dispositivo ha estado en contacto con usuarios contagiados. Para ello, se tienen listas con identificadores de los casos positivos.

  Cada uno tiene sus ventajas y desventajas. Dependiendo del proyecto se terminará adoptando uno u otro en base a los requisitos que debe cumplir.

2.3.2 Protocolos actuales

Durante la pandemia producida por la Covid-19 se han diseñado y desarrollado diversos protocolos para el rastreo por proximidad, cada uno con características únicas que afrontan el mismo desafío desde puntos de vista diferentes. No quiere decir que haya uno mejor ni peor; por ejemplo, en el caso de Corea del Sur, la efectividad de las técnicas utilizadas fueron muy altas, aunque por otra parte atentan gravemente contra la privacidad de sus ciudadanos.

Estos son algunos de los protocolos más conocidos que trabajan de manera muy similar.

- **DP-3T** [5]: Este sistema es anónimo y descentralizado. Envia la mínima información necesaria al servidor. No es posible rastrear a alguien que no haya sido positivo, y de igual forma, desconoce cualquier tipo de información de los usuarios. Utiliza códigos pseudoaleatorios criptográficos como identificación.

  Su propósito está limitado exclusivamente a notificar a aquellos usuarios que han estado en contacto cercano con un positivo.

  Su funcionamiento es el siguiente. Cada usuario emite por BLE un identificador efímero (EphIDS) que es cambiado cada 10 minutos. Este identificador es recibido por dispositivos cercanos y junto a él se guarda el tiempo y la cercanía de exposición.

Los EphIDs son generados por una función pseudoaleatoria derivada de la clave secreta (SK) del cliente. La clave secreta rota diariamente mediante $SK_t = H(SK_{t-1})$ siendo H una función Hash.

En caso de que un usuario dé positivo. El cliente enviará la clave Sk_t del día que se considera contagioso. El servidor distribuirá esta clave por todos los demás usuarios. Estos serán los encargados de generar todos los EphID derivados de la clave y comprobar si han estado en contacto.

Es por esto, que el servidor no tiene que hacer ningún tipo de procesamiento, su única función es la distribución de claves y verificación de que provengan de un positivo real.

- **PEPP-PT** [6][7][8]: Desarrollado para que sea adoptado por la mayor cantidad de países Europeos posibles. Este protocolo a diferencia de DP-3T, es centralizado. Es decir, un servidor controla y administra los posibles contagios. Aun siendo centralizado, busca mantener todo lo posible la privacidad del usuario.

  Aparte de notificar al usuario de un posible contagio, una entidad externa puede ver y controlar que usuarios han estado expuestos. Para poder utilizar este sistema es necesario identificarte ante la entidad.

  El funcionamiento es el siguiente:

  El usuario se registra en la aplicación. La aplicación avisará al servidor el cual le entregará un ID permanente y una lista de Ephemeral Bluetooth Identifiers (EBDIS). El servidor mantendrá un registro de los EBDIS otorgados a cada ID que consultará en caso de detectar un positivo.

  El dispositivo utilizará BLE para descubrir y mostrar EBDIS. Se guardará en la memoria interna del dispositivo información relativa al contacto: tiempo de exposición, velocidad del usuario, conexión bluetooth… Los EBDIS son de un único uso y se irán cambiando cada cierto tiempo, una vez que el cliente se quede sin EBDIS, deberá solicitar nuevos códigos al servidor.

  En caso de que uno de los usuarios sea positivo deberá de obtener una verificación por parte de la entidad sanitaria para reportarlo. Enviará una lista de EBDIS obtenidos por BLE al servidor.

  Las aplicaciones deberán consultar al servidor si han estado en contacto, recibiendo una respuesta del número de contactos.

- **TCN Protocol** [9]: Utiliza un sistema de rotación de claves parecido a DP-3T. Los datos se mantienen en local. Mientras que un sujeto no de positivo, sus

contactos no son expuestos, por tanto es imposible rastrearles. El funcionamiento es parecido a DP-3T, ademas es un sistema distribuido.

Su funcionamiento al igual que los dos protocolos anteriores es mediante la emisión de códigos mediante BLE. Los códigos que se propagan son generados mediante la derivación de la clave (rak) junto a una clave de verificación (rvk), se obtienen dos valores: $tck_0 = H(rak)$; $tck_n = H(rvk \,||\, tck_{n-1})$, donde *"H"* es una función hash y *"||"* significa concatenar.

La clave tck_n se rota cada día y de esta se obtienen los TCN mediante $tcn_i = H(i \,||\, tck_i)$. Son estos los códigos que se envían por BLE para controlar los contactos.

Para poder recrear la secuencia es necesario enviar la clave rvk junto al tck_{j1}. A partir de estos valores es posible obtener toda la secuencia posterior a tck_{j1}. Donde J1 es el punto desde el que se quiere obtener la secuencia posterior.

Estos valores son distribuidos a los usuarios para que comprueben si han estado en contacto con un positivo.

Existen otros protocolos utilizados para el rastreo empleando todo tipo de medios y técnicas. Los nombrados anteriormente comparten una característica: utilizan BLE para la emisión de códigos pseudoaleatorios. Podría utilizarse también GPS o el propio WIFI para identificar los contactos, pero al tener que utilizar dispositivos móviles se utilizan técnicas que economicen el consumo. BLE cumple perfectamente esta propiedad.

Además, vemos como la mayoría buscan la distribución de códigos pseudoaleatorios. Con esto consiguen mantener la privacidad del usuario y dificultar el seguimiento de usuarios con fines maliciosos.

3. Diseño

3.1. Objetivos

El objetivo de este proyecto es ayudar en la prevención de contagios de enfermedades infecciosas. Un equipo de rastreadores podrá visualizar qué personas han estado en contacto. De esta manera se puede empezar un aislamiento preventivo hasta que se confirme el contagio.

Se realizará el diseño e implementación de un sistema para la monitorización de propagaciones de enfermedades. Es necesario que sea capaz de rastrear los contactos entre personas y por otro lado mostrar una lista informativa a los rastreadores.

Otro punto a favor a la hora del desarrollo será utilizar la información recolectada para estudios epidemiológicos.

3.2. Alcance

El resultado que pretendemos obtener es un prototipo software. Se cubrirán las necesidades mínimas en cuanto a funcionalidad para el rastreo y monitorización.

3.3. Captura de requerimientos

Sin descuidar la privacidad de los usuarios apostaremos por la eficacia del sistema. Es por ello que todos los usuarios deberán estar identificados con datos personales, para que el equipo de rastreadores puedan llamarles si fuera necesario.

Los contactos serán registrados y guardados para su posterior revisión.

Los **requisitos funcionales** del proyecto serán los siguientes:

- Todo usuario debe estar perfectamente identificado.
- La a*plicación* móvil mantendrá los datos de contacto en privado, siempre que el usuario no introduzca un código de verificación válido.
- Cuando haya un contacto, la aplicación podría guardar la posición GPS. Esto será configurable en la propia aplicación.
- El registro del usuario se realizará al abrir la aplicación por primera vez
- Los rastreadores solo podrán ver información relativa a personas con una prueba positiva.

- Los sanitarios deben confirmar el contagio del usuario.
- Los sanitarios podrán comprobar con cuantos contactos ha estado un usuario.
- Los administradores del sistema podrán crear usuarios nuevos o borrar la información almacenada de ellos.
- Los usuarios normales no tendrán acceso al panel de control.
- Cualquier usuario podrá ver estadísticas anónimas sobre el uso de la aplicación (usuarios registrados, número de contactos registrados...).
- La activación y desactivación del rastreador debe de ser accesible de manera sencilla.
- Un usuario no debe de poder acceder a partes del sistema en los que no tiene permiso.
- Un usuario solo podrá ser añadido de manera manual al sistema.

Los **requisitos no funcionales** son:

- El acceso a los datos debe ser rápido y dar respuesta en un tiempo breve.
- Debe ser posible la distribución de la carga en varios servidores.
- El coste del despliegue debe de ser bajo ante una cantidad limitada de usuarios.
- La aplicación móvil debe gastar poca batería.
- Debe poder internacionalizarse.
- Detección de proximidad sin interacción humana.

3.4. Propuesta de la solución

En esta sección, se comentará la solución propuesta en base a los requisitos obtenidos anteriormente.

Se utilizará un sistema similar a **PEPP-PT**. El servidor será el encargado en generar los códigos efímeros que utilizarán los clientes móviles para detectar los contactos. Los códigos son aleatorios para que un observador externo no pueda identificar al usuario que los está emitiendo pero sí será conocido por el servidor. El código

efímero solo se emitirá durante un periodo breve de tiempo, no podrá usarse dos veces.

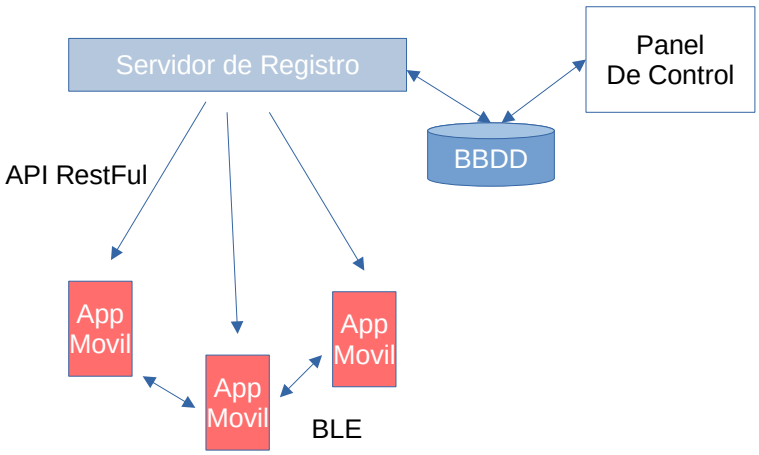
En caso de un contagio, el registro de contactos del usuario en cuestión será enviado al servidor para su revisión por parte de los rastreadores.

Comenzaremos mostrando la arquitectura de la aplicación para después especificar sobre cada uno de los elementos.

3.4.1 Arquitectura general

Necesitaremos tres elementos en este proyecto, por un lado una **aplicación móvil** que será la encargada de encontrar dispositivos cercanos y guardarlos en la memoria interna del teléfono. Un **backend** llamado servidor de registro que será el encargado de recibir los reportes de la aplicación móvil en caso de notificar un caso positivo. Y un **panel de control** el cual mostrará información a los rastreadores y sanitarios. Además, este panel permitirá a un administrador añadir nuevos usuarios.

El **panel de control** y el **backend** tendrán acceso a una base de datos donde guardarán la información recolectada de los usuarios. El acceso directo a la base de datos no es lo más recomendable por los problemas de seguridad que pueda ocasionar. Aun así, se usará este esquema para mantener la simplicidad de este prototipo. Los usuarios móviles tendrán que utilizar el servidor de registro para poder guardar los datos en la base de datos.

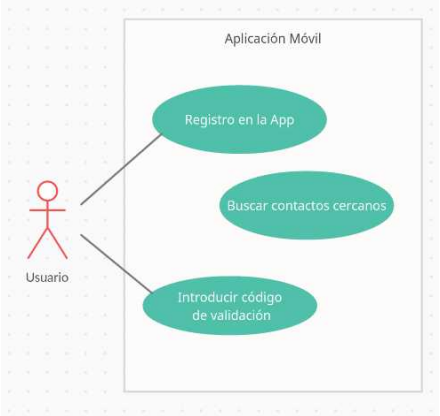


3.4.2 Subproyectos

El trabajo se ha dividido en tres subproyectos para facilitar la implementación. Tenemos la aplicación Android, el servidor de registro y el panel de control.

3.4.2.1 Android App

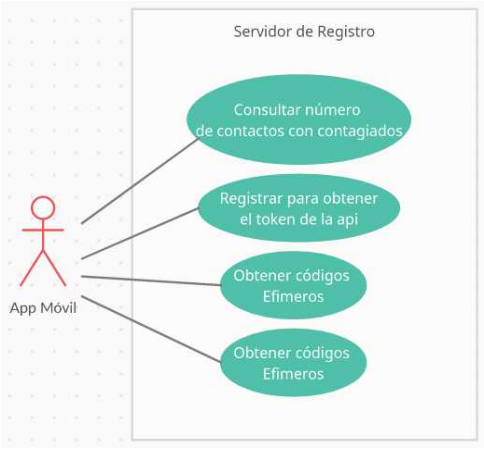
La aplicación móvil tendrá una interfaz gráfica simple. Su función principal es registrar los contactos con otros dispositivos cercanos, en caso de que un usuario introduzca un código de validación correcto enviará un reporte con los contactos de los últimos 14 días.



3.4.2.2 Servidor de registro

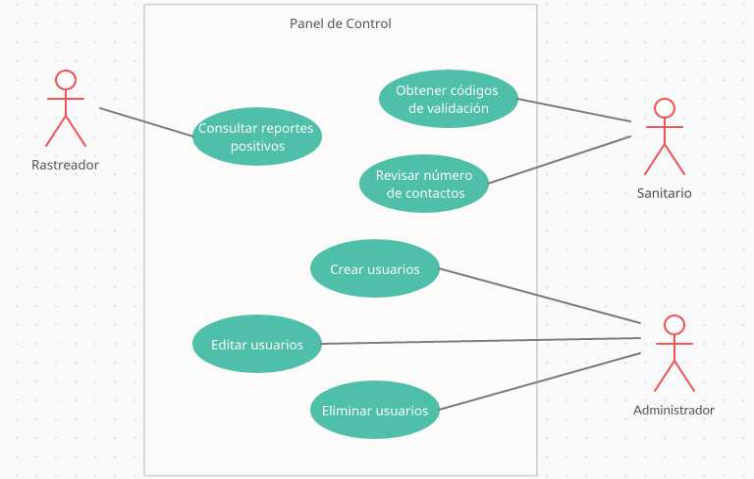
Será utilizado como backend por la aplicación móvil para obtener y enviar los siguientes datos:

- Obtener códigos bluetooth nuevos.
- Obtener notificaciones de posible contacto.
- Enviar códigos bluetooth en caso de ser positivo.
- Registrarse en el servidor a través del token dado a un usuario.



Para una descripción más detallada véase Intercambio de mensajes entre App Móvil y Servidor de Registro.

3.4.2.3 Panel de Control



Es una web web que nos permitirá registrar a nuevos usuarios y hacer un seguimiento de los contactos cercanos. Para diferenciar el tipo de usuario que accede a la web se utilizarán roles (ROLE_USER, ROLE_TRACKER, ROLE_SANITARY, ROLE_ADMIN). Cada uno de ellos accederá a un subconjunto de la aplicación web.

- El rastreador (ROLE_TRACKER) podrá, de manera rápida y eficiente determinar qué personas deberán estar en cuarentena. A diferencia de otros sistemas de seguimiento, el rastreador tendrá acceso directo a los contactos. Desde un buscador, podrá seleccionar a un usuario que haya reportado ser positivo y ver todos sus contactos. Un rastreador podrá marcar a los contactos como revisados o no.
- El equipo sanitario (ROLE_SANITARY) podrá realizar dos funciones. Una de ellas será revisar si un usuario ha estado en contacto con un positivo y su probabilidad de contagio, para así determinar de manera rápida si sus síntomas pueden estar relacionados. La otra función será dar el código de validación a los usuarios.
- El admin (ROLE_ADMIN) será el encargado de administrar la creación de estos usuarios.
- Los usuarios (ROLE_USER) no necesitan acceso a la página web, pues las funciones que deben de realizar serían desde la aplicación móvil.

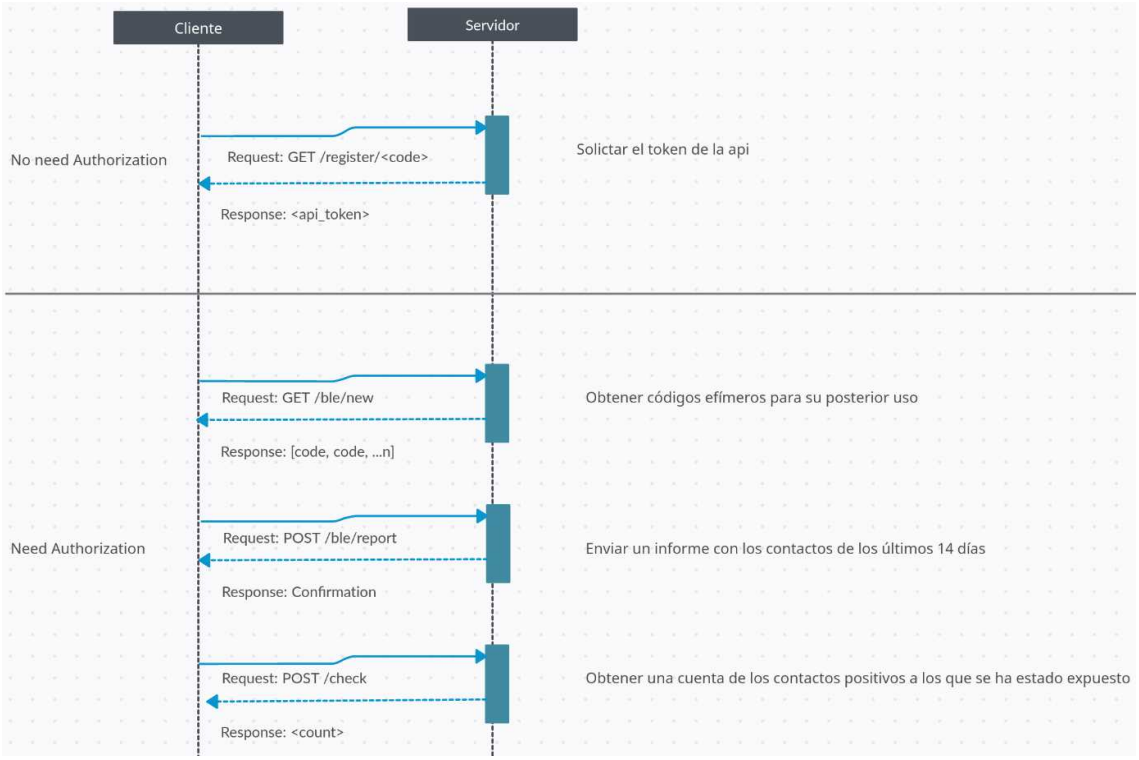
El acceso de los distintos usuarios se realizará desde el mismo portal. Dependiendo del tipo de usuario, se desbloqueará el acceso a cada una de las zonas, pudiendo

existir usuarios con acceso completo a todas las funciones, siempre y cuando tenga todos los roles necesarios.

En el desarrollo de la web se ha pensado en la internacionalización, todas las oraciones y frases de la interfaz deben poder ser traducibles.

3.4.3 Intercambio de mensajes entre App Móvil y Servidor de Registro

Para el intercambio de mensajes utilizaremos una API RestFul que será consultada por el cliente móvil. Utilizaremos 4 tipos de mensajes diferentes ilustrados en el siguiente diagrama.



- **Registro de App**: Este mensaje se enviará la primera vez que se utilice la app móvil. A través del token de verificación un usuario obtendrá el tokén de la aplicación que será el que utilice la aplicación para verificar su identidad.
- **Obtener códigos bluetooth:** La aplicación móvil deberá solicitar nuevos código bluetooth al servidor cuando utilice todos los usados anteriormente. Aunque, podría solicitar nuevos códigos sin necesidad de utilizarlos todos.

- **Notificar positivo:** El cliente enviará un mensaje con un código de validación correcto junto a todos los códigos bluetooth externos que haya ido registrando. El servidor guardará todos estos contactos.
- **Comprobar contactos:** El cliente comprobará si ha sido contacto con alguno de los infectados. En caso positivo devolverá el número de contactos.

Tanto las solicitudes como las respuestas se harán utilizando JSON. Para una descripción más detallada, véase Comunicación API RestFul.

3.4.4 Detección de dispositivos próximos

Para la detección de dispositivos próximos debemos de utilizar las tecnologías que traen incorporadas los teléfonos móviles: Red Móvil, WiFi, Bluetooth, GPS...

Uno de los requisitos que debemos de cumplir es un gasto bajo de batería, es por ello que, utilizar GPS o WiFi podría ser un problema. Los escáneres mediante WiFi están limitados por el sistema Android además de presentar un consumo alto. Por otro lado, GPS aparte del consumo tiene el problema de que debe de comparar la ubicación para calcular la distancia, teniendo que compartir todas las ubicaciones en las que ha estado el usuario. Esto genera un problema de privacidad el cual no podemos asumir. Lo más acertado sería el uso de Bluetooth en su modo Low Energy.

Debemos también asegurarnos de que los cálculos realizados sean precisos y que no considere un contacto cuando un usuario esté muy alejado de otro. Por ello, no podemos depender de la red de telefonía pues el error que tiene puede variar entre un centenar de metros a varios kilómetros. El GPS calcularía falsos contactos como por ejemplo cuando dos usuarios estén en el mismo edificio pero diferente planta. Sí serían buenos candidatos Bluetooth y Wifi pues se puede hacer un cálculo estimado debido a que su alcance es corto.

Para el intercambio de mensajes utilizaremos beacons con el formato EddyStone. Para más información véase Beacons.

3.4.4.1 Intercambio de Identificadores Efímeros

Los beacons serán el medio por el cual intercambiemos la información. El identificador será generado por el backend de forma aleatoria. De esta forma, un usuario obtendrá una cierta cantidad que dejará guardados en la memoria del dispositivo. Los emitirá a intervalos fijos de tiempo y desechará una vez que termine su uso, de este modo un mirador externo solo podrá rastrear a una persona durante este tiempo. Los códigos solo serán rastreable por el servidor que los generó y el propio dispositivo.

Para un observador externo le será imposible vincular el identificador con una persona física.

3.4.5 Reportar un positivo

Para reportar un positivo, es necesario que un sanitario verifique esta condición. De esta manera se elimina la posibilidad de que usuarios maliciosos envíen el reporte incorrectamente.

Solo los sanitarios son capaces de generar un código de validación para el usuario. Este código sera introducido en la aplicación móvil la cual enviara los contactos de los últimos 14 días.

3.4.6 Base de datos

Usaremos una base de datos orientada a grafos. Este tipo de base de datos se adapta perfectamente al proyecto, la principal característica que vamos a guardar son las relaciones entre contactos. El acceso a este tipo de información es mas sencilla y rápida en una base de datos orientada a grafos debido a que las relaciones son guardadas de manera implícita.

Con consultas más simples se pueden obtener mejores datos pues el lenguaje que utiliza llamado “Cypher” esta focalizado en navegar a través de los nodos y relaciones. Una consulta de este tipo en una base de datos relacional puede llegar a ser muy compleja y poco intuitiva además de costoso en computación.

Se pueden aplicar algoritmos para grafos que exploran los diferentes caminos entre nodos utilizando las relaciones. Esto permite calcular de manera óptima el camino más corto entre dos contactos, distancia entre nodos o agrupaciones en un tiempo razonable.

Toda esta información puede ser usada para estudios epidemiológicos. [10]

3.5. Planificación y presupuestos

La realización de este proyecto será en los meses de Junio, Julio y Agosto estando terminado para septiembre. La realización del trabajo se dividirá en cuatro partes, por un lado se realizará una investigación sobre otros proyectos de características parecidas y se realizará el diseño inicial del proyecto. Esta parte comprenderá las tres primeras semanas de junio, del 1 al 19 de junio.

En la segunda parte se desarrollarán los subproyectos necesarios desde el 26 de junio hasta el 13 de agosto. Teniendo una semana, del 17 de julio al 25 de julio, sin actividad por motivos personales. La decisión de asignar más tiempo al desarrollo es por la inexperiencia en algunos de los frameworks utilizados.

En la tercera parte del 16 de agosto al 20 de agosto se harán pruebas de funcionamiento.

Para finalizar, se realizara una corrección de la documentación del 23 al 27 de agosto.

Con la estimación temporal, se calcularán los costes económicos del proyecto. El número de horas totales trabajadas teniendo en cuenta que serán 5 horas diarias: 12 semanas * 5 días * 5 horas = 300 horas.

Estas horas están dividas en dos tipos de trabajo: el trabajo de analista informático, que toma las tres primeras semanas, y, por último, el trabajo de programador.

El tiempo total de analista ha sido de 3 semanas * 5 días * 5 horas = 75 horas. Y el de desarrollador 8 semanas * 5 días * 5 horas = 225 horas. Comprobando algunos

portales de trabajo obtenemos fácilmente el precio/hora de los dos trabajos: para desarrollador varía mucho dependiendo si es senior o junior, en nuestro caso al ser junior cobraremos 10€/h. De analista de software el precio es un poco superior, al ser junior cobraremos lo mínimo que se sitúa entorno a los 11€/h.

Además del tiempo de trabajo, tenemos que añadir los costes indirectos, la amortización del material y el beneficio que deseamos obtener.

Para los costes indirectos añadiremos un 10%. Para la amortización de bienes tendremos en cuenta los elementos con los que hemos trabajado: un ordenador 1000€ y dos pantallas externas 200€ que hace un total de 1200€. La amortización del equipo será en 5 años por lo que pertenece un 12 semanas / (5 años * 52 semanas) = 4,61% de su uso. Calculamos la cantidad de dinero que pertenece por el tiempo de uso: 1200€ * 4.61% = 55,38€.

En la siguiente tabla desglosaré los gastos totales:

Descripción	**€ / hora**	**Duración horas**	**Precio**
Analista	11	75	825 €
Desarrollador	10	300	3.000 €
Amortización de bienes			55,38 €
Subtotal			**3.880,38 €**
Gastos Indirectos 10%			388,03 €
Margen de beneficio 15%			582,05 €
Total			**4.850,46 €**
		+ IVA (21%)	**5.869,05 €**

El presupuesto final es de **5.869,05 €.**

4. Detalles del proceso de desarrollo

Para la realización de este proyecto se va a utilizar una metodología de trabajo ágil llamada Scrum. Como el proyecto solo se desarrollara por una persona, algunas partes de la metodología se omitirán al no ser necesarias. Los autores ofrecen de manera gratuita una guía que puede ser consultada en la siguiente dirección: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf

El trabajo se dividirá en sprints que son periodos breves de tiempo en los que se completa un trabajo definido. Cada sprint debe de organizarse antes de comenzar y definir claramente las metas teniendo en cuenta el tiempo disponible para completarlas.

Debido a que cada sprint comienza con una planificación, es posible adaptar y aprender según se desarrolla el software. Los sprints se adaptarán a las necesidades encontradas en cada momento y se motivará a completarlas en el tiempo establecido [11].

4.1. Sprint 1

Para el primer sprint comenzaremos por el **servidor de registro**. Esta parte es crítica para hacer pruebas en la aplicación móvil. Este sprint durará dos semanas, desde el día 21 junio al 4 de julio. El desarrollo se realiza en PHP con el Framework Symfony.

La primera semana se utilizará para aprender sobre Neo4j y Symfony, mientras que la otra se utilizará para realizar el proyecto. La primera semana veremos aspectos básicos de funcionamiento de Symfony, gestión de usuarios, gestión de firewalls, creación de controladores, etc. En cuanto a Neo4J aprenderemos la sintaxis Cypher parecida a SQL y bastante sencilla. La segunda semana se empleará para realizar el desarrollo y testear la API con Postman.

Los objetivos de este Sprint son:

1. Crear un proyecto PHP utilizando composer.
2. Preparar un servicio en Symfony para el acceso a los datos utilizando Neo4J.
3. Preparar el servicio de autentificación.
4. Realizar los controladores para la API.

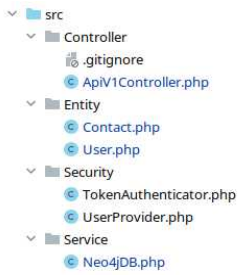
4.1.1 Desarrollo

Utilizaremos el IDE PhpStorm para la realización de este proyecto. Este integra una serie de herramientas que nos permitirá de manera muy sencilla preparar el esqueleto de la aplicación a través de composer.

Creamos un proyecto nuevo a través del asistente, seleccionaremos que es un proyecto de composer. En el siguiente cuadro seleccionamos que queremos utilizar la aplicación *symfony/skeleton.* Esta opción nos preparará la base.

Otro de los puntos que hay que tener en cuenta, es la integración de la base de datos. Symfony trae compatibilidad por defecto con las bases de datos de SQL. Estas son controladas a través de Doctrine que contiene un ORM y DBAL. En nuestro caso al utilizar una base de datos de grafos, necesitamos añadir una dependencia a través de composer, la librería *laudis/neo4j-php-client*. Además tendremos que preparar el sistema de usuarios para que obtenga los datos desde allí.

Para manejar los datos, vamos a crear un servicio en Symfony que será inyectado como dependencia en las clases que lo necesitemos. De este modo podremos abstraer de cierta manera el guardado o la obtención de datos, permitiendo un código más limpio en el controlador ademas de no mezclar la lógica de la aplicación con las peticiones de datos.



Se han utilizado las clases *Service/Neo4jDB.php* para el acceso a la base de datos. Y las siguientes clases *Security/TokenAuthenticator.php* y *Security/UserProvider.php* para abstraer el acceso a los usuarios por parte de symfony. Ha sido una de las partes más complicadas a la hora del desarrollo de la API, ya que hay que seguir estrictamente las indicaciones de Symfony.

El desarrollo del controlador ha sido trivial y no se entrará en detalle.

4.1.2 Test de la API utilizando Postman

Como ya se comentó previamente, no tenemos todavía disponibles los clientes móviles. Por ello vamos a utilizar postman para generar consultas y probar que la api funciona correctamente.

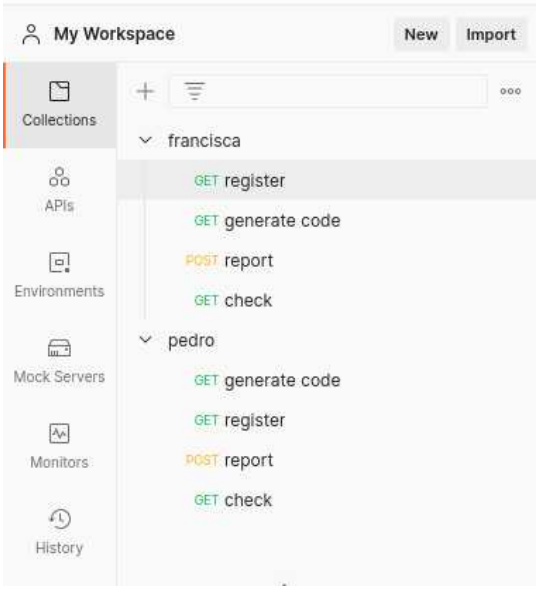
Introducimos dos usuarios a la base de datos con los datos mínimos necesarios para que funcione correctamente.

```
CREATE (:User
```

```
  {name:"Pedro", dni:"12345C", role:"ROLE_USER", register_token:"abcdfg"})

 CREATE (:User
  {name:"Francisca", dni:"54321C", role:"ROLE_USER",
register_token:"123456"})
```

Nos vamos a postman y creamos dos colecciones, una que contenga los credenciales para el usuario *Pedro* y otra para el usuario *Francisca*.



Registramos a los usuarios. Esto nos devolverá el token para loguearnos en la API. Por ejemplo, este es el Json devuelto para Francisca.


```
{
    "api_token": "4a7e66a16e73a8dfb68cc8641f38441f",
    "msg": "Correct register code"
}
```

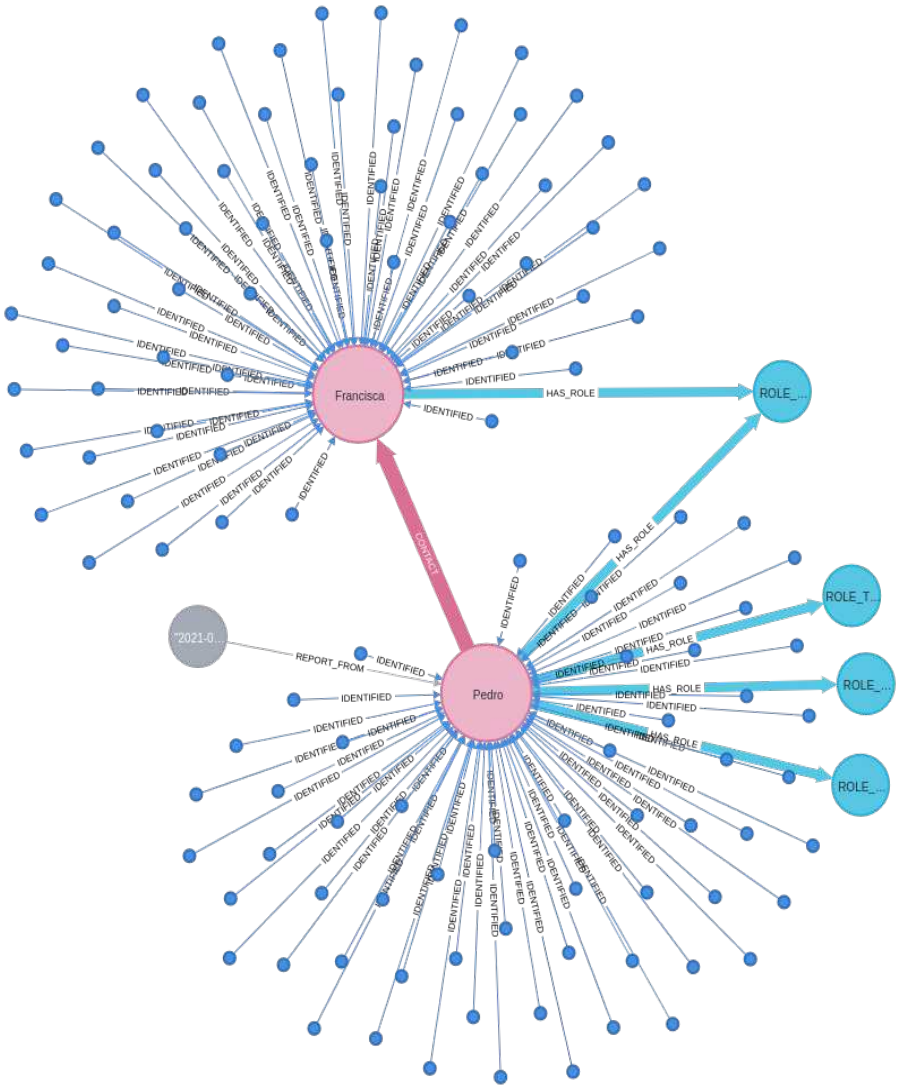

Introducimos en Postman los Token y probamos los métodos *<endpoint>/api/v1/ble/new, <endpoint>/api/v1/ble/report* y *<endpoint>/api/check*. También hay que probar que no son accesibles por usuarios con tokens incorrectos.


```
{
    "msg": "A user token is necesary for use this endpoint"
}
```

Ahora haremos una prueba del uso normal del API para comprobar que realiza los cambios correctos en la base de datos. Para ello seguimos los siguientes pasos:

- Registrar a Francisca y Pedro.
- Generar Códigos para ambos.
- Reportar a Pedro como positivo incluyendo un código de francisca en el mensaje.

El resultado en la base de datos es el siguiente:



Podemos apreciar como cada uno de los usuarios tiene 60 identificadores que han sido generados por el método *<endpoint>/api/v1/ble/new* . Cuando el usuario

reporta el contacto con un identificador efímero, el controlador se encarga de crear una relación *CONTACT* con el usuario que lo posee y crear un nuevo nodo Report con información relacionada.

4.2. Sprint 2

El Sprint 2 tomará dos semanas que serán del 5 de julio al 16 de julio. En estas dos semanas se llevará acabo el panel de control. Está dividido en varias secciones para cada uno de los usuarios que lo utilizan, los rastreadores, los sanitarios y el administrador. Además cuenta con otra sección con estadísticas del estado de la aplicación.

En los dos primeros días, es decir del día 5 al 6, se preparará el sistema de inicio de sesión. Se gestionarán los usuarios y se permitirá acceder a la web. Los días 7, 8 y 9, se llevará acabo el panel al que tendrán acceso los administradores, para poder crear nuevos usuarios de manera sencilla o borrar los ya existentes.

En la segunda semana, comenzaremos con el desarrollo de las funciones del personal sanitario permitiendo, por un lado, conocer si un usuario ha estado en riesgo y, por otra parte, dar códigos de validación; Esta tarea nos tomará del 12 al 13. Para terminar, del 14 al 17 estaremos trabajando con el panel del rastreador.

Se desarrollará con el framework Symfony con lo que se podrá reutilizar parte del código utilizado en el sprint 1 para el acceso a la base de datos, añadiendo aquellos métodos que sean necesarios.

4.2.1 Desarrollo

Comenzamos preparando el entorno de trabajo. Al igual que antes, creamos el esqueleto de la página web utilizando compose, utilizamos el paquete *symfony/website-skeleton.*
Utilizaremos el framework BootStrap para el frontend, esto nos facilitará la creación de plantillas pues la mayoría de elementos ya se encuentran integrados con un visionado uniforme. Preparamos la configuración para utilizar WebPack, esta tecnología agilizará el desarrollo con javascript y css pues se integra de manera automática en symfony.

Utilizaremos NPM para instalar bootstrap y gestionar dependencias externas. WebPack lo utilizaremos para compilar el código sass de bootstrap cada vez que hagamos un cambio.

Para el inicio de sesión trabajamos de forma parecida al sprint 1. Por un lado necesitamos la clase UserProvider para obtener los usuarios de la base de datos y

por otro el autenticador que permite comparar el usuario y contraseña. Esto son las bases del sistema de inicio sesión.

El segundo día trabajamos con la interfaz gráfica, creando un formulario para el inicio de sesión. Utilizamos la consola de symfony con el comando *"symfony make:auth".* Nos aparecerá una interfaz por consola que nos guiará en la creación del formulario, rellenamos la información que nos solicita y realizará los siguientes cambios en el proyecto:

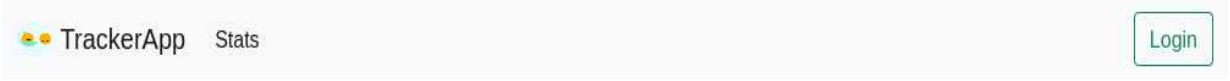
- Crea las rutas de /login y /logout hacia el controlador
- Crea una plantilla para mostrar el formulario de inicio de sesión.
- Genera una clase que extiende de *AbstractGuardAuthenticator* para comprobar a los usuarios
- Actualiza el archivo de configuración de seguridad

La generación automática de formularios no es perfecta por parte de symfony, tenemos que realizar algunos cambios dentro de las clases generadas para indicar que comprobaciones tiene que hacer para obtener el usuario y comprobar la contraseña.

Dedicamos los tres días siguientes a crear la estructura básica de la web y el área para crear y modificar usuarios.

La estructura básica de la web esta compuesta por una barra de navegación en la parte superior, el cuerpo del contenido a mostrar y un footer. Cada uno de estos componentes se ha creado independiente de los demás en una plantilla, para la vista final se unen unos con otros para formar la estructura básica.

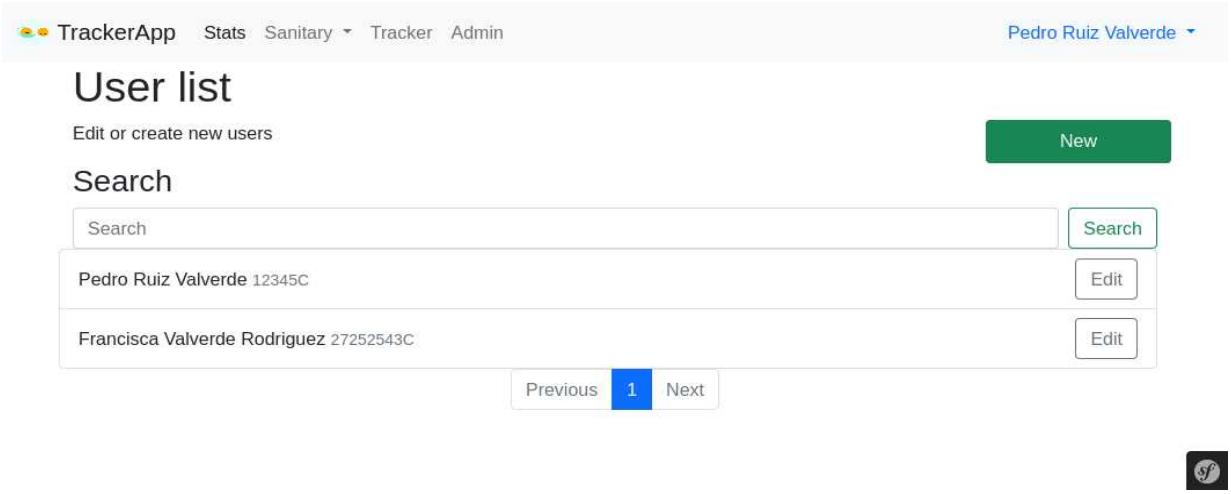
La barra de navegación tiene un menú con enlaces a las diferentes áreas de la web, dependiendo de los roles que tenga el usuario se mostrarán unas u otras.



En caso de entrar con un usuario con todos los roles, la barra se verá de la siguiente forma:

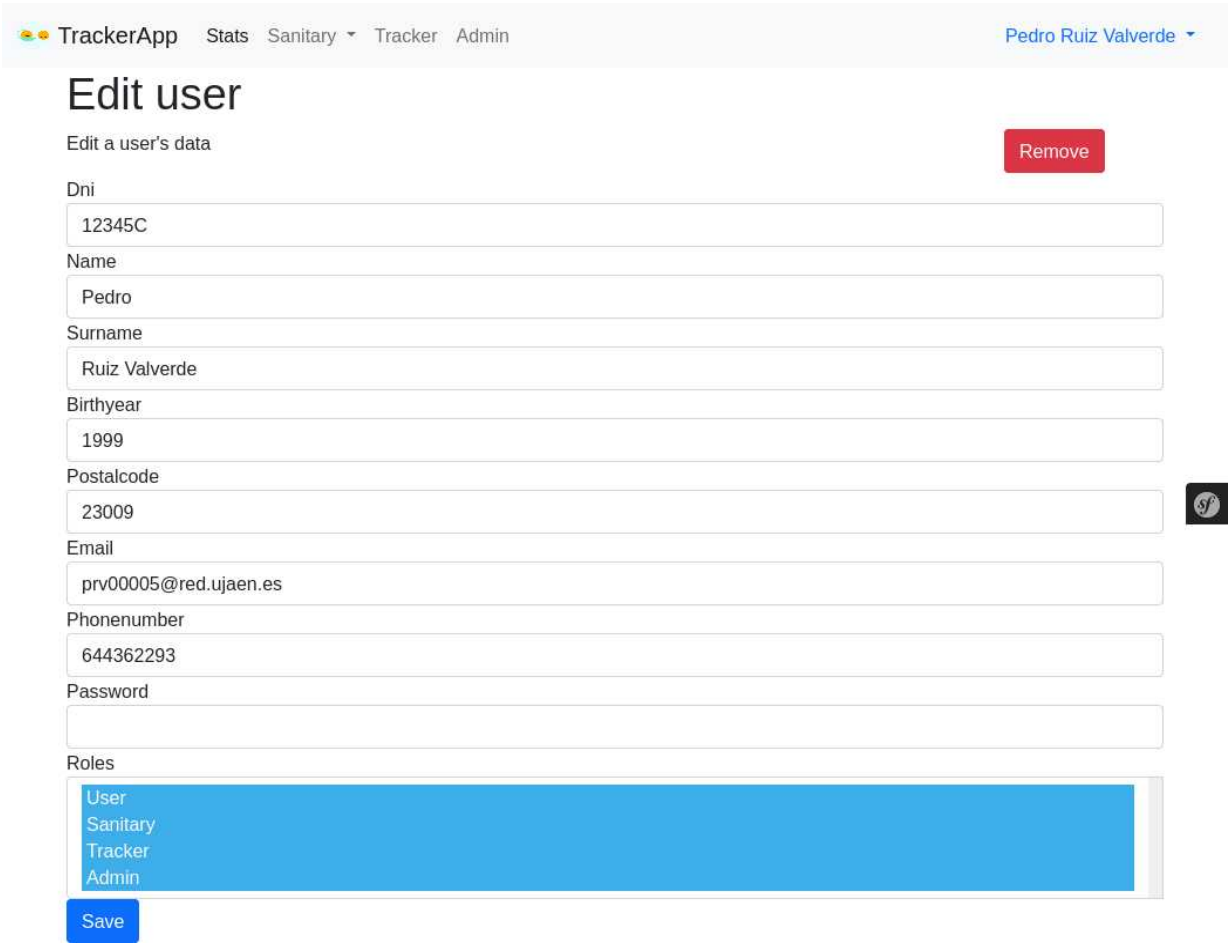
Los días 7, 8 y 9 de julio procedemos a crear el acceso para los administradores web. Este constará de una lista con un buscador donde se mostrarán a todos los usuarios. Se podrá añadir, eliminar y editar usuarios a través de botones.

Para la lista de usuarios se ha integrado un paginador junto a un buscador. Symfony integra un paginador bastante fácil de usar, pero no se ha podido utilizar por tener Neo4J.Esto ha supuesto algunos quebraderos de cabeza, con un poco de lógica se ha resuelto.



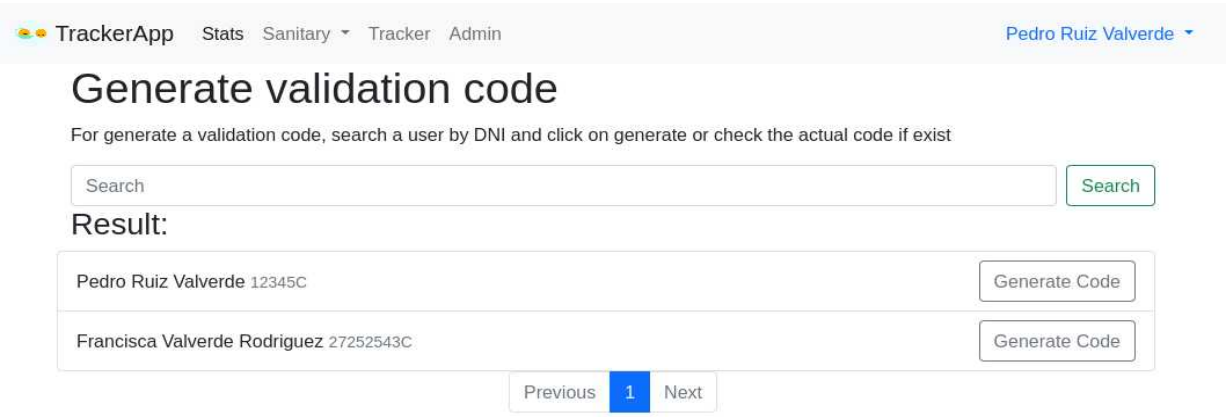
Las opciones de Nuevo y Editar son muy parecidas en cuanto a implementación. Ambas están compuestas por un formulario creado con la librería de symfony y que en caso de ser la opción de editar, esta relleno por defecto con los datos del usuario a modificar. En caso contrario está vacío y sin la opción de eliminar como es lógico.

Aquellos botones que pueden ser atacados mediante CSRF han sido dotados de protección mediante el sistema de seguridad de Symfony. La comprobación se realiza comparando un token temporal que es generado exclusivamente para cada usuario.



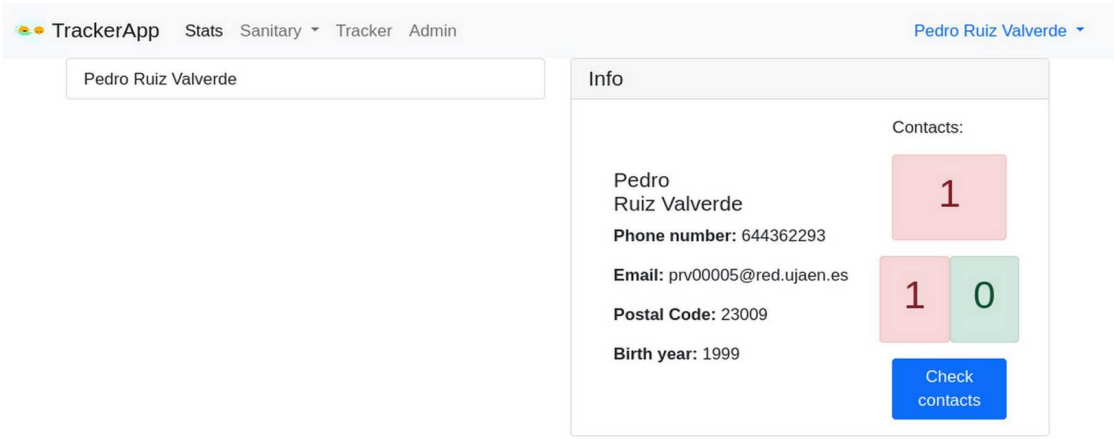
Del 12 al 13 trabajamos en el panel de los sanitarios. Esta parte requirió poco esfuerzo pues reutilizamos la lista de usuarios creada para la gestión de usuarios y tardamos un día menos del esperado. Tanto el área de validación como la que muestra los contactos positivos se controlan casi de igual manera, es por ello que la adaptación de una a otra fue trivial.

La generación del código de validación queda de la siguiente forma:



Para terminar el sprint nos ponemos con el apartado del rastreador. Esto nos tomará 3 días, desde el 14 hasta el 16.

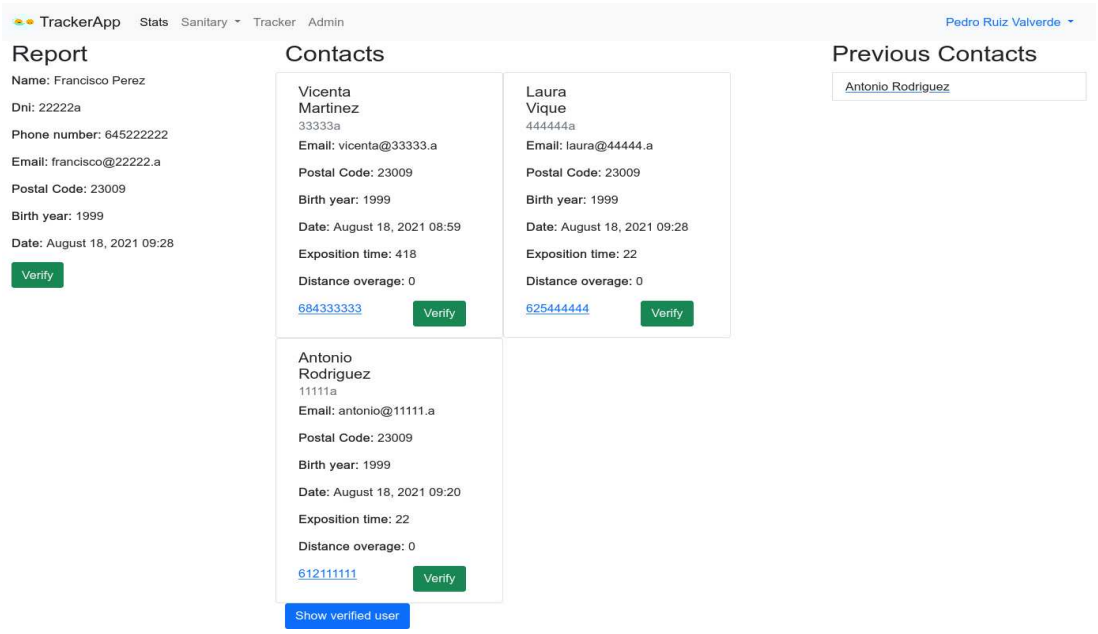
El primer día se realizó un boceto de las funciones y visualización que tendrá esta parte. Seguidamente se modelo la interfaz en twig para posteriormente integrarla en el controlador con datos aleatorios para comprobar el resultado final.



El segundo día y tercero, se comenzó a implementar las funcionalidades. Se añadió javascript para la carga asíncrona de las tarjetas de los usuarios que se muestran al lado derecho al pulsar sobre el nombre. Para ello se utilizó la librería de Stimulus que permite interaccionar con elementos del DOM a través de etiquetas.

Empezamos también con la interfaz gráfica para mostrar los reportes. Está formada por 3 columnas: la primera mostrará información sobre la persona contagiada, la segunda contiene los contactos relacionados y la última muestra contactos con personas que hayan tenido la enfermedad previamente. En cada uno de los contactos se puede pulsar un botón para marcarlo como verificado.

El rastreador podrá marcar como verificado un reporte para que desaparezca de la lista.



4.3. Sprint 3

Para el tercer sprint, comenzaremos con la aplicación android. Este sprint durará 3 semanas del 26 de julio al 13 de agosto.

La aplicación Android utiliza Android SDK 21 con la que abarcamos el 94.1% de los dispositivos Android que están funcionando a junio de 2021. Es la mínima versión que podremos utilizar debido a la limitación de las librerías utilizadas.

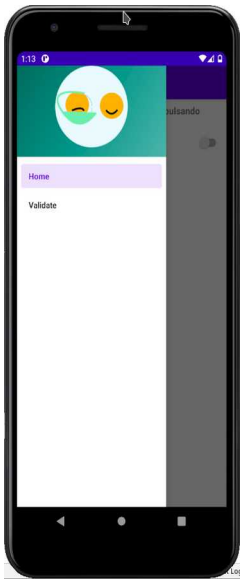
La aplicación será muy básica. Tendrá una interfaz sencilla y completa para poder activar, desactivar, validar positivo y ver algunas estadísticas sobre el uso.

Todos los beacons recibidos se quedarán guardados en la memoria interna del dispositivo durante 14 días. Si se diera el caso, todos los beacons recibidos durante los últimos 14 días serán enviados al servidor.

Para poder validar el positivo, el equipo sanitario debe proporcionar un código de validación.

4.3.1 Desarrollo

Preparamos el proyecto utilizando una de las plantillas que nos muestra Android Studio al crear el proyecto. Esta contiene un menú lateral con algunas opciones, cada opción es un fragmento de la actividad principal.



Modificamos las plantillas para añadir ciertos elementos con los que interactuaremos. Vamos a encontrar dos opciones en el menú lateral, la ventana de home desde donde podremos activar o desactivar el tracker y la opción de validar para reportar un caso positivo de Covid-19. Por ahora no vamos a añadir funcionalidad pero sí los vamos a dejar visibles en la interfaz para posteriormente añadirle el control.

4.3.1.1 Almacenamiento

Para almacenamiento, vamos a utilizar la biblioteca Room. Esta biblioteca trabaja con Sqlite por debajo. No tenemos que inicializar, ni crear tablas, Room se encarga de lo necesario para hacerla funcionar.

La gestión de datos en Room está dividida en tres clases principales. Las entidades, que vienen a ser una abstracción de los atributos que contiene una tabla. Los objetos de acceso a datos, llamados DAO (Data Access Objects), estas deben de ser inicializadas una sola vez debido al alto consumo de recursos que tienen[12]. Es

por ello que utilizamos el patrón de diseño único en ingles "*singleton design pattern*", para inicializar una única vez el objeto de base de datos por aplicación.

Guardaremos en la base de datos dos entidades Token y Beacon. Token será utilizada para guardar los códigos efímeros mientras que Beacon se utilizará para guardar los beacons obtenidos junto a algunos datos relacionados con ellos. Para más información véase Estructura de la base de datos.

Para almacenar el token del api utilizaremos el objeto SharedPreferences.

4.3.1.2 Permisos

Android limita el acceso a partes del sistema por medio de permisos. Los divide en dos tipos de permisos, los críticos y los no críticos. Aquellos que son críticos suele comprometer de manera alta la privacidad del usuario, como puede ser acceso a la memoria, a la cámara, micrófono, etc. Estos permisos necesitan confirmación de manera explícita por el usuario. Los permisos no críticos no necesitan que sean confirmados de manera explícita por el usuario, pero al igual que los críticos deben de estar incluidos en el manifest.

Nuestra aplicación necesita acceso al bluetooth para compartir los códigos efímeros y a internet para poder registrarse, enviar reportes… El acceso a internet es un permiso no crítico, así que, añadiéndolo al manifest será suficiente (*android.permission.INTERNET*). Para el acceso a bluetooth es necesario tanto ponerlo en el manifest como realizar una solicitud de aceptación al usuario en tiempo de ejecución (*android.permission.ACCESS_FINE_LOCATION).*

4.3.1.3 Acceso a la API RestFul

Para contactar con la API desarrollaremos una librería muy simple que contactará con el servidor y posteriormente devolverá los datos. Android implementa muchos mecanismos para forzar al desarrollador a seguir las mejores pautas, uno de estos mecanismos es que impide que una app utilice el protocolo http sin SSL. Para poder hacer las pruebas sin SSL es necesario indicarlo en el manifest.

Finalmente la librería constará de los siguientes métodos para realizar las consultas:

- registerToken(String) : String
- getContacts() : Contacts
- sendReport(String, List<Beacon>) : Int
- getTokens() : List

- RegistryApi(String)

4.3.1.4 Registro de la App

El registro del usuario se realizará al abrir la aplicación por primera vez. Se abrirá la activity *RegisterActivity* en la que se deberá introducir el código de registro. En caso de error se mostrará un mensaje encima del botón de enviar, si el registro se hace correctamente se cerrará la actividad y se guardará el api token en las preferencias de la aplicación utilizando *SharedPreferences.*

4.3.1.5 Obtener los Beacons próximos

Para obtener los beacons se utilizará la librería android beacon library. Utilizaremos el beacon Eddystone EID. Esta librería se inicializa indicando una serie de filtros para que excluya aquellos paquetes que no los pase.

Debemos de indicarle que clase será nuestro consumidor, para ello extendemos la Actividad principal con BeaconConsumer y RangeNotifier. Se enlaza con bind para obtener periódicamente los beacons cercanos a través del callback *didRangeBeaconsInRegion()*.

Antes de almacenar los beacons en la base de datos son guardados en ram hasta que se cierra la aplicación. Este es el comportamiento elegido porque actualmente la aplicación solo funcionará cuando esté en primer plano. En caso de que se implementara como servicio es necesario que guardara estos datos cada 15 minutos que es el mismo periodo de tiempo en el que rotará el identificador efímero. De esta manera si el móvil se apagara sin previo aviso se tendrán guardados la mayoría de beacons recibidos.

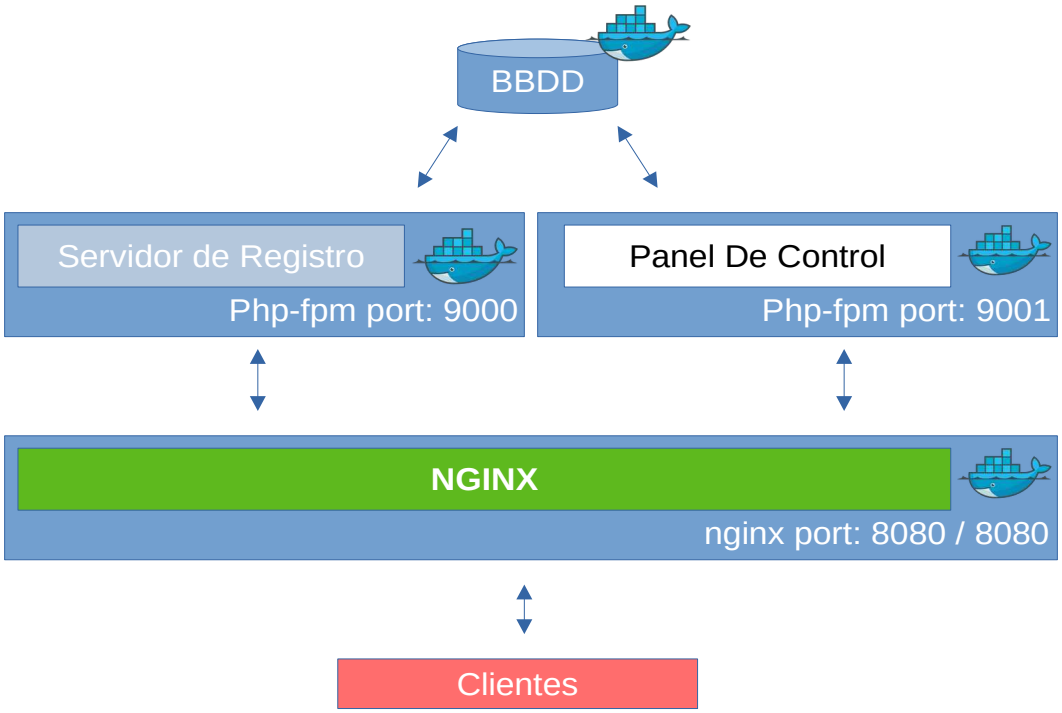
4.3.1.6 Reportar un caso positivo de Covid-19

Para reportar un caso positivo se ha creado una interfaz muy sencilla con un campo para introducir texto y un botón de enviar. En este campo se introduce el código de validación que nos otorga el equipo sanitario, si es correcto se envía un mensaje al servidor con todos los contactos que se han ido guardando en los últimos 14 días. Esto permite al rastreador revisar con que usuarios has estado cerca y poder contactar con ellos si fuera necesario.

5. Despliegue

Haremos un despliegue de la aplicación web y de la Api Restful utilizando Docker. Con esto facilitamos el lanzamiento de tantos contenedores como sean necesarios para suplir la demanda a la api.

Utilizaremos DigitalOcean (proveedor de servicios en la nube) para almacenar las imágenes y para lanzar los contenedores.



Se utilizarán dos redes aisladas para la comunicación interna de los contenedores. Por un lado tendremos una red para la comunicación del panel de control y el servidor de registro con la base de datos para asegurarnos de que Nginx no pueda acceder directamente a la base de datos. Por el otro lado crearemos una red en la que se comunicará Nginx con el servidor de registro y el panel de control.

Finalmente se podrá acceder al contenedor de nginx desde internet a través de los puertos 8081 y 8080 para el panel de control y el servidor de registro respectivamente.

Como el despliegue lo realizaremos sobre una sola máquina, utilizaremos las herramientas que nos proporciona Docker.

Para la primera configuración de la base de datos podemos usar dos procedimientos: Introducir los datos manualmente o crear un controlador en una de las dos aplicaciones que en caso de que este vacía la rellene con un usuario administrador. En nuestro caso lo haremos de la segunda manera para facilitarnos el despliegue.

5.1. Imágenes

El primer paso que realizaremos será la creación de las dos imágenes en donde estarán nuestras aplicaciones. Ambas imágenes serán muy parecidas en cuanto a configuración. Comenzaremos en el de la api el cual nos servirá posteriormente para hacer el del panel de control.

En el segundo paso prepararemos un contenedor con la base de datos Neo4j. Muy posiblemente, no necesitemos crear uno nuevo y podremos utilizar el que nos ofrece la comunidad de Neo4j. Para nginx, utilizaremos el que nos proporciona el propio software incluyendo la configuración necesaria para lanzar la api y la web.

Las imágenes las subiremos al registro que dispone DigitalOcean para posteriormente descargarlas en el servidor.

5.1.1 Servidor de Registro y Panel de Control

Para las imágenes que contienen nuestras dos apps con Symfony utilizaremos la imagen de php:fpm7.4 como base.

Será necesario instalar algunas dependencias con el comando apt-get (composer, wget) y las librerías de php (sockets y zip).

Copiaremos las carpetas de nuestra aplicación (bin/, config/, public/, src/ y ficheros necesarios para composer).

Para terminar la configuración reemplazaremos los ficheros de configuración de php-fpm para ajustarlo a nuestro diseño.

Esta configuración se ubicará dentro de la carpeta Docker de cada proyecto. El fichero Dockerfile tendrá las instrucciones para realizar las acciones antes nombradas.

Para más información véase Panel de control y Api.

5.1.2 Nginx

Para la imagen de Nginx utilizamos nginx:latest como base y copiamos la configuración de las dos webs a la carpeta *"/etc/nginx/conf.d/"* .

Para más información véase Nginx.

5.2. Docker compose

Para facilitarnos el despliegue utilizaremos docker compose. Con compose se utiliza un fichero Yaml en donde se configura los servicios que se quieren lanzar. Se pueden definir directamente las redes y las opciones con las que se debe de lanzar cada contenedor, posteriormente se lanza ejecutando *'docker-compose up'.* El fichero resultante es el siguiente:

Definimos algunas variables para que el software tenga acceso a la base de datos.

También definimos las dos redes que se deben de utilizar para la comunicación interna de los contenedores *frontend* y *backend*.

Para nginx abrimos dos puertos: el del panel de control y el de la api.

```yaml
version: "3.9"
services:
  panel:
    image: tfg/panel:latest
    networks:
      - backend
      - frontend
    volumes:
      - panel:/var/www/panel
    environment:
      - NEO4J_USER=neo4j
      - NEO4J_PASSWORD=jf8Dhc7dmkEWfdf9f7DS
      - NEO4J_HOST=neo4j

  api:
    image: tfg/api:latest
    networks:
      - backend
      - frontend
    volumes:
      - api:/var/www/api/
    environment:
      - NEO4J_USER=neo4j
      - NEO4J_PASSWORD=jf8Dhc7dmkEWfdf9f7DS
      - NEO4J_HOST=neo4j

  neo4j:
    image: neo4j
    networks:
      - backend
    environment:
      - NEO4J_AUTH=neo4j/jf8Dhc7dmkEWfdf9f7DS

  nginx:
    image: tfg/nginx:latest
    ports:
      - 8081:8081/tcp
      - 8080:8080/tcp
    networks:
      - frontend
    volumes:
      - api:/var/www/api/
      - panel:/var/www/panel

volumes:
  api:
  panel:

networks:
  backend: {}
  frontend: {}
```

5.3. Despliegue en local

Antes de realizar el despliegue en la nube vamos a probar en el ordenador de desarrollo. Necesitamos construir las imágenes de la api, del panel de control y de nginx, para ello nos situamos en las carpetas de los dos primeros proyectos y ejecutamos el siguiente comando:

docker build -f docker/Dockerfile -t tfg/<project_name> .

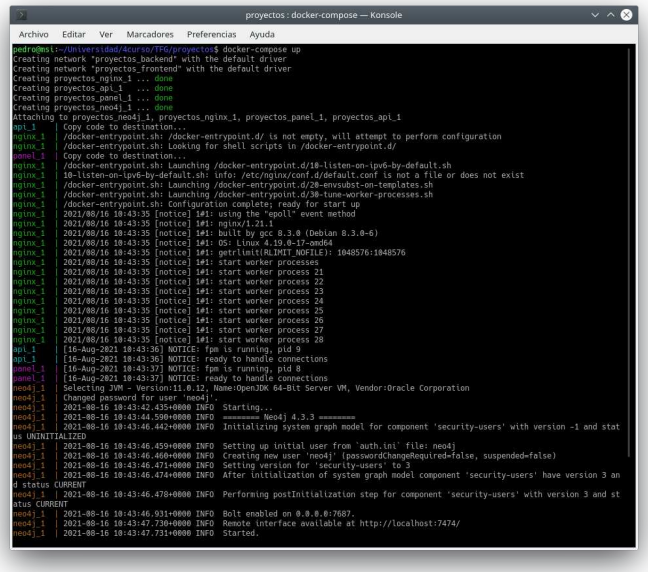
Este comando se encarga de localizar el fichero Dockerfile dentro de la carpeta docker y utiliza como contexto la carpeta actual. En poco tiempo tendremos la imagen construida.

Para nginx no hace falta buscar el dockerfile en otra carpeta y se debe de ejecutar el comando sin el argumento "-f *docker/Dockerfile* ".

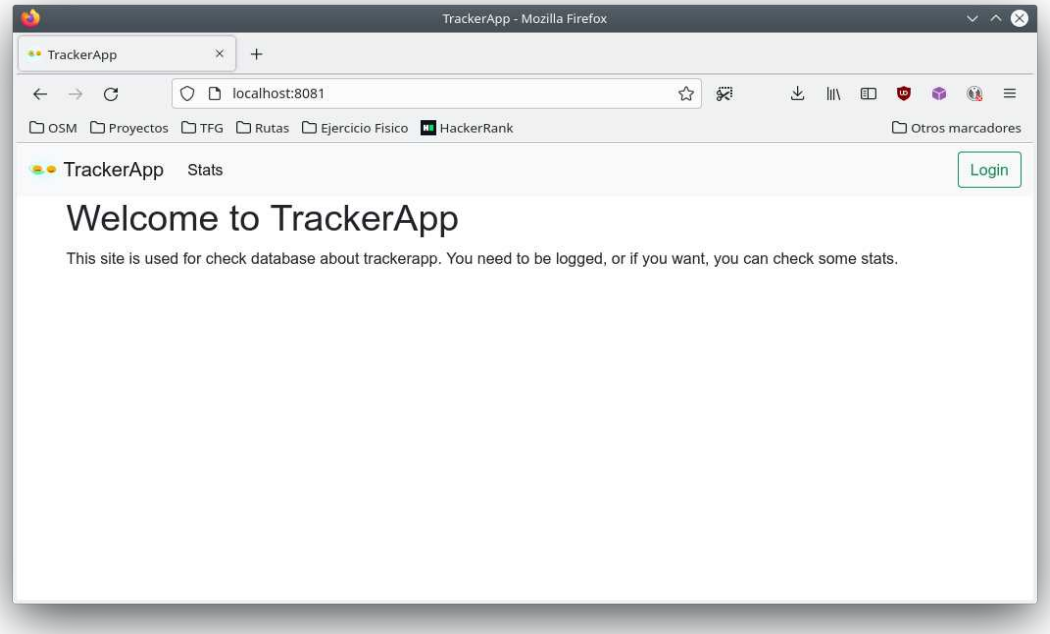
```
pedro@msi:~/Universidad/4curso/TFG/proyectos$ docker images
REPOSITORY                          TAG       IMAGE ID       CREATED          SIZE
tfg/panel                           latest    cef0a3e61e20   39 minutes ago   483MB
tfg/api                             latest    c742f6e17dde   42 minutes ago   446MB
tfg/nginx                           latest    c6c9138114c8   46 minutes ago   133MB
```

Con las imágenes construidas y docker-compose podemos hacer la primera prueba.

Lanzamos los servicios con *"docker-compose up"* y esperamos a obtener los logs de la inicialización.



El lanzamiento es un éxito. Intentamos acceder a la web con la dirección localhost:8081.



Ahora que ya hemos probado el funcionamiento correcto de la aplicación podemos continuar con el despliegue en un servidor real.

5.4. Despliegue en DigitalOcean

Para realizar el despliegue en DigitalOcean tendremos que seguir los pasos hechos en el despliegue en local con la excepción de que las imágenes ya estarán creadas. Necesitamos subir las imágenes a algún servicio externo para que sean accesibles desde internet.

La interfaz de DigitalOcean es bastante sencilla e intuitiva, es por ello que no voy a especificar el procedimiento para contratar el vps y el registro de contenedores para docker. El sistema operativo del vps es Debian 10. Para el acceso al vps utilizaremos openssh con una clave asimétrica. Para el acceso al registro de contenedores utilizaremos la consola doctl que genera automáticamente los credenciales y los registra en docker.

5.4.1 Subida de imágenes al registro

Para subir las imágenes necesitamos variar el nombre añadiendo la URL del servidor donde serán guardadas. Utilizamos el siguiente comando:

docker image tag tfg/<project_name> registry.digitalocean.com/tfg/<project_name>

Con el nombre cambiado y con el servicio de docker loggueado en el registro procedemos a subir las imágenes con

docker push registry.digitalocean.com/tfg/<project_name>

5.4.2 Instalación de docker en el Vps

Accedemos al servidor mediante la consola con el software openssh. Procedemos a instalar docker mediante el script que ofrecen en la web get.docker.com, posteriormente instalamos docker-compose.

Docker:

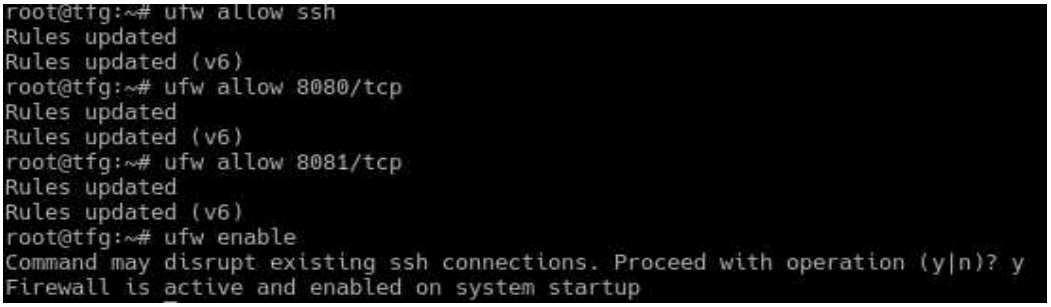
curl -sSL get.docker.com | sh

Docker compose:

sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

5.4.3 Configuración del firewall

Vamos a utilizar **ufw** acrónimo de **Uncomplicated Firewall** para configurar el firewall. Permitimos el acceso desde el puerto 22/tcp, 8080/tcp y 8081/tcp y activamos el firewall. Por defecto se bloquea cualquier conexión entrante menos las



establecidas.

5.4.4 Copiar fichero docker-compose.yml

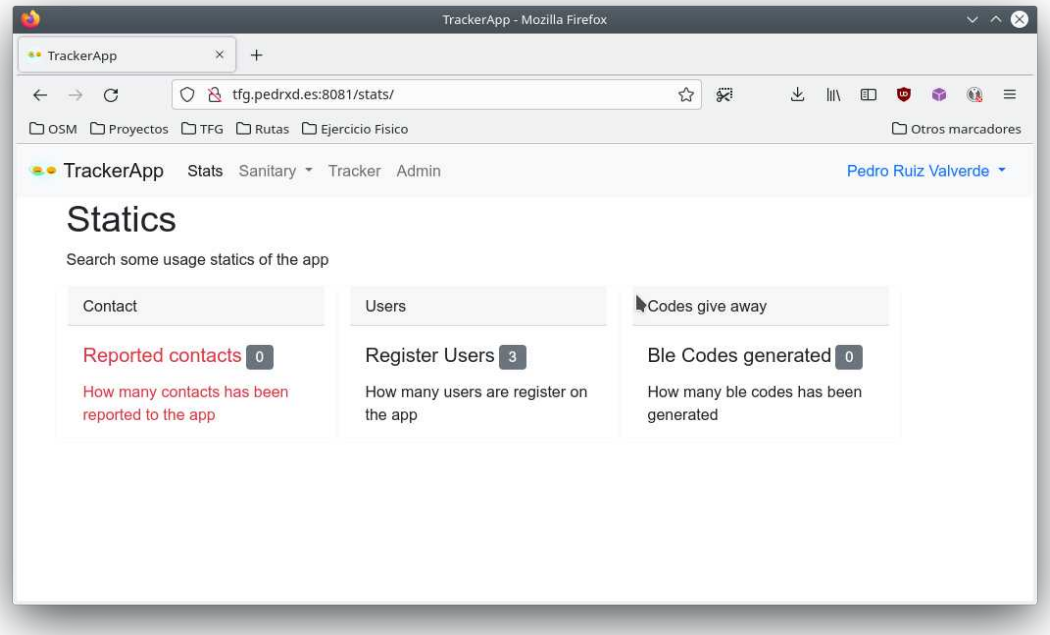
Para copiar el fichero podemos seguir dos métodos: Enviando el fichero a través de scp o copiando y pegando el contenido a través de la consola. Por facilidad vamos a utilizar el segundo método copiando y pegando el contenido.

5.4.5 Lanzar los servicios

Lanzamos los servicios declarados en el docker compose. Para ello ejecutamos "*docker-compose up"* y esperamos a que se descarguen y ejecuten. Obtenemos una salida por la terminal igual a la que hicimos en local.

Ejecutamos el método que realiza la configuración inicial de la base de datos entrando en la dirección <endpoint>/setup desde el navegador.

Comprobamos el correcto funcionamiento de la web iniciando sesión:


TrackerApp - Mozilla Firefox
TrackerApp
tfg.pedrxd.es:8081/stats/
OSM
Proyectos
TFG
Rutas
Ejercicio Fisico
Otros marcadores
TrackerApp
Stats
Sanitary
Tracker
Admin
Pedro Ruiz Valverde
Statics
Search some usage statics of the app
Contact
Reported contacts 0
How many contacts has been reported to the app
Users
Register Users 3
How many users are register on the app
Codes give away
Ble Codes generated 0
How many ble codes has been generated

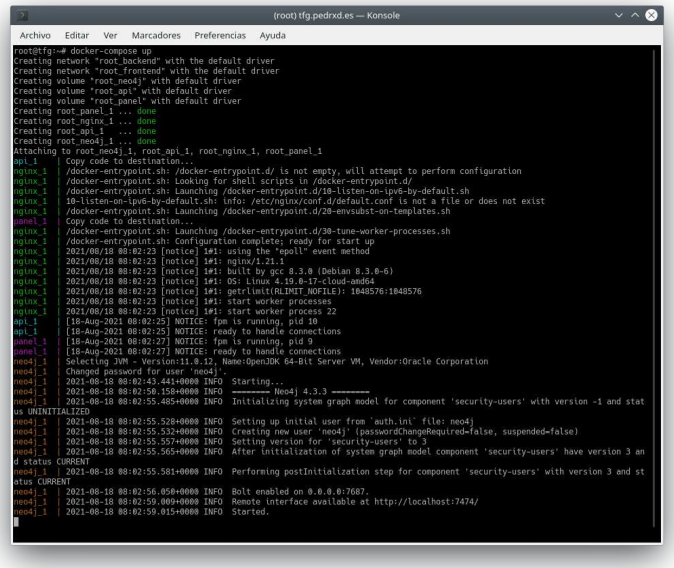
6. Prueba de funcionamiento

Vamos a realizar algunas pruebas de funcionamiento del sistema. Para ello seguiremos los siguientes pasos:

1. Lanzar docker-compose.
2. Ejecutar el método /setup.
3. Iniciar sesión.
4. Crear 4 usuarios.
5. Registrar 4 dispositivos con la aplicación móvil.
6. Colocar los dispositivos 1, 2 y 3 cerca para que registren un contacto.
7. Generar un código de validación e introducirla en el dispositivo 1.
8. Repetimos pasos 7 y 8 para el dispositivo 2 y 4.
9. Comprobar la vista de tracker
10. Comprobar la base de datos.

6.1. Lanzar docker-compose

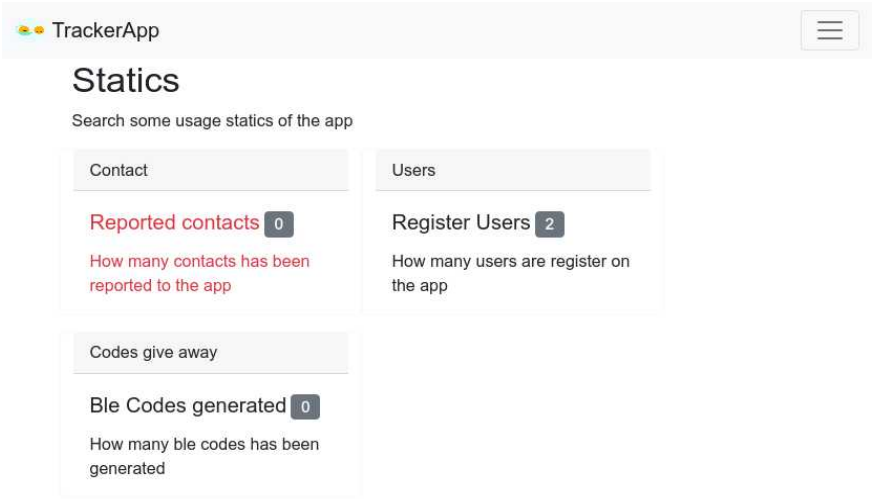
Como se ha mostrado en la sección de despliegue. Ejecutamos el comando *“docker-compose up”* y esperamos a que todos los servicios se ejecuten.



6.2. Ejecutar el método Setup

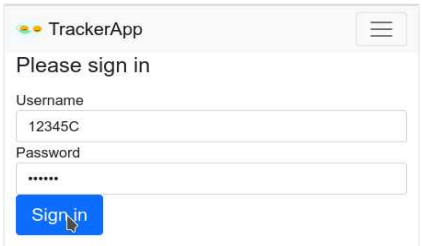
Es necesario rellenar la base de datos con los datos iniciales. Como comentamos anteriormente, se escogió crear un controlador el cual crearía los roles y el usuario admin si la base de datos está vacía. Para lanzarlo necesitamos acceder a la dirección <endpoint>/setup de la web.

Si todo fue bien, se redirigirá a la página raíz. Comprobamos en la página de stats los usuarios registrados:

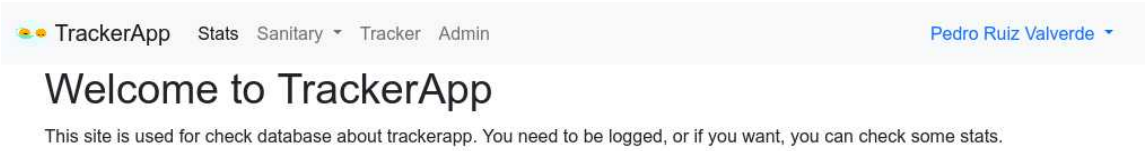


6.3. Iniciar sesión

Iniciamos sesión con el usuario administrador que se ha configurado en el paso anterior. Debemos de cambiarle la contraseña a una mas segura debido a que todas las instancias inician con la misma contraseña por defecto.

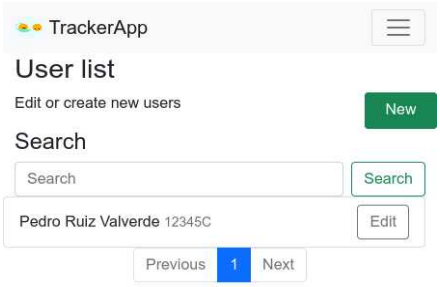


Si nos hemos loggueado correctamente deberá de aparecer el nombre del usuario administrador a la derecha de la barra de navegación junto a las opciones para navegar a cada una de las secciones.



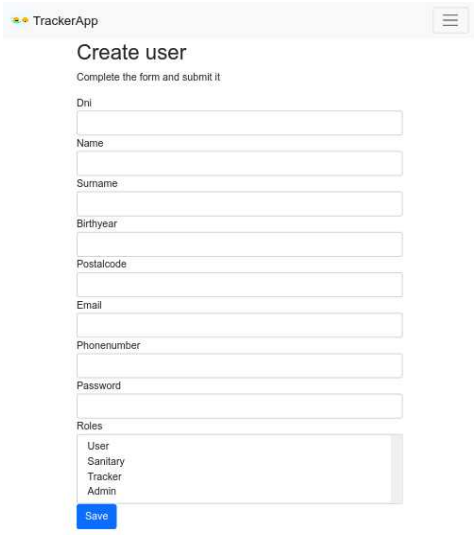
6.4. Crear 4 usuarios nuevos

Pulsamos en el botón “Admin” de la barra de navegación que nos llevará al apartado de la gestión de usuarios.



En esta pantalla podemos crear nuevos usuarios pulsando sobre el botón verde “New”. En caso de querer editar un usuario se debe de pulsar en el botón “Edit” que aparece al lado de cada usuario.

Pulsamos en “New” que nos llevará a un formulario en donde completaremos los datos del nuevo usuario.

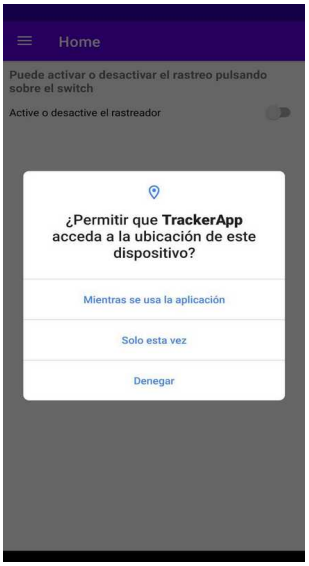


Al finalizar pulsamos sobre el botón de “Save”. Esto guardará el nuevo usuario en la base de datos y te redirigirá a una nueva página en donde aparecerá el código de registro para que el usuario lo introduzca en la aplicación móvil.

Realizamos este proceso para los cuatro usuarios que queremos crear. La lista muestra los usuarios al finalizar el proceso:

6.5. Registrar 4 dispositivos móviles

Instalamos la aplicación en cuatro dispositivos. Introducimos el código de los usuarios en cada uno de los dispositivos y aceptamos los permisos.



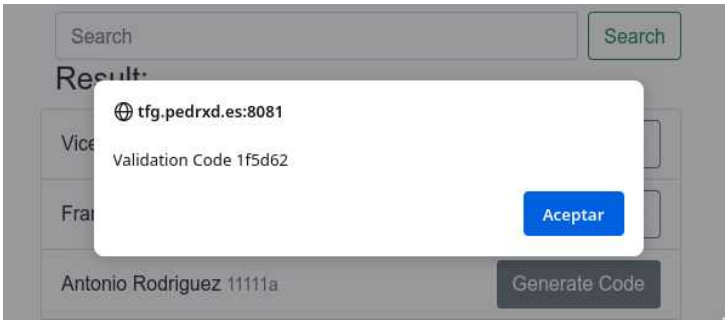
6.6. Colocar cerca dispositivos 1, 2 y 3

Colocamos los dispositivos cerca y activamos el rastreador. Tras medio minuto aproximadamente alejamos los dispositivos. El dispositivo 1 tendrá un contacto con el 2 y 3.

6.7. Generar código de validación

Para reportar el caso positivo en el dispositivo 1, el sanitario entrará en el apartado “Sanitary” → “Validation code” de la página web. Aparecerá una lista con todos los

usuarios, podemos filtrar por DNI o nombre. Se pulsa en el botón “Generate Code” para generar el código que se mostrará en un PopUp.



Se introduce el código en la app móvil para realizar el reporte.

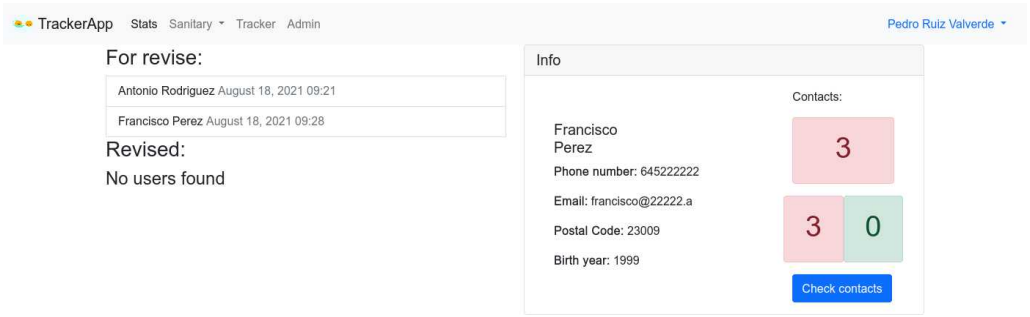
6.8. Repetimos el proceso para el dispositivo 2 y 4

Repetimos el proceso para los dispositivos 2 y 4. El dispositivo 2 tendrá contactos con 1, 3 y 4.

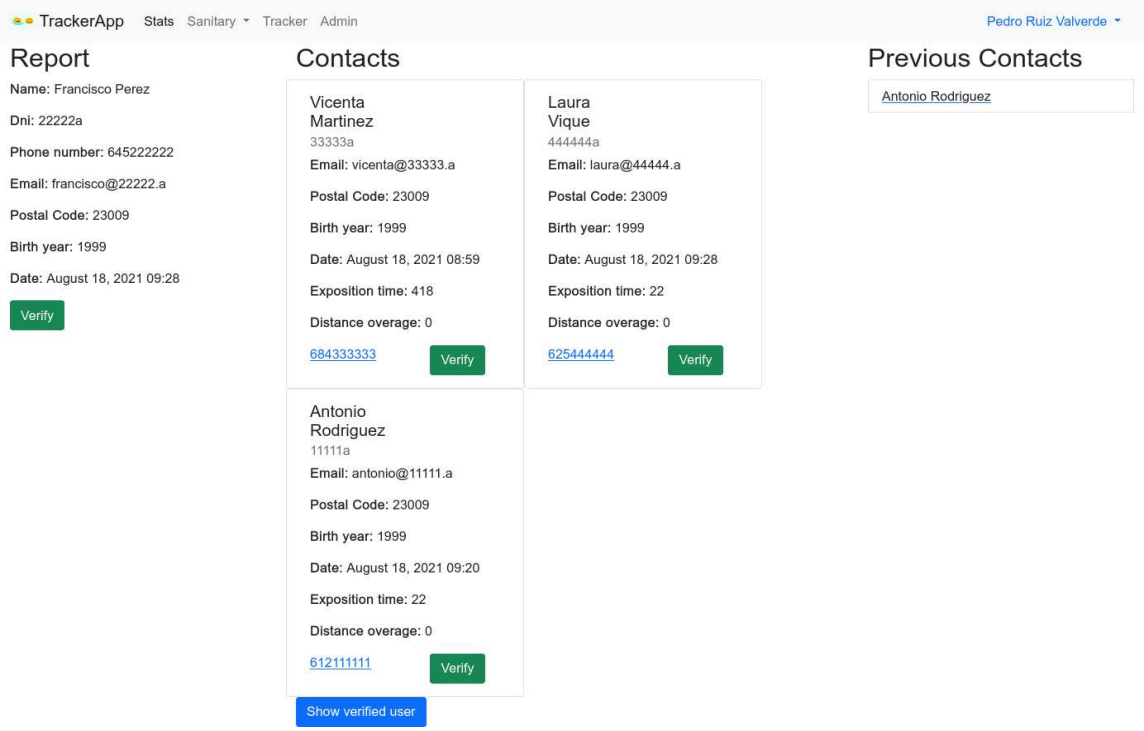
6.9. Comprobar la vista del Tracker

Con los contactos reportados al servidor es momento de comprobar como se visualizan estos contactos. Para ello el rastreador debe de entrar en la opción “Tracker” del panel de control donde se encontrará con la siguiente web.

Podemos hacer click en los nombres para mostrar una preview del número de contactos que ha tenido el usuario, aparecerá en verde los usuarios que hayan sido revisados y en rojo aquellos que no.



Para comprobar de manera mas detallada el reporte se pulsará sobre el botón de “Check Contacts”.



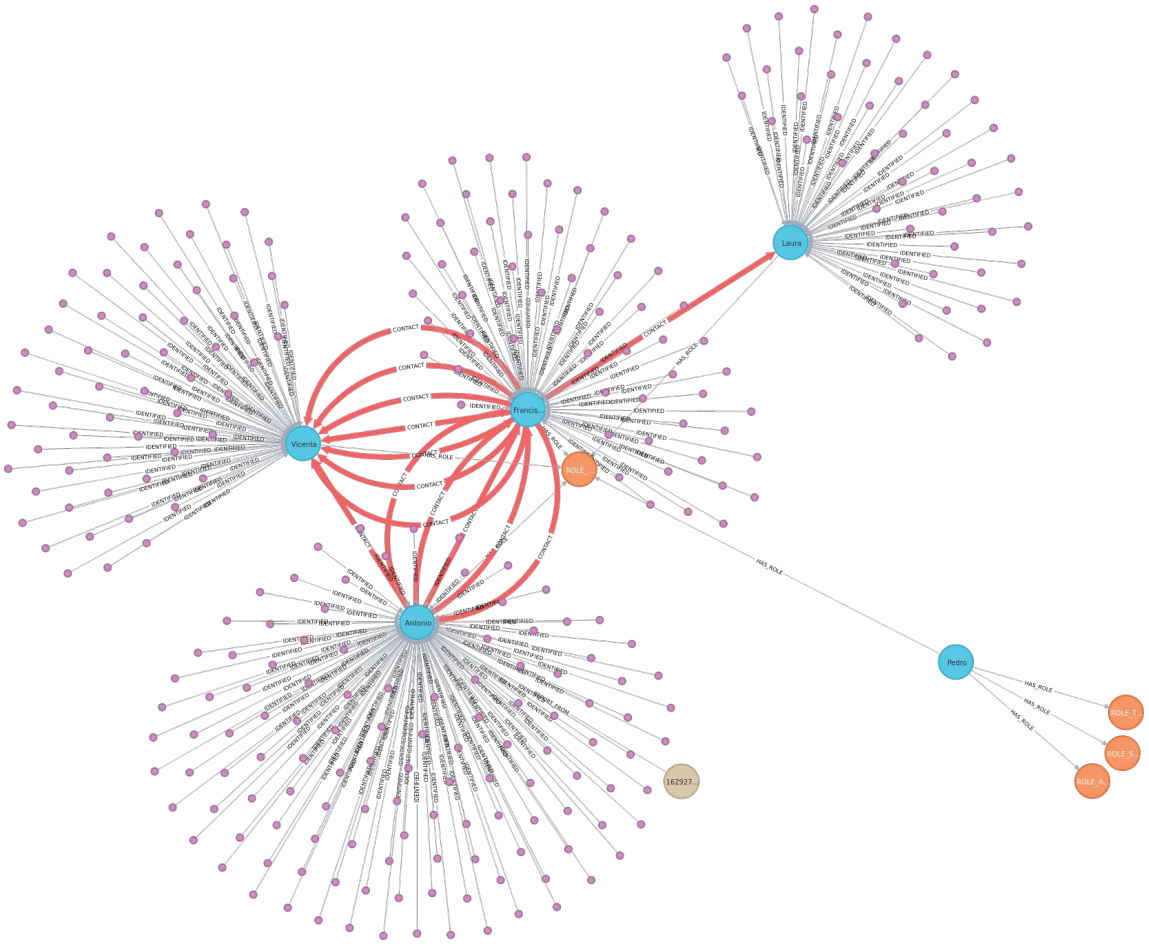
En esta pestaña se puede revisar datos relacionados con las exposiciones. Esta dividida en tres columnas: la primera columna llamada “Report” contiene información sobre el usuario que reportó, la segunda columna contiene información de los contactos cercanos, mientras que la tercera muestra aquellos contactos que fueron positivos en los últimos 14 días.

En la última columna se muestran aquellos usuarios que pudieron contagiar al usuario del actual reporte.

Si pulsamos sobre “Verify” en alguno de los contactos este será escondido en la parte de abajo de la web. Será necesario pulsar sobre el botón “Show verified User” para mostrarlo de nuevo.

6.10. Comprobar la base de datos

Accedemos a la base de datos temporalmente permitiendo el acceso a los puertos 7474/tcp y 7687/tcp. El acceso lo realizamos desde el navegador utilizando el puerto 7474, también se puede utilizar la aplicación desktop de Neo4j.



Los nodos están representados por círculos de diferentes colores: El azul son los usuarios, el naranja los roles, los morados los códigos efímeros y los grises los reportes.

El renderizado de nodos está limitado en esta versión de Neo4J por tanto algunos nodos no son visibles.

Las relaciones están representadas por líneas, la única que he diferenciado son los contactos que son de color rojo.

7. Dificultades encontradas

En el desarrollo del Trabajo de Fin de Grado he encontrado algunas dificultades que han supuesto un esfuerzo extra en la finalización de este. Uno de los mayores problemas ha sido en el desarrollo de la aplicación Android.

Anteriormente, no había hecho ningún tipo de desarrollo para Android. Con el entusiasmo elegí utilizar Kotlin como lenguaje principal que es el que recomienda Google. Tuve que aprender como se desarrollaba en Android, los conceptos básicos, como se gestionaban los permisos, como se lanzaban los procesos en segundo plano etc. Esto se mezcló con que Kotlin era nuevo también para mí, por tanto tenía que aprender como funcionaba la plataforma a la vez que entendía el lenguaje en el que se debía de desarrollar.

Después de estar casi dos semanas preparando la interfaz y aprendiendo, tuve que tomar una difícil decisión, preferí dejar Kotlin a favor de Java, un lenguaje que aprendí en la carrera, gracias a esto pude finalizar el proyecto en el tiempo establecido.

El resto de tecnologías no supusieron un problema, en un tiempo corto fui capaz de entenderlas y utilizarlas.

El lenguaje Cypher muy parecido a SQL, pareció ser complicado en un primer momento pero una vez aprendido resultó más agradable que el propio SQL. La manera de relacionar un nodo con otro, de obtener resultados, de guardar propiedades en los objetos era muy intuitiva y parecida al lenguaje natural.

Symfony, aunque en un principio fue complicado conectar la base de datos, después fue trivial realizar las consultas y obtener los resultados. Otra de las partes más difíciles fue integrar el sistema de autentificación con Neo4J, aunque gracias a las guias de Symfony y tras muchas pruebas funcionó correctamente.

8. Mejoras propuestas

Hay varias áreas que pueden ser mejoradas y necesarias si se le quiere dar un uso real.

Para la app móvil, se debería de implementar el rastreador en un servicio en primer plano para que Android no detenga la ejecución. Aunque parezca contradictorio, un servicio en primer plano se ejecuta aunque el usuario deje de interactuar con la aplicación y muestra una notificación de que está activo. En cambio, un servicio en segundo plano se mantiene durante un tiempo y Android no asegura que se mantenga en ejecución [13].

En el panel de control se puede incluir una heurística para determinar el grado de posible contagio. La heurística podría utilizar información como la velocidad del dispositivo, los contactos, si el dispositivo está en un interior o exterior, etc. para que indique de manera precisa si alguien ha sido contagiado o no priorizando mostrar a los rastreadores aquellos con un valor más alto.

Además, se podría incorporar un grafo con los contactos en vez de mostrarlos a través de una lista. Esta representación sería mas intuitiva y el rastreador podría de un vistazo conocer el estado de la propagación.

También el apartado visual se puede mejorar añadiendo colores y distribuyendo el contenido de una forma más confortable. Esta opción es fácil de realizar pues solo necesitaría modificar las plantillas de twing, el código se mantendría igual.

Para finalizar, comprobaría el control de errores para que se gestionaran todos adecuadamente y se guardara en un registro para la posterior revisión. Aunque actualmente gestiona los errores y los muestra por pantalla, no realiza ningún tipo de registro persistente.

9. Conclusión

Ha sido un proyecto bastante completo en el que se han utilizado muchísimas tecnologías nuevas que han supuesto un reto. Esto me ha ayudado a comprender la importancia de la documentación pues una buena documentación se convertía en un entendimiento rápido de esta tecnología mientras que en otros casos tenia que aprender mediante el ensayo y error.

Cabe destacar que lo aprendido en este proyecto me será de gran ayuda en un futuro. Además, me ha dado a entender el valor de la experiencia, si volviera a realizar este mismo proyecto me tomaría bastante menos tiempo. Esto lo pude comprobar al realizar el cambio de Kotlin a Java, que me tomo menos de un día en cambiar el lenguaje por que ya entendía los conceptos de Android.

Por último, he aprendido también sobre la importancia de la planificación. Es necesario hacer un buen análisis inicial tomando un tiempo en el cual se debe de reflexionar sobre cada apartado, solo de esta manera se puede derivar a un desarrollo fluido ahorrando muchas horas de programación.

10. Apéndices

10.1. Especificación de la API RestFul

La comunicación entre la app móvil y el servidor de registro se hará con el protocolo de transporte HTTP y se utilizará el formato JSON.

Todos contienen un atributo llamado “msg” con el estado de la solicitud en idioma entendible por un humano.

10.1.1 Registro de App

Solicitud GET: {endpoint}/api/v1/register/{*register_code*}

Respuesta:

```
200 OK
{
  “api_token”: “<api_token>”
  “msg”: “Correct register code”
}
```

```
404 Not Found
{
  “msg”: “Incorrect register code, request a code at tfg.pedrxd.es”
}
```

Donde:

- *register_code:* Es el código que se entrega al usuario para registrar su app de 8 caracteres. Debe de caducar si no se utiliza en 2 días.
- *api_token:* Es el código que utilizará para comunicarse con la API de 128 bits

10.1.2 Obtener códigos efímeros nuevos

Solicitud GET: {endpoint}/api/v1/ble/new

```
X-AUTH-TOKEN: <api_token>
```

Respuesta:

```
200 OK
{
      “msg”: “Generated <N> news codes”,
      “codes”: [“<code1>”, “<code2>” … N],
}
```

```
401 Unauthorized
```

```
WWW-Authenticate: Bearer realm=”Use of the Token Api”
{
      “msg”: “A user token is necesary for use this endpoint”
}
```

Donde:

- *codes:* Son los códigos bluetooth que debe de emitir el dispositivo.

10.1.3 Notificar positivo

Solicitud POST: {endpoint}/api/v1/ble/report

```
X-AUTH-TOKEN: <api_token>

{
  "validation_token": "<validation_token>",
  "codes": [
    {
      "blecode": "",
      "datetime": "",
      "exposition": "",
      "signal_max": "",
      "signal_overage": "",
      "latitude": "",
      "longitude": "",
      "precision": "",
      "speed": ""
    },
    {
      "blecode": "",
      "datetime": "",
      "exposition": "",
      "signal_max": "",
      "signal_overage": "",
      "latitude": "",
      “longitude": "",
      "precision": "",
      "speed": ""
    }
  ]
}
```

Respuesta:

```
200 OK
{
      “count”: <n>,
      [“invalid”: <N>],
      “msg”: “Correctly received <n> codes”
}
```

```
401 Unauthorized
```

```
WWW-Authenticate: Bearer realm=”Use of the Token Api”
{
      “msg”: “A user token is necesary for use this endpoint”
}
```

Donde:

- *blecode*: Código que identifica al usuario en contacto
- *datetime*: Especifica el timestamp en el que se realizó el primer encuentro
- *exposition*: Tiempo total del encuentro
- *signal_max*: Intensidad de señal máxima durante el encuentro
- *signal_overage*: Intensidad de señal media durante el encuentro
- *latitude*: Latitud en la que se realizó el encuentro
- *longitude*: Longitud en la que se realizó el encuentro
- *precision*: Precisión de la ubicación GPS
- *speed*: Velocidad cuando se realizó el encuentro.

10.1.4 Comprobar contactos

Solicitud GET: {endpoint}/api/v1/check/

```
X-AUTH-TOKEN: <api_token>
```

Respuesta:

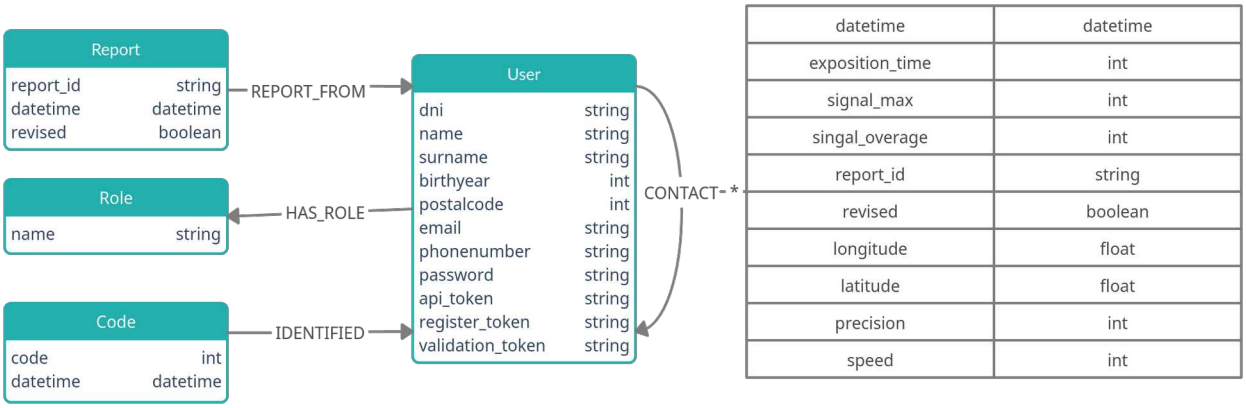
```
200 OK
{
      “count”: <Count>,
      “msg”: “You have <Count> contact registered”
}
```

```
401 Unauthorized
WWW-Authenticate: Bearer realm=”Use of the Token Api”
{
      “msg”: “A user token is necesary for use this endpoint”
}
```

10.2. Estructura de la base de datos

En esta sección vamos a describir las estructuras de los datos a guardar en la base de datos y en la App móvil.

10.2.1 Servidor



En la parte servidor no se guardarán los datos por tablas, como podríamos esperar de base de datos relacionales. Se guardarán en dos tipos de estructuras, por un lado tendremos los nodos y por otro las relaciones. En ambas se pueden guardar atributos. Los **nodos** serán los siguientes:

User: Esta estructura sera almacenada en la base de datos del servidor. Deberá de contener información personal del usuario, una contraseña para poder iniciar sesión, y el tipo de perfil:

- **dni**: DNI de la persona
- **name:** El nombre de la persona, para que el rastreador pueda identificarle.
- **surname:** Apellidos de la persona, para que el rastreador pueda identificarle.
- **birthyear:** Información que puede ser de interés para el epidemiólogo.
- **postalcode:** Información que puede ser de interés para el epidemiólogo.
- **email:** Segunda vía para contactar con el usuario
- **phonenumber:** Primera vía para contactar con el usuario.
- **password:** Para iniciar sesión en la página web
- **api_token:** Token con el que la app móvil identificara al usuario

- **register_token:** Token que introducirá el usuario en la app para obtener el TokenMovil.
- **validation_token:** Token para poder enviar los contactos al servidor.

Role: Indica el tipo de role que posee un usuario.

- **name**: Nombre del rol

Code: Se almacenarán los códigos efímeros en este nodo. Relacionados con el usuario al que pertenece.

- **Code:** Código hexadecimal de 8 bytes.
- **datetime:** Fecha de la creación del código

Report: Guarda información sobre un usuario positivo

- **report_id:** Id de 8 bytes aleatorio
- **datetime:** Fecha de creación del reporte
- **revised:** True si un rastreador ha comprobado el reporte o false si no lo ha comprobado.

Las **relaciones** que disponemos son las siguientes:

(User a)-[:Contact]→(User b): Relaciona un usuario con otro para identificar que ha habido un contacto. Guarda información relacionada con el contacto.

- **Usuario a:** Usuario que ha dado positivo
- **Usuario b:** Usuario con el que ha estado en contacto
- **datetime:** La hora y la fecha de la primera exposición.
- **exposition_time:** Tiempo total que se ha tenido cerca
- **signal_max:** Guarda la potencia máxima de exposición relativo a la señal bluetooth
- **signal_overage:** Guarda la exposición media a la señal bluetooth.
- **report_id:** Id del reporte al que pertenece
- **revised:** True si un rastreador a revisado un contacto o false en caso contrario.
- **GPS:** Información relativa al GPS (Opcional)
  - **longitude:** Longitud de la ubicación
  - **latitude:** Latitud de la ubicación

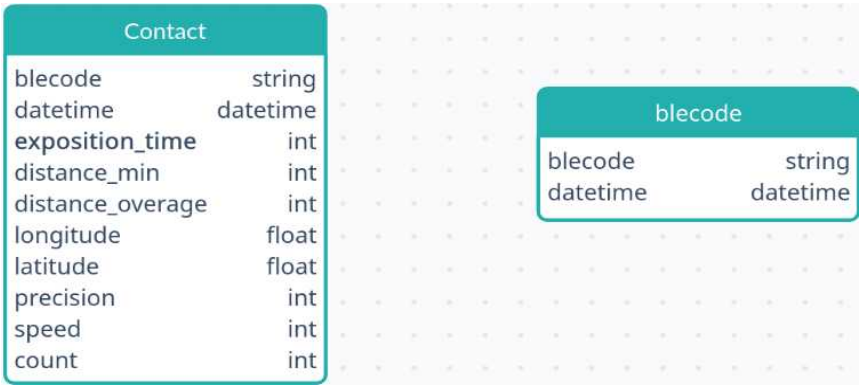
- **precision:** Precisión de la ubicación en metros
- **speed**: Velocidad al tener el primer contacto

(Code a)-[:IDENTIFIED]→(User b): Relaciona un código efímero con un usuario. No tiene atributos

(User a)-[:HAS_ROLE]→(User b): Indica los roles que posee un usuario. No tiene atributos

(Report a)-[:REPORT_FROM]→(User b): Relaciona un reporte de positivo con un usuario. No tiene atributos

10.2.2 Móvil



En la parte móvil. Los datos se guardarán utilizando Sqlite3 a través de la librería Room. Tendremos dos tablas, una de ellas es para guardar los contactos y otra para guardar los códigos efímeros disponibles para utilizar.

Aproximación/Contacto Móvil: Esta tabla conforma el encuentro con un código bluetooth que sera guardado en un dispositivo móvil.

- **blecode:** Código bluetooth que se ha encontrado.
- **datetime:** La hora y la fecha de la primera exposición.
- **exposition_time:** Tiempo total que se ha tenido cerca
- **distance_min:** Guarda la distancia mínima de la exposición
- **signal_overage:** Guarda la distancia media de la exposición
- **count:** Guarda el número de encuentros con un código
- **GPS:** Información relativa al GPS (Opcional)
  - **longitude:** Longitud de la ubicación

- **latitude:** Latitud de la ubicación
- **precision:** Precisión de la ubicación
- **speed:** Velocidad al tener el primer contacto

BleCodes: Para guardar los códigos que son generados en el servidor

- **blecode:** Código bluetooth
- **datetime:** Fecha que fue obtenido del servidor

10.3. Referencias

- **Beacons** → Segmento de datos retransmitido a traves de bluetooth para detectar la presencia de otros dispositivos. Contienen datos sobre la proximidad y el identificador efímero
- **Identificador Efímero** → Código de 6 byte que identifican a una persona durante un tiempo determinado. Son únicos y generados por el servidor.
- **Token Autentificación** → Es un token de identificación HTTP Bearer. Es utilizado para que el servidor pueda reconocer el usuario que está realizando la petición.
- **Código de acceso** → Este código es de un único uso. Debe de ser dado a los usuarios para que sea introducido en la app. A partir de este se obtiene el Token de Autentificación.
- **Código de validación** → Otorgado al usuario una vez que se obtiene un caso positivo de covid (U otra enfermedad). Es introducido en la app, los datos de contagio son enviados al servidor.

10.4. Preparación de los contenedores de Docker

Para preparar las imágenes se ha creado un directorio llamado docker/ en la raíz de los proyectos panel de control y la api y un directorio aparte para la imagen de nginx.

10.4.1 Panel de control y Api

Tanto para el panel de control como para la api se ha utilizado la misma configuración variando el nombre y destino de los ficheros. La carpeta está formada por 3 archivos:

- Dockerfile: Este fichero contiene los pasos necesarios para obtener la imagen final del programa. Cada línea está compuesta por una instrucción y los argumentos.

```
FROM php:7.4-fpm

WORKDIR /opt/panel

# Install dependencies
RUN apt-get update && apt-get install wget curl -y
RUN wget
https://raw.githubusercontent.com/composer/getcomposer.org/76a7060ccb
93902cd7576b67264ad91c8a2700e2/web/installer -O - -q | php --
RUN apt-get install zip unzip libzip-dev -y
RUN docker-php-ext-install sockets zip

#Copy project
COPY assets/ /opt/panel/assets/
COPY bin/ /opt/panel/bin/
COPY config/ /opt/panel/config/
COPY public/ /opt/panel/public/
COPY src/ /opt/panel/src/
COPY templates/ /opt/panel/templates/
COPY translations/ /opt/panel/translations/

COPY composer.json /opt/panel/composer.json
COPY symfony.lock /opt/panel/symfony.lock
COPY composer.lock /opt/panel/composer.lock
COPY package.json /opt/panel/package.json
COPY webpack.config.js /opt/panel/webpack.config.js
COPY .env /opt/panel/.env


#Prepare symfony
RUN php composer.phar install --no-dev --optimize-autoloader

#Config PHP-FPM
RUN mv "$PHP_INI_DIR/php.ini-production" "$PHP_INI_DIR/php.ini"
RUN rm /usr/local/etc/php-fpm.conf.default && rm /usr/local/etc/php-
fpm.d/www.conf.default
```

```
COPY docker/www.conf /usr/local/etc/php-fpm.d/www.conf

#Expose
EXPOSE 9001

VOLUME ["/var/www/panel/"]

COPY docker/entrypoint.sh /bin/entrypoint.sh
RUN chmod +x /bin/entrypoint.sh

#Entrypoint
ENTRYPOINT ["/bin/entrypoint.sh"]
```

- entrypoint.sh: Este script es ejecutado al lanzar el contenedor. Su función es copiar los ficheros de la aplicación actualizada al volumen desde donde es leída por php. Posteriormente lanza el programa php-fpm.

```
#!/bin/bash

#Copy files to volume
echo "Copy code to destination..."
cp -r /opt/panel /var/www/
chown www-data:www-data -R /var/www/panel

#Launch php-fpm
php-fpm
```

- www.conf: Este fichero contiene configuración que utilizará php-fpm para lanzar la aplicación. Contiene información como el número de núcleos que se utilizarán, el puerto en el que escuchará las peticiones, etc.

10.4.2 Nginx

El contenedor de nginx se ha realizado con la imagen nginx como base por lo que la única función de nuestro Dockerfile es substituir la configuración. La configuración esta formada por 3 ficheros:

- Dockerfile:

```
FROM nginx
RUN rm /etc/nginx/conf.d/default.conf
COPY api.conf /etc/nginx/conf.d/api.conf
COPY panel.conf /etc/nginx/conf.d/panel.conf
```

- api.conf: Configuración para la página del servidor de registro.

```
server {
    listen       8080;
    root /var/www/api/public;
    location / {
```

```
        try_files $uri /index.php$is_args$args;
    }
 location ~ ^/index\.php(/|$) {
        fastcgi_pass api:9000;
        fastcgi_split_path_info ^(.+\.php)(/.*)$;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME
$realpath_root$fastcgi_script_name;
        fastcgi_param DOCUMENT_ROOT $realpath_root;
        internal;
    }
    location ~ \.php$ {
        return 404;
    }
}
```

- panel.conf: Configuración del panel de control

```
server {
    listen       8081;

    root /var/www/panel/public;

    underscores_in_headers on;

    location / {
        try_files $uri /index.php$is_args$args;
    }

    location ~ ^/index\.php(/|$) {
        fastcgi_pass panel:9000;
        fastcgi_split_path_info ^(.+\.php)(/.*)$;
        include fastcgi_params;


        fastcgi_param SCRIPT_FILENAME
$realpath_root$fastcgi_script_name;
        fastcgi_param DOCUMENT_ROOT $realpath_root;

        internal;
    }

    location ~ \.php$ {
        return 404;
    }

}
```

10.5. Tecnologías utilizadas

10.5.1 Room

Room es una librería de Android que ofrece una capa de abstracción sobre SQLite permitiendo un acceso más robusto a la base de datos. Entre las facilidades que ofrece verifica en tiempo de compilación las sentencias SQL, anotaciones sencillas que minimizan consultas repetitivas. [14]

Los tres componentes principales son:

- Entidades: Son una representación de las tablas de la base de datos.
- DAO: Contiene las consultas para acceder a los datos
- Base de datos: Sirve como punto de acceso principal a la base de datos.

10.5.2 Docker

Docker es una plataforma abierta para el desarrollo y despliegue de aplicaciones. Docker ejecuta las aplicaciones en contenedores que permiten abstraer el sistema que está por debajo. Los contenedores se pueden ejecutar independientemente del sistema operativo que se utilice y dejan de ser un problema las dependencias porque el propio contenedor trae las librerías necesarias para la ejecución.

Docker contiene una serie de herramientas que permiten configurar y lanzar de manera sencilla varios servicios a la vez. Una de las más sencillas es docker composer que mediante un fichero yaml se puede configurar el lanzamiento de varios servicios independientes a la vez.

10.5.3 Beacons

Los beacons son dispositivos que emiten una señal a modo de identificador, utilizan la tecnología Bluetooth. Para la transmisión del mensaje utilizaremos el formato EddyStone diseñado por Google a través de la librería Android Beacon Library.

10.5.3.1 Eddystone [15]

Son tramas que se emiten a través de bluetooth. Existen cuatro tipos de tramas que se utilizan para diferentes fines.

- Eddystone-UID: Es útil para identificar dispositivos.

- Eddystone-URL: Utilizado para transmitir URL y que los clientes puedan acceder a ella.
- Eddystone-TLM: Contiene información para telemetría como puede ser el voltaje de la batería, número de paquetes etc..
- Eddystone-EID: Utilizado para retransmitir códigos efímeros durante un tiempo. Los códigos deben de ser resueltos remotamente por un servicio.

En nuestro caso vamos a utilizar los Eddystone-EID por que queremos retransmitir los códigos efímeros para que luego el servidor resuelva quien ha sido el contacto.

10.5.4 Framework Symfony

Symfony es uno de los frameworks más importantes en el desarrollo web. Contiene una serie de componentes reusables llamados *bundles* que pueden ser cargados según la necesidad.

Además cuenta con una gran comunidad y documentación por lo que la introducción a este framework es sencilla. La curva de aprendizaje no es muy grande, siempre y cuando se domine PhP. De cierta manera, Symfony te obliga a utilizar las mejores prácticas en el desarrollo por lo que se obtendrá software de calidad siempre y cuando se mantengan las pautas recomendadas.

10.5.4.1 Stimulus

Stimulus es un framework de Javascript para interaccionar con una web html de manera sencilla. Permite el desarrollo rápido abstrayendo el acceso a elementos del dom a través de “targets” y también ayuda en el procesamiento de eventos como puede ser al presionar un botón.

Una de las ventajas que tiene utilizar esta librería sobre otras es la simplicidad y lo bien integrada que esta en Symfony.

10.5.5 Base de datos Neo4j

El APIrest y el Panel de Control deben de almacenar la información en una base de datos. Para ello, vamos a utilizar una base de datos de grafos, más en concreto Neo4J. Esta utiliza el lenguaje Cypher para realizar consultas

Esta nos permitirá hacer búsquedas utilizando grafos obteniendo datos muy interesantes. Se puede

obtener una lista de posibles contagiados no solamente del contacto más cercano, si no contactos de los contactos para hacer un seguimiento más exacto.

10.6. Servicios utilizados

10.6.1 DigitalOcean

DigitalOcean es un proveedor de servicios IaaS, PaaS y SaaS. Desde un panel de control puedes realizar la reserva de un VPS, registros para docker, bases de datos y multitud de servicios más que pueden ser consultados en su web.

Los VPS los denominan “Droplets”.

El cobro lo realizan por horas a unos precios muy competitivos.

La URL de la web es digitalocean.com

Bibliografía

1: Pablo Linde, España tiene menos de la mitad de los rastreadores necesarios: https://elpais.com/sociedad/2020-07-18/espana-tiene-menos-de-la-mitad-de-los-rastreadores-necesarios-para-controlar-la-epidemia.html
2: Enrique Perez, Corea del Sur pide a las operadoras localizar a todos los que estuvieron en la zona de bares contagiada : https://www.xataka.com/privacidad/app-rastreo-gps-corea-sur-pide-a-operadoras-localizar-a-todos-que-estuvieron-zona-bares-contagiada
3: Apple, Google, Exposure Notifications: https://www.google.com/covid19/exposurenotifications/
4: Álex Llorca Llinares, Aplicaciones de rastreo de vecinos europeos: https://www.eldiario.es/tecnologia/aplicaciones-rastreo-europa-sistemas-vecinos_1_6172292.html
5: , Decentralized Privacy-Preserving Proximity Tracing: https://github.com/DP-3T/documents/blob/master/DP3T%20White%20Paper.pdf
6: PEPP-PT Team, Pan-European Privacy-Preserving Proximity Tracing : https://github.com/pepp-pt/pepp-pt-documentation/blob/master/PEPP-PT-high-level-overview.pdf
7: PEPP-PT Documentation, :
8: PEPP-PT Team, Robert Specification V1: https://github.com/pepp-pt/pepp-pt-documentation/blob/master/10-data-protection/ROBERT-concept-illustration.pdf
9: TCN Coalition, TCN Protocol: https://github.com/TCNCoalition/TCN
10: Oracle, ¿Que es una base de datos orientada a grafos?: https://www.oracle.com/mx/big-data/what-is-graph-database/
11: Ken Schwaber, Jeff Sutherland, The Scrum Guide: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf
12: , Save data in a local database using Room: https://developer.android.com/training/data-storage/room/
13: Android, Android Service Overview: https://developer.android.com/guide/components/services
14: Google, Como guardar contenido en una base de datos en Android: https://developer.android.com/training/data-storage/room/
15: Google, Eddystone-UID: https://github.com/google/eddystone/tree/master/eddystone-uid