PROYECTO FIN DE CICLO

IES JOSÉ LUIS MARTÍNEZ PALOMO

Departamento de Informática y Comunicaciones
Técnico Superior en Desarrollo de Aplicaciones Multiplataforma

PressureTrack: Un Servicio Web para el seguimiento de la presión arterial

Autor: **Guillermo Palazón Cano**
Tutor: **Antonio Alcaraz**

Alquerías, 6 de junio de 2024

1. Objetivos del proyecto. Definición.

En la actualidad, la forma en la que se accede a la información y los servicios online y la forma en la que se comunican entre sí las aplicaciones, los servicios web se han convertido en herramienta imprescindible para el desarrollo de aplicaciones multiplataforma.

Al utilizar servicios web, los desarrolladores pueden crear una única API (del inglés, application programming interface, en español, interfaz de programación de aplicaciones) que maneja el negocio y los datos, y las interfaces de usuario solamente tienen que hacer uso de esta API sin importar el lenguaje de programación en el que están desarrolladas, el dispositivo que hace uso de las mismas, sistema operativo, etc. Esto hace que sean esenciales y se reduzca el tiempo de desarrollo de aplicaciones que funcionan de manera consistente en múltiples plataformas, como Android, iOS y la web.

El proyecto PressureTrack va a consistir en el desarrollo de un servicio web con un objetivo muy concreto y es el registro y mantenimiento de usuarios en el mismo y que para estos se pueda llevar un control de la presión arterial que tienen de manera individual.

Este servicio permite dar de alta usuarios, modificar su clave e incluso eliminarlos si así se quisiera y para ellos se podrá ir registrando la presión arterial que tienen en un momento específico y acceder en cualquier momento al histórico de estas mediciones.

1.1 Objetivos del proyecto

Según la Sociedad Española de Hipertensión-Liga Española para la Lucha contra la Hipertensión Arterial (Seh-Lelha), únicamente en España existe más de 14 millones de personas que sufren de hipertensión, enfermedad provocada por tener unos niveles inadecuados de presión arterial. Esta hipertensión es una enfermedad que de no ser controlada y tratada adecuadamente puede derivar en enfermedades muy peligrosas como puede ser la insuficiencia cardiaca, infarto, derrame cerebral, deterioro cognitivo, demencia, etc.

Si bien es verdad que muchas veces, existe un componente genético en la hipertensión, muchas veces también está relacionada con el estilo de vida o la dieta.

Los riesgos de tener esta enfermedad y la gran cantidad de casos existentes deben tenernos siempre vigilantes de la misma, ya que en algunos casos se manifiesta de manera clara con síntomas como dolores de cabeza, dificultad de respirar, retención de líquidos, entre otros… pero otras veces estos síntomas no suelen aparecer de manera tan clara o no solemos asociarlos a que puedan ser provocados por esta tensión arterial.

PressureTrack nace con el objetivo de proporcionar un componente que pueda ser usado desde otras aplicaciones para que estas puedan llevar este control de una manera muy sencilla, permitiendo registrar mediciones de nuestra presión arterial sistólica y diastólica y poder consultar los últimos datos que hayamos registrado en la misma, para llevar en todo momento un control de la misma y poder ver si tenemos que tomar alguna medida o debemos acudir a nuestro médico de cabecera en el caso de tener una tendencia que se aleja de la adecuada, ya que la aplicación se centrará

únicamente en su medición, debiendo ser siempre un médico, el responsable de decidir cuál es el mejor tratamiento en el caso de que fuera necesario.

1.2 Descripción del proyecto

Una vez que hemos visto la importancia de esta presión arterial y que nuestro objetivo es crear un componente que permita llevar de manera sencilla este registro para que los datos sean fácilmente accesibles vamos a definir un poco más en qué va a consistir PressureTrack.

Este proyecto va a nacer como un servicio web de tipo REST que podrá ser utilizado directamente desde aplicaciones que se comunican como servicios web tipo Postman o con aplicaciones desarrolladas para dispositivos iOS, Android, etc.

Nos hemos decantado por un servicio web ya que puede ser utilizado posteriormente en cualquier sistema y también porque gozan de gran actualidad, ya que cada vez es más frecuente oír el término microservicio a la hora de desarrollar software donde, a través de APIs bien definidas que son desarrolladas por equipos pequeños e independientes y que son utilizados por aplicaciones para conseguir sus objetivos.

¿Cómo funciona realmente un servicio web?

1. El cliente realiza una petición a través del protocolo HTTP a través de una aplicación cliente (que podría ser incluso un navegador, aunque es cierto que en ocasiones para el intercambio de datos no es la aplicación más cómoda).
2. La petición se envía a una URL específica (endpoint) que representa un recurso en el servidor. Esta petición además puede ser de diferentes tipos: GET (obtener datos), POST (crear), PUT (actualizar), DELETE (eliminar), etc.
3. Esta petición incluye un cuerpo en ocasiones con datos (otras veces van en la propia URL) y también incluye una serie de encabezados y detalles de autenticación (como JWT que serán explicados más adelante en profundidad).
4. El servidor recibe la petición y la procesa, realizando las operaciones (si se tiene permiso) que fueran necesarios y que suelen hacer necesario de operaciones con la base de datos.
5. El servidor genera una respuesta que suele incluir un código de estado HTTP y que indica el resultado de la operación. Valores frecuentes, el 200 (OK) indicando que todo ha ido bien, 201 (CREATED) indicando que la creación de datos ha resultado exitosa, 400 (BAD REQUEST) para algún error, 401 (UNATHORIZED) si no se tiene permisos para realizar esa operación, etc.
6. El servidor envía la respuesta al cliente, esta suele contener en muchas ocasiones datos en formato JSON y es el cliente el que debe ser capaz de manejar esta respuesta.

Si bien es cierto que esto va a ser transparente al usuario, ya que este lo único que va a tener que conocer es una serie de URL y una serie de datos que debe enviar, tanto los datos como la aplicación va a estar desplegada en la nube. De esta forma inicialmente vamos a tener pocos recursos dedicados (menor coste) a nuestro servicio web, pero estos podrían escalarse si vemos que el número de usuarios aumenta, aumentando tanto los recursos de la base de datos como del servidor donde está

desplegada. Además, nos vamos a garantizar alta disponibilidad, mayor seguridad y un mantenimiento y una gestión más simplificada que si hubiéramos optado por mantener esta información y máquinas locales con contenedores Docker como era nuestra primera opción, pero que por cuestiones relativas al proveedor de servicios de internet fueron finalmente descartadas.

En nuestro servicio web debemos tener en cuenta que estamos tratando con datos médicos, debemos dotar al mismo de seguridad ya que no debemos permitir que personas ajenas pudieran acceder a historiales médicos ni que un usuario accediera a las mediciones de presión arterial de otro, por lo que debemos añadir una capa de seguridad para que solamente cada usuario pueda acceder a su información en cada momento.

Concluyendo este apartado, PressureTrack va a consistir en un servicio web para el seguimiento, control y gestión de la presión arterial ya que va a permitir a los clientes registrar tanto su presión arterial sistólica como diastólica y poder consultar estos registros. Este servicio web está abierto a que sea usada con aplicaciones propias para ello o que se desarrollen aplicaciones multiplataforma que hagan uso del mismo teniendo en cuenta que la información no será local al dispositivo y en el que la seguridad será fundamental.

1.3 Definición del alcance, limitaciones y beneficios del sistema

El servicio web PressureTrack nace con el objetivo de alcanzar el máximo número posible de usuarios, ya que es un componente que podrá ser utilizado por cualquier aplicación que lo desee.

Los sistemas que deseen usar el servicio web lo único que deberán tener cuenta es conocer para usar convenientemente los endpoints diseñados del mismo, por lo que haciendo uso de estos a través de su URL y pasando los datos adecuados, ya sea a través de la ruta o del cuerpo de la petición (incluyendo cabeceras de autenticación en algunos casos) proporcionarán al usuario los resultados deseados.

El alcance de este servicio web no puede estar previsto ya que dependerá de que existan más o menos aplicaciones que quieran hacer uso de este, por ello inicialmente será gratuito para favorecer que sea apetecible su uso. Esto es posible a que se contará con servicios gratuitos tanto para la base de datos como en el servicio de despliegue. Tanto el servicio de base de datos como el servidor de despliegue tiene la posibilidad de poder ser escalado y contratar una mayor capacidad en el caso de la base de datos y de mayor velocidad y potencia de recursos para el servidor de despliegue.

Una limitación que puede tener nuestro servicio web es que partimos de un conjunto inicial de opciones que realiza nuestro servicio web que pueden resultar escasas en algunas aplicaciones. Nuestro servicio web irá actualizándose con frecuencia incluyendo nuevas opciones ya pensadas por el equipo como la incorporación de avatar para el usuario o la posibilidad de envío de mensajes para recuperar contraseñas olvidades. Además, se incrementarán también a partir de demanda de las aplicaciones que usen nuestro servicio.

Otra de las principales limitaciones iniciales de nuestro servicio web es que no nace asociado a ninguna aplicación ya que únicamente vamos a lanzar el microservicio que

permita hacer esta gestión de usuarios y registros de presión arterial. Hemos realizado un estudio de mercado y existen ya aplicaciones que realizan una gestión de presión de arterial como son MyDiary, Diario de presión arterial o Presión Arterial.

Uno de los pasos que vamos a hacer para fomentar el uso de nuestro servicio web es ofrecer a estas aplicaciones (aunque algunas hace tiempo que no están actualizadas y es posible que ya no se encuentren en mantenimiento) y otras similares el uso de nuestro servicio web, facilitándoles en el caso de que sea necesario la migración de datos que pudieran tener.

Además de ofrecérselo a estas aplicaciones también vamos a ofertar este servicio a aquellos desarrolladores de software que suelen realizar aplicaciones relacionados con la salud y a aquellas empresas que suelan trabajar con aparatos que midan esta presión arterial.

¿Por qué creemos que podemos tener éxito en estos sectores? En muchas ocasiones estas aplicaciones almacenan los datos de las mediciones en local, es decir, se guardan en bases de datos tipo SQLite que solamente se encuentra en el propio dispositivo por lo que las mediciones de un dispositivo no se encuentran en otro (salvo que sea la propia base de datos la que esté subida en la nube). Además, si un usuario cambia de móvil o incluso si únicamente desinstala la aplicación se pierden todas las mediciones que ha tenido hasta ahora. Trabajar con el servicio web, hace que la aplicación se deba despreocupar de toda la gestión de la base de datos y únicamente deba utilizar el servicio que se le ofrece.

Otras aplicaciones seguro que están haciendo uso de servidores propios para este mantenimiento, haciendo uso de nuestro servicio web también se deben olvidar de toda esta gestión que seguramente le está llevando un tiempo extra de mantenimiento y de gestión de datos (copias de seguridad), disponibilidad de estar 24 horas los 365 días, rendimiento de los servicios, etc. Toda esta gestión puede ser olvidada por su parte.

Todos estos son beneficios que nuestro servicio web va a proporcionar y que creemos que puede hacer que haga que estas aplicaciones se decanten por hacer uso de este en lugar de otras soluciones propias. Conforme nuestro servicio sea imprescindible para estas aplicaciones será el momento de monetizar el mismo y sacarle un rendimiento económico acorde al servicio prestado.

2. Análisis del sistema

Una vez que tenemos definido nuestro proyecto, debemos establecer el sistema que nos va a permitir desarrollarlo. Para ello, debemos investigar en primer lugar los requisitos y requerimientos que se necesitan, para que nuestro proyecto software sea efectivo y pueda resultar exitoso.

Dentro de este análisis vamos a comenzar explicando el proceso de obtención de información que hemos seguido, el estudio de los recursos que van a ser necesarios, tanto humanos, como recursos hardware y software y vamos a finalizar el apartado estableciendo una planificación y un estudio de viabilidad.

2.1 Obtención de información

Para analizar la información que es necesaria para nuestra aplicación he estudiado un poco sobre la presión arterial, analizando que al final esta información se basa principalmente en dos parámetros, la presión sistólica y diastólica.

Al tener cercanos familiares con este tipo de problemática de salud, ya era una información que conocía, pero aun así he investigado sobre el tema y he visto que estos valores son los más importantes a la hora de llevar este registro y también he investigado los valores máximos y mínimos que son recomendados para estos valores.

También, han sido analizadas aplicaciones similares (algunas de ellas han sido nombradas con anterioridad) para obtener información sobre qué datos registran, cómo los piden, como los muestran, etc., ya que tener un punto de vista que ya haya abordado este problema nos puede dar ideas sobre los datos más requeridos y peticiones más apropiadas para los mismos.

Esta obtención de información que hemos realizado no se ha centrado únicamente en los datos sino como ya hemos dicho en las peticiones, de esta manera podemos tener de una forma clara con qué servicios mínimos se debe lanzar nuestro nuestro microservicio API REST y cuáles deben ser los siguientes pasos para que el mismo se afiance.

2.2 Recursos necesarios

En este apartado del proyecto nos vamos a centrar en hacer un desglose de todos los recursos que van a ser necesarios para poder implementar los requerimientos de nuestra aplicación.

Dentro de estos recursos nos vamos a centrar en los recursos humanos que lo van a implementar, el hardware necesario y una parte no menos importante que es el software utilizado para llevar a cabo el proyecto.

También vamos a dedicar un artículo en el que vamos a describir la planificación que vamos a tratar de llevar a cabo para que el proyecto sea viable y tengamos garantías de que cumple los objetivos en el plazo correspondiente. Para ellos vamos a identificar y temporalizar las fases de este, identificando las principales actividades o tareas que deben llevarse a cabo en el desarrollo del proyecto, secuenciando las mismas y haciendo una estimación de su duración.

2.2.1 Recursos humanos y hardware

Debemos tener en cuenta que es un proyecto final de ciclo formativo y que este se está realizando de manera individual y no en grupo, por tanto, como recurso humano contamos únicamente con una única persona que será el encargado de desarrollar todas las tareas que componen el proyecto. Eso sí, dispondrá de la ayuda y la compañía a lo largo del mismo de su tutor.

En el caso de que este proyecto finalmente viera la luz con cierto éxito y se tuviera que ampliar enormemente el servicio web que ofrece, incluyendo incluso nuevos ámbitos como podría ser la gestión de glucosa para diabéticos, el equipo podría ser ampliado, debiendo de contar con otros componentes que se encargaran de desarrollar las diferentes partes de este servicio web.

En cuanto al hardware, para el desarrollo del proyecto no ha hecho falta hacer ningún desembolso y se ha premiado trabajar con herramientas gratuitas. El equipo que se está utilizando para desarrollar el proyecto es el equipo personal del alumno del proyecto. No es una máquina excesivamente potente ya que ya tiene algunos años y todo el software instalado en la misma para este desarrollo es software libre que no consume una gran cantidad de recursos.

En cuanto al sistema operativo de la máquina y debido al software que se va a utilizar para desarrollar el proyecto, se podría haber utilizado cualquiera ya que son programas que están disponibles de manera libre para los principales sistemas operativos.

Respecto al hardware, podríamos pensar también en los servidores donde alojar los datos y donde desplegar la aplicación. Si bien como ya he anticipado se barajó en un principio la posibilidad de tener estos en un servidor o máquina propia teniendo todos estos servicios dockerizados. Al final, casi que por cuestiones ajenas se descartó esta opción ya que se debía conseguir que el proveedor de servicios de internet nos permitiera acceder a estos desde fuera de la red interna y era un trabajo extra para sumar al proyecto. Además, se pensó que para alcanzar otros mecanismos de disponibilidad, seguridad y rendimiento estos servicios sería mejor que estuvieran alojadas y desplegados con servicios que nos dieran esta garantía y que pudieran ser escalables en función de las demandas que pudiera alcanzar nuestro servicio web.

En la búsqueda de estos se han estado barajando muchas opciones siendo siempre una de las principales premisas es que inicialmente fueran gratuitos, fáciles de usar y que permitieran ser escalados en función de la demanda de uso.

Para el almacenamiento de la base de datos y tras una exhaustiva búsqueda en la que se estudiaron diferentes servicios como Firebase, nos hemos decantado por **Supabase** ya que tiene una opción inicial gratuita que nos permite utilizar una base de datos relacional (PostgreSQL) que creemos que es muy adecuada a los requerimientos de nuestro proyecto.

Para el despliegue de la aplicación también hemos analizado diferentes opciones, y si en un principio nos habíamos decantado con Heroku, finalmente estuvimos viendo que la versión gratuita no cumplía con el mínimo que exigía este proyecto, por lo que al final hemos optado por **render**, que nos permite el despliegue de nuestro API Rest de manera sencilla ya que está totalmente sincronizado con el repositorio GitHub que

contiene el proyecto y que nos proporciona una URL “fija” de acceso a los endpoints de nuestro servicio web.

2.2.2 Recursos software

En este apartado vamos a especificar algunos de los principales programas, así como tecnologías que vamos a emplear para poder desarrollar nuestro proyecto, especialmente aquellas cuya importancia es mayor y cuyo empleo ha requerido de una mayor investigación y estudio, por ser tecnologías en algunos casos que no se habían estudiado en el desarrollado del ciclo formativo y, por tanto, ha sido necesaria inclusive seguir alguna formación para poder hacer uso de estas.

- Sistema operativo. En nuestro caso, el sistema operativo de la máquina que va a desarrollar el software es un sistema MacOS y no ha hecho falta cambiar el mismo. Por el software a utilizar, el sistema podría haber sido desplegado desde cualquier otro sistema (UNIX/Linux o un sistema operativo Windows).
- Entorno de desarrollo integrado (IDE). Para el desarrollo del servicio web que vamos a llevar a cabo tenemos a nuestra disposición un amplio abanico de IDEs como Intellij o Eclipse. Por su sencillez y por conocimiento del mismo, vamos a utilizar Netbeans para este desarrollo.
- Lenguaje de programación JAVA para el desarrollo de API Rest. Nuestro servicio web está basado en el framework Spring, utilizando el lenguaje de programación Java para la implementación de este.
- Frameworks y librerías: Para el Desarrollo de nuestro servicio web haremos uso Spring framework. Este Framework permite construir aplicaciones web Java (y otros sistemas) con mucha simplicidad ya que integra con facilidad diversas tecnologías y librerías. Dentro de Spring framework vamos a destacar los dos principales recursos que hemos utilizado del mismo: Spring Boot y Spring Security.
- Base de datos relacional PostgreSQL. Para el almacenamiento de nuestros datos vamos a disponer de una base de datos relacional PostgreSQL, que como ya hemos comentado va a estar alojada en el servicio Supabase.

Todos estos recursos software que hemos comentado son gratuitos, no hace falta hacer ningún desembolso económico para la utilización de los mismos y son multiplataforma, por lo que pueden ser utilizados en la mayoría de los sistemas operativos existentes.

2.3 Planificación y estudios de viabilidad

A la hora de abordar un proyecto una cuestión imprescindible en el mismo es determinar si este va a ser o no viable. Para ello vamos a hacer una planificación y un estudio de este y vamos a determinar un plan de acción a seguir.

En primer lugar, hemos diferencia las fases en las que está dividido nuestro proyecto, las cuáles son: Fase de definición del proyecto, Fase de implementación, Fase de pruebas, Fase de documentos y Fase de mantenimiento.

2.3.1 Fase de definición del proyecto. Análisis y diseño de este.

Partiendo de la idea que tenemos de crear un servicio web y que este verse sobre un tema de salud tan sensible como es de la monitorización de la presión arterial, en esta primera etapa principalmente nos hemos centrado en darle forma al proyecto.

Teniendo la idea tenemos que plantearnos las tecnologías apropiadas para desarrollar el mismo y los conocimientos que son necesarios para poder desarrollarla y de los cuáles se carecen. Nos hemos decantado por utilizar algunos contenidos ya vistos en algunos módulos, como el de Acceso a datos, pero ampliando estos ya que si bien allí se vio el desarrollo de Spring Boot para un proyecto de estas características nos faltaba la parte en la que se desarrolla el Servicio Web API Rest.

Por otra parte, hemos también estudiado qué sistema gestor de base de datos podría ser más adecuado para la solución que deseamos conseguir, obteniendo como resultado que mejor optar por una solución relacional como puede ser MySQL o PostgreSQL en lugar de una no relacional.

Una vez que tenemos claro el SGBD y viendo que vamos a alojar la misma en la nube, hemos barajado varias opciones para alojar la misma. Como ya hemos comentado y tras analizar varias opciones, nos hemos decantado finalmente por el servicio Supabase ya que nos permite el alojamiento de bases de datos de tipo relacional, en este caso PostgreSQL.

Otra cuestión importante en esta fase de análisis era dónde alojar nuestro servicio web. Tras analizar varios de los más relevantes como Heroku, AWS o Parse, nos hemos decantado por Render, ya que además de que tiene opción gratuita para arrancar el proyecto, nos proporciona mucha facilidad para desplegar nuestros servicios web en el mismo, proporcionándonos una URL para que otras aplicaciones puedan acceder a nuestros servicios.

Si nos centramos en la **viabilidad** del proyecto, podemos ver que económicamente es totalmente viable ya que no ha hecho falta hacer ningún gasto, ya que el equipo para el desarrollo es el equipo personal propio del alumno y no ha hecho falta hacer ninguna compra, el software es todo libre y no ha hecho falta tampoco hacer ninguna adquisición en este sentido y tanto para la base de datos como para el despliegue de la aplicación se ha optado por dos soluciones que inicialmente tienen una parte gratis que es la que va a ser usada a expensas del éxito o no del microservicio, y que en el caso, de que este lo tuviera, optar por una solución de pago que tampoco es elevada y que haría que el proyecto siguiera siendo viable.

Por tanto, si no tenemos en cuenta el desembolso hecho anteriormente por el equipo, ni la conexión a internet, luz y otros posibles gastos que se pueden considerar asumidos por el alumno, el coste del proyecto sería de 0 euros. Está claro, que realmente lleva aparejado un coste, aunque sea la luz invertida en el desarrollo del mismo.

Teniendo claro lo que queremos hacer, que es un servicio web, con una base de datos que estará alojada de tipo PostgreSQL y que se desplegará en Render, los siguientes pasos que hemos seguidos en este análisis y diseño ha sido darle forma a nuestra base de datos.

Para la base de datos, debemos tener en cuenta que la parte de usuarios deben cumplirse ciertos requisitos para que la implementación de Spring Security sea sencilla

y que debe estar relacionada con la parte en la que se guardan los registros de presión arterial para los usuarios.

Dentro de este análisis y diseño hemos decidido también cuáles iban a ser los endpoints con los que vamos a lanzar el servicio y que nos garantizan un mínimo de funcionalidad para el mismo. De entre estos servicios debemos diferenciar entre los servicios para la gestión de los usuarios y los servicios para la gestión de los registros de la presión arterial. Estos son un mínimo, conforme el servicio web vaya creciendo se irán añadiendo nuevos.

Para los usuarios necesitamos crear usuarios, que se puedan loguear los mismos, cambiar su contraseña y borrar un usuario si quisiera (este borrado implica el borrado de todos los registros de presión arterial que se tienen asociados de este).

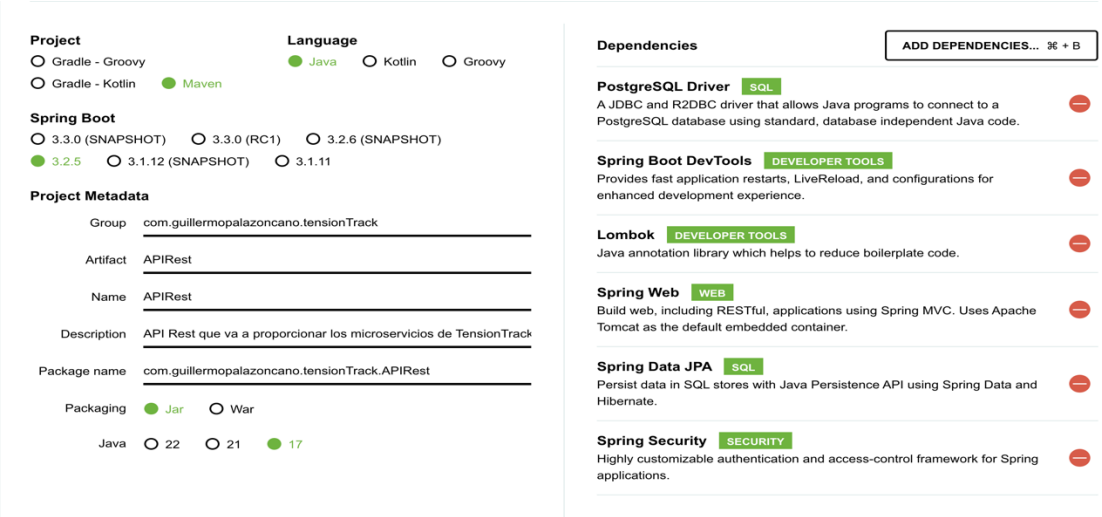
Respecto a los registros de presión arterial debemos ser capaces de dar un registro para un usuario, poder consultar todos los registros del usuario, consultar los registros entre dos fechas y borrar un registro específico.

En cualquier caso, no debemos olvidar que esto es una fase inicial de análisis y diseño y que alguna de las decisiones que son tomadas aquí, podría ser modificadas, pudiendo tener como consecuencias que estas modificaciones conlleven otras modificaciones en cadena.

2.3.2 Fase de implementación del proyecto

Es la fase que conlleva una mayor carga horaria de las fases del proyecto, ya que supone el desarrollo en sí del proyecto. En esta fase vamos a indicar algunos de los principales hitos que debemos conseguir de cara a que el proyecto tenga éxito.

1. Diseño físico de la base de datos. Es una base de datos sencilla con solamente dos registros y que se crea directamente con el Editor de base de datos del que dispone Supabase.
2. Generación del proyecto. Para la creación y lanzamiento del proyecto se deberá hacer uso de Spring Boot, ya que te genera un esqueleto con una funcionalidad mínima de aplicación (servicio) web, con las dependencias y librerías elegidas. Un proyecto Spring Boot se puede generar de muchas formas (en algunos IDEs incluso va incorporado), siendo una de las frecuentes a través de Initializr (https://start.spring.io), al que indicando el tipo de proyecto (en nuestro caso será Maven), el lenguaje (en nuestro caso será Java), la versión de Spring Boot, los metadatos e indicando las dependencias ya te genera un esqueleto de proyecto. Un ejemplo, similar al nuestro (solamente cambiaría el nombre del proyecto que fue modificado con posterioridad sería).



3. Desarrollo de los endpoints del servicio web. En este paso debemos implementar los endpoints correspondientes a los usuarios y sus registros que fueron seleccionado en la fase de definición del proyecto.

4. Implementación de seguridad. Si siempre la seguridad de los datos es de vital importancia, en este proyecto que se están gestionando datos médicos debemos trabajar la misma un poco más, no debiendo permitir un uso inadecuado de los datos. Para la realización de esta tarea y por no tener conocimiento ninguno de su funcionamiento será necesario de una formación y de un estudio previo.

5. Despliegue del proyecto. En este paso debemos ser capaces de sincronizar nuestro proyecto con el control de versiones github y configurar el servidor render para que sea capaz de “beber” la información de allí, de tal manera que cuando se produzcan cambios en el github, el proyecto que se encuentra desplegado también se actualice para los usuarios de nuestro servicio web siempre acudan a la última versión.

Para realizar un punto, no debe ser necesario haber finalizado con el punto anterior obligatoriamente. Por poner ejemplos, a la misma vez que se van desarrollando endpoints, se debe ir implementando la seguridad y se puede ir desplegando el proyecto (al menos la parte que no necesita seguridad) para ir probando que despliega adecuadamente.

2.3.3 Fase de pruebas

En esta fase, cada uno de los endpoints que componen nuestro servicio web deben ser concienzudamente probados para tratar de detectar posibles errores y corregir los mismos.

Para poder probar estos, tendremos que hacer uso de la aplicación Postman que es una potente herramienta gratuita (aunque tiene versión de pago que para nuesras pruebas no será necesaria) que permite realizar peticiones de una manera simple y poder testear APIs de tipo REST.

Para realizar estas pruebas no debemos haber finalizado la fase de implementación ya que se irán realizando de manera progresiva conforme se va desarrollando el código de los endpoints.

2.3.4 Fase de documentación

Aunque la fase de documentación suele ser con frecuencia la más descuidada no será así en nuestro proyecto, teniendo en cuenta, que esta memoria ya puede ser considerada documentación ya que contiene información de interés sobre el mismo y que su consulta puede ser conveniente para entender el mismo.

Para la documentación del servicio web se hará uso de la herramienta Swagger (https://swagger.io) que consiste en un conjunto de reglas, especificaciones y herramientas que ayudan a documentar APIs. Esta documentación será de gran interés para todas las aplicaciones que decidan usar nuestro servicio web. Esta herramienta también es desconocida por lo que será también necesaria una formación previa.

2.3.5 Fase de mantenimiento

La fase de mantenimiento va a ser posterior a la entrega de este proyecto, en cualquier caso, nos gustaría hacer mención sobre ella para que quedará reflejada en esta memoria. Dentro del mantenimiento del proyecto vamos a resaltar dos tipos de mantenimiento principalmente. Por un lado, el mantenimiento correctivo y por otro el mantenimiento perfectivo.

Aunque el servicio web va a ser probado y queremos que los errores que pudiera tener sean mínimos, ninguna aplicación está libre de tener algún fallo. En el mantenimiento correctivo se corrigen errores o defectos que, a pesar de las pruebas no se detectaron.

Con el mantenimiento perfectivo se incluyen en nuestro servicio web aquellas mejores que pueden ser de rendimiento o nuevas funciones adiciones que pueden aportar beneficios al mismo y que, muchas de ellas son descubiertas conforme el cliente vaya utilizando el mismo.

2.3.6 Actividades o tareas

Por tanto, podemos concluir que las principales tareas que vamos a tener que realizar son las siguientes, debiendo tener en cuenta que, si bien vamos a secuenciar las mismas en un orden “idílico” de realización, muchas de estas tareas se van a realizar de forma paralela como ya se ha ido comentando. En otros casos sí está más clara la dependencia de no poder realizar una hasta que no esté realizada otra. Por ejemplo, no es posible llevar a cabo ningún endpoint del servicio web hasta que no esté creada la base de datos.

Tras un exhaustivo repaso de las distintas fases, las tareas o actividades más importantes que podemos identificar, teniendo en cuenta que el proyecto debe comenzar el 11 de marzo a implementarse (el 8 de marzo se nos comunicó el tutor del proyecto), debe finalizar el 5 de junio (6 de junio es la fecha de entrega del mismo) se propone la siguiente secuenciación de tareas con una duración estimada para cada una de ellas.

Fase de definición del proyecto

- Planteamiento del proyecto → 0,5 horas. Fecha realización: 11/03/2024 – 11/03/2024.
- Designación tecnologías → 0,5 hora. Fecha realización: 11/03/2024 – 11/03/2024.
- Estudio plataformas para alojamiento base de datos y servicios web. Elección de estas → 1,5 horas. Fecha realización: 11/03/2024 – 11/03/2024.
- Estudio viabilidad proyecto → 0,5 horas. Fecha realización: 12/03/2024 – 12/03/2024.
- Modelo entidad/relación → 0,5 hora. Fecha realización: 14/03/2024 – 14/03/2024.
- Selección endpoints servicio web → 1,5 horas. Fecha realización: 14/03/2024 – 15/03/2024.

Fase implementación del proyecto

- Diseño físico de la base de datos → 1 hora. Fecha realización: 18/03/2024 – 18/03/2024.
- Generación del proyecto → 1 horas. Fecha realización: 19/03/2024 – 19/03/2024.
- Desarrollo endpoints servicio web → 20 horas. Se desglosa en:
    - Endpoints usuarios → 10 horas. Fecha realización: 20/03/2024 – 07/04/2024.
    - Endpoints registros presión arterial 10 horas. Fecha realización: 2/04/2024 – 19/04/2024.
- Implementación seguridad servicio web (incluye formación) → 25 horas. 22/04/2024 – 15/05/2024
- Despliegue del proyecto → 3 horas. Fecha realización: 6/05/2024 – 8/05/2024.

Fase pruebas

- Pruebas servicio web → 15 horas. Fecha realización: 2/04/2024 – 30/05/2024

Documentación

- Memoria proyecto → 5 horas. Fecha realización: 20/05/2024 – 5/06/2024
- Documentación servicio web → 5 horas. Fecha realización: 28/05/2024 – 05/06/2024

Total horas estimadas: 80 horas.

2.3.7 Diagrama de gantt

Para el desarrollo del diagrama de Gantt hemos partido de la plantilla con licencia creative commons (https://create.microsoft.com/es-es/template/diagrama-de-gantt-simple-4bf6b793-490f-4623-84ca-c9c6251a91fc) y la hemos modificado para que se adapte a nuestras necesidades y, por supuesto, a nuestros datos.

Una vez que tenemos en cuenta las distintas tareas con sus fechas, el diagrama de Gantt resultante es el siguiente.

PressureTrack

Guillermo Palazón Cano

Inicio del proyecto: lun, 2024-03-11

Semana para mostrar: 1

TAREA	ASIGNADO A	INICIO	FIN
Definición del proyecto			
Planteamiento proyecto	Guillermo	11-3-24	11-3-24
Designación tecnologías	Guillermo	11-3-24	11-3-24
Estudio plataformas alojamiento. Elección	Guillermo	12-3-24	12-3-24
Estudio viabilidad proyecto	Guillermo	14-3-24	14-3-24
Selección endpoints servicio web	Guillermo	14-3-24	15-3-24
Implementación proyecto			
Diseño físicco base datos	Guillermo	18-3-24	18-3-24
Generación del proyecto	Guillermo	19-3-24	19-3-24
Endpoints usuarios	Guillermo	20-3-24	7-4-24
Endpoints registros	Guillermo	2-4-24	19-4-24
Seguridad servicio web	Guillermo	22-4-24	15-5-24
Despliegue del proyecto	Guillermo	6-5-24	8-5-24
Fase pruebas			
Pruebas endpoints	Guillermo	2-4-24	30-5-24
Fase documentación			
Memoria	Guillermo	20-5-24	5-6-24
Documentación servicio web	Guillermo	28-5-24	5-6-24

3. Diseño del sistema

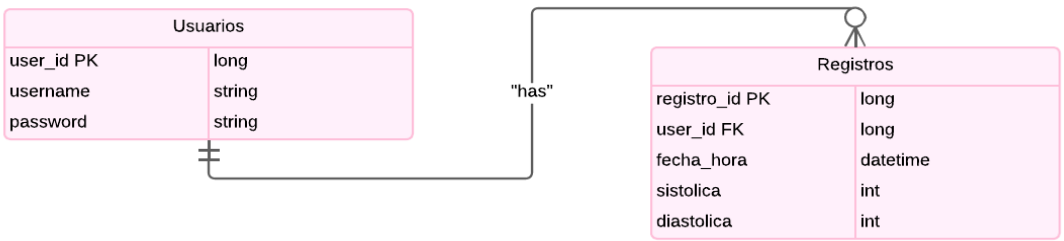
Al carecer un servicio web de interfaz, en este apartado de diseño nos vamos a centrar fundamentalmente en dos partes diferenciadas. Por un lado, vamos a ver la estructura de nuestra base de datos y el modelo entidad relación resultante y, por otro lado, vamos a tratar el diseño y parametrización de los distintos endpoints que componen nuestro servicio web, en el que detallaremos los tipos de petición que serán, sus valores a recibir en los mismos y las respuestas que deberán dar los mismos.

3.1. Diseño de los datos. Modelo Entidad relación.

Para registrar nuestros usuarios, Spring Security nos exige unos mínimos. Si bien es cierto que estos no son gran cosa y están en relación con la información que teníamos pensado almacenar sobre los mismos. En este sentido, Spring Security solamente nos exige necesita tener una tabla donde estén los usuarios con un campo donde se almacena el username. Esta columna no obliga a tener ningún tipo de valor, podría recibir el mail del usuario o algún Nick (la aplicación que utilice el servicio que decida el valor que quiera almacenar en el mismo). Además del username, Spring Security nos obliga a tener una columna con la contraseña del usuario (se registrará cifrada), pero vamos que este dato es de habitual registro siempre, por lo que Spring Security no nos está obligando a nada que no tuviéramos ya contemplado y que fuera necesario.

Spring Security también trabaja con roles para los usuarios. Dotando de posibilidad de uso de endpoints en función de que el usuario logueado tenga uno u otro rol. Tras un análisis de nuestra aplicación no nos es necesario el uso de roles ya que solamente distinguiremos dos tipos de servicios, los que pueden ser usados sin necesidad de estar logueado (como poder hacer un registro de usuario o hacer un login) y los que será necesario estar logueado (consultar los registros de presión arterial de un usuario). De esta manera como no vamos a usar los roles tampoco vamos a tener una tabla que almacene los mismos.

Por tanto, nuestra base de datos va a estar formada inicialmente por dos tablas únicamente, una en la que se encuentran almacenados los usuarios y otra en la que se encuentran almacenadas las presiones arteriales de dichos usuarios. El modelo entidad relación resultante sería algo tan sencillo en el que podemos ver las dos tablas y una única relación 1:N existente entre ambas tablas.



3.2. Tecnologías utilizadas.

Aunque ya se hizo una introducción a estas tecnologías y se nombraron algunas de ellas en un anterior apartado de recursos software, vamos a comentar en este

apartado con un poco más de profundidad algunas de las principales tecnologías que hemos utilizado para desarrollar nuestro proyecto.

- Spring framework. Framework que permite construir aplicaciones Java (y en la actualidad también de otros sistemas como Kotlin) con mucha simplicidad y que integra con facilidad diversas tecnologías y librerías, proporcionando un esqueleto de funcionalidad que solamente necesita de retoques específicos para las peculiaridades de una aplicación.
- Spring Boot es parte fundamental en la tecnología Spring, ya que es un lanzador de este tipo de proyectos, genera toda la configuración inicial y tecnologías que va a utilizar el proyecto Spring, acelerando el desarrollo de microservicios como es nuestro caso, ya que unifica toda la gestión de librerías y dependencias, facilitándolo el despliegue del servidor proporcionando un servidor Tomcat sin tener que programar nada.
- Spring Security es otro módulo fundamental del framework Spring para dotar de seguridad a sus aplicaciones. Permite la implementación de OAuth o el uso de token Json Web Tokens (incluso la combinación de ambas tecnologías). En este proyecto hemos incluido el uso de Spring Security por la necesidad de dotar de seguridad a los datos ya que esta aplicación cuenta con datos sensibles que no deben poder ser accedidos por cualquiera. Es un módulo muy importante en este proyecto.
- Lombok. En los proyectos muchas veces se tiene un código muy repetitivo o similar. Lombok es una librería de Java que trata de reducir el código que se escribe. Para ello se nutre de anotaciones como pueden ser @Getter, @Setter, @NoArgsConstructor, @AllArgsConstructor, @RequiredArgsConstructor, @Builder, @Data, @Value, etc., entre muchas otras.
- Base de datos PostgreSQL. En este proyecto se hace uso de un Sistema Gestor de Base de Datos Relacional de tipo PostgreSQL y que va a estar alojado en una plataforma en la nube (Supabase).
- JWT (Json Web Tokens). Dentro de la seguridad que hemos dotado a la aplicación queremos destacar el uso de JWT que es un estándar abierto de creación de tokens ampliamente utilizado en seguridad, especialmente relacionados con la autenticación en las aplicaciones y la autorización a utilizar las mismas. JWT está integrado con Spring Security y nos permite que los usuarios se pueden logear respecto al servicio API Rest y que ciertos servicios del API Rest solamente puedan ser accedidos por estos usuarios que se encuentran logueados, incluyendo incluso conceptos más amplios como el de rol, en el que solamente se puede acceder a uno u otros servicios en el caso de que se tenga del rol adecuado.

3.3. Diseño de procedimiento. Endpoints del servicio web.

Este apartado lo vamos a dedicar a diseñar los diferentes endpoints que debe tener nuestro servicio web. Un endpoint es una funcionalidad de nuestro servicio web, las aplicaciones clientes podrán hacer uso de estas a través de peticiones (HTTP

Request) y nuestro servicio API Rest, les dará una respuesta a las mismas (HTTP Response).

Cada recurso del servicio web tiene que ser identificado por una dirección web (URL) con una estructura determinada y la respuesta también tendrá una estructura determinada con una parte de la respuesta en formato de texto (generalmente JSON o simplemente una única cadena).

Las peticiones pueden ser de distintos tipos en función de las operaciones que realice la funcionalidad concreta. Entre estas podemos destacar

- GET → Equivale a la operación CRUD Read. Solicita la recuperación de datos.
- POST → Equivale a la operación CRUD Create. Se utiliza para enviar una entidad o un recurso específico causando cambios en los datos del servidor.
- PUT → Equivale a la operación CRUD Update. Se utiliza para reemplaza la representación actual del recurso en el destino.
- DELETE → Equivale a la operación CRUD Delete. Se utiliza para borrar un recurso específico.
- Existen otras más como PATCH o HEAD.

En las respuestas que da nuestro servicio tenemos que diferenciar dos partes, por un parte el código de respuesta y por otro lado la parte texto de la respuesta. Al ser una respuesta HTTP, estos son códigos de estado de respuesta HTTP. Estos se suelen agrupar en grupos: Los que van del 100 al 199 son respuestas informativas, del 200 al 299 respuestas satisfactorias, del 300 al 399 redirecciones, del 400 al 499 errores de los clientes y del 500 al 599 errores de los servidores, pudiendo ser utilizados otros códigos de 3 dígitos aunque no es una práctica recomendable.

Entre los códigos de estado HTTP más utilizados tenemos:

- Respuestas satisfactorias: 200 OK (todo correcto), 201 Created (se ha creado el registro), 202 Accepted (se acepta la solicitud para un posterior procesamiento) y 204 No Content (todo correcto con una respuesta en blanco).
- Errores cliente: 400 Bad request (no se entiende la petición), 401 Unathorized (no autorizado), 403 Forbidden (prohibido), 404 Not found (no se han encontrado datos), 405 Method not Allowed (el método no sirve para ese tipo de solicitud) y 409 Conflict (no puede realizarse por conflicto en el servidor).
- Errores del servidor: 500 Internal Server Error (error interno del servidor)., 501 Not implemented (no implementado) y 503 Service Unavailable (servidor temporalmente no disponible).

Existe cierta estandarización sobre los códigos de estado HTTP que se deben devolver en según qué petición se haga. Nos basaremos en los mismos para seguir dicho estándar (pueden ser consultados por ejemplo en el siguiente enlace https://www.dariawan.com/tutorials/rest/http-methods-spring-restful-services/).

Además del código de error devolverá siempre un mensaje de éxito en formato cadena o una respuesta en formato JSON.

3.3.1 Endpoints usuarios

Las primeras funcionalidades que vamos a proporcionar en función a la gestión de usuarios son las siguientes:

- Registro de usuarios → Se deberá indicar el username, password y verificación de la contraseña. Devolverá el código de éxito 201 (CREATED) si todo ha ido correctamente (y devolverá el Usuario para que ya podamos acceder a su código de usuario) y el código 400 (BAD REQUEST) si no se ha podido llevar a cabo dicho servicio. No es necesario estar logueado para llevar a cabo esta funcionalidad (si no tienes usuario no puede estar logueado).
- Login de usuario → Se deberá indicar el username y password. Devolverá un código de éxito 201 (CREATED) y un 400 (BAD REQUEST) si el login no ha resultado exitoso. Aunque CREATED parece que está más orientado a cuando se inserta un registro en la base de datos, en este caso devolverá un 201 puesto que se crea el JWT del usuario al realizar el login del mismo. No es necesario estar logueado (de hecho, es el servicio que realiza login de los mismos por lo que no tendría ningún sentido).
- Cambiar contraseña. Debe indicarse la contraseña antigua, la nueva y la verificación de esta. Este método necesita que el usuario esté logueado por lo que también habrá que proporcionar una cabecera con el JWT de dicho usuario. Se devolverá el código de éxito 200 (OK) si todo ha ido correcto y 400 (BAD REQUEST) si ha habido problemas al realizar dicho cambio. Se necesita estar logueado para cambiar la contraseña.
- Borrar usuario. Esta funcionalidad lleva consigo el borrado de los registros del usuario. Se borrará aquel usuario cuyo JWT se indique en la cabecera de autenticación de la petición al endpoint, por lo que el usuario debe estar logueado para poder proceder a su borrado. Devolverá el código de éxito 204 (NO CONTENT) si todo ha funcionado de manera correcta y el código de error 404 (NOT FOUND) si no se ha podido realizar este borrado.

3.3.2 Endpoints registros presión arterial

Las primeras funcionalidades que vamos a proporcionar en función a la gestión de los registros de presión arterial son las siguientes:

- Obtener todos los registros de presión arterial de un usuario. Se obtendrán todos los registros de presión arterial para el usuario logueado que se haya pasado logueado en un JWT en la cabecera. Devolverá un listado con registros (que podría estar vacío, si no encuentra registros para el usuario) y siempre devolverá el código de éxito 200 (OK).
- Obtener todos los registros de presión arterial de un usuario entre dos fechas. Funcionará exactamente igual que el anterior con la salvedad que habrá que indicar una fecha inicio y una fecha fin de la que se quieren obtener dichos registros.
- Crear un registro de presión arterial. Habrá que indicar la medición diastólica y sistólica y se creará un registro para el usuario logueado que se indique en la

cabecera de autenticación de la petición. Devolverá el código de éxito 201 (CREATED) si todo ha ido correctamente (y devolverá el Registro de presión arterial para que podamos acceder a su código de usuario) y el código 400 (BAD REQUEST) si no se ha podido llevar a cabo dicho servicio.

- Borrar un registro de presión arterial. Se deberá indicar el código del registro. Se comprobará que el registro pertenece al usuario logueado que se ha indicado en la cabecera de la petición. Los códigos de error que podrá devolver son 404 (NOT FOUND) si no existe el registro, 403 (FORBIDDEN) si el registro no pertenece al usuario logueado y por tanto tiene prohibido borrarlo. En el caso de que todo haya ido correctamente y se haya podido realizar el borrado devolverá el código de éxito 204 (NO CONTENT).

4. Implementación

En este apartado nos vamos a centrar en diferentes apartados de la implementación, separando tres partes, por un lado, la base de datos, por otro lado, el servidor de despliegue de la aplicación y, finalmente en la implementación del servicio web.

Los dos primeros servicios básicamente ha sido configurarlos adecuadamente, pero al ser servicios que no tenemos que implementar (básicamente utilizar), su carga de trabajo ha sido mucho menor que el desarrollo del servicio web, por lo que será en este aspecto al que le dedicaremos un mayor esfuerzo en su desarrollo en esta memoria.

4.1. Implementación de la base de datos.

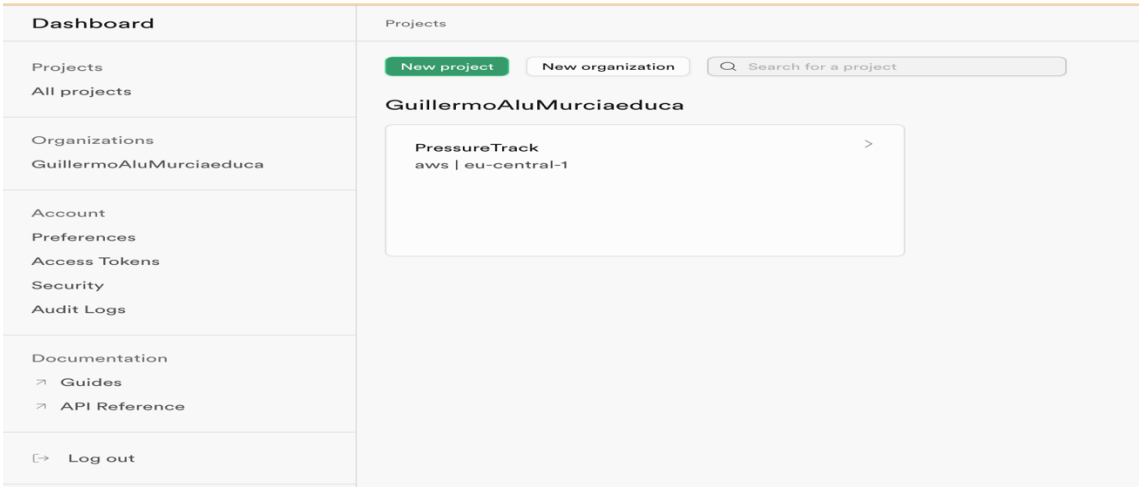
Supabase es una solución completa para el desarrollo de aplicaciones web y móviles, proporcionando una serie de herramientas para crear y gestionar los backends.

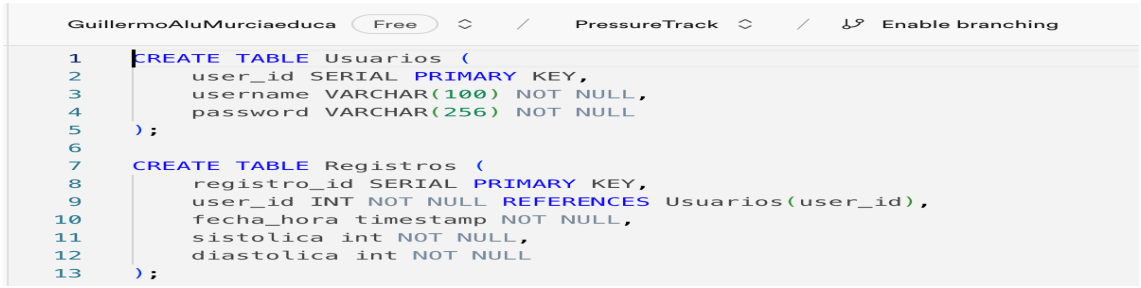
Entre estos servicios y herramientas en la nube que nos proporciona y que tenemos a nuestra disposición, una por las que está destacando y que está haciendo que Supabase esté siendo muy usada es poder gestionar bases de datos. En este caso concreto es una base de datos relacional (PostgreSQL).

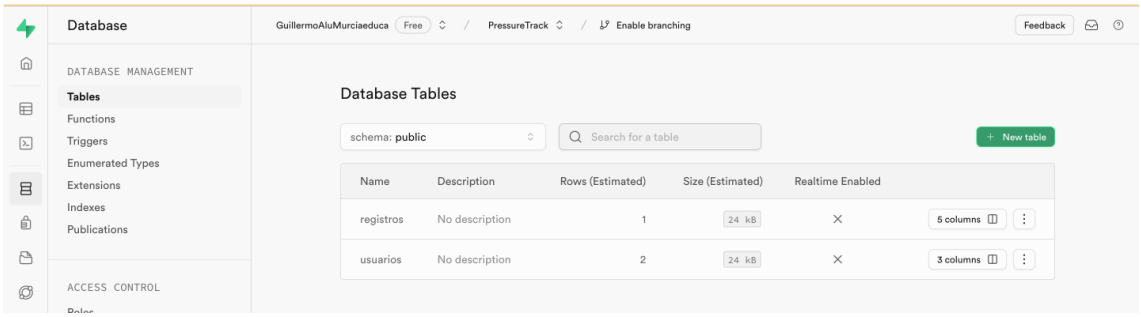
Una de las ventajas que presenta Supabase es su plan gratuito ideal para sitios web simples o proyectos que están iniciando (tiene un límite de dos proyectos). No obstante, tiene algunos planes de pago cuya relación calidad precio es muy buena, y permite escalar este servicio progresivamente.

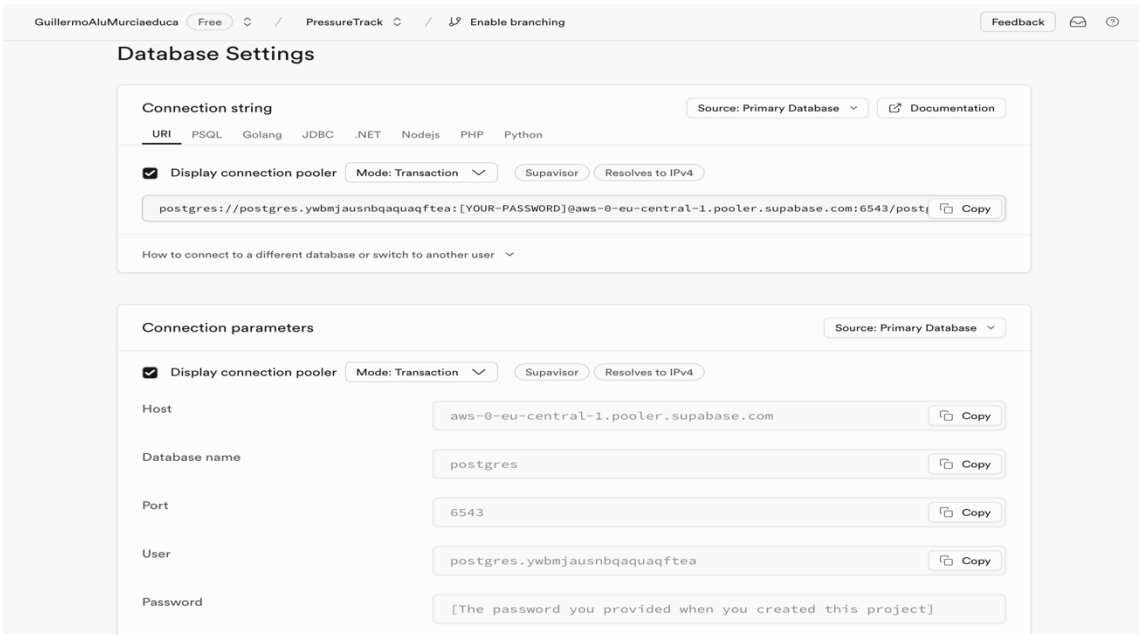
Por tanto, nos hemos registrado en Supabase con el usuario GuillermoAluMurciaeduca, hemos creado el proyecto PressureTrack y dentro de dicho proyecto hemos creado la base de datos, con las dos únicas tablas que se debían crear a partir de nuestro modelo entidad relación (Supabase tiene un editor SQL por lo que no hace falta tener un cliente para la creación o la inserción de datos en las mismas, pudiéndose hacer todo desde la propia aplicación online).

Supabase ya te indica directamente la ruta y configuración para poder acceder a la base de datos, con lo que solamente tienes que copiar la ruta (en nuestro caso en nuestro proyecto API Rest lo tendremos en una variable de entorno para que no aparezca directamente el valor).


Dashboard
Projects
All projects
Organizations
GuillermoAluMurciaeduca
Account
Preferences
Access Tokens
Security
Audit Logs
Documentation
Guides
API Reference
Log out
Projects
New project
New organization
Search for a project
GuillermoAluMurciaeduca
PressureTrack
aws | eu-central-1


GuillermoAluMurciaeduca
Free
PressureTrack
Enable branching
CREATE TABLE Usuarios (
user_id SERIAL PRIMARY KEY,
username VARCHAR(100) NOT NULL,
password VARCHAR(256) NOT NULL
);
CREATE TABLE Registros (
registro_id SERIAL PRIMARY KEY,
user_id INT NOT NULL REFERENCES Usuarios(user_id),
fecha_hora timestamp NOT NULL,
sistolica int NOT NULL,
diastolica int NOT NULL
);


Database
DATABASE MANAGEMENT
Tables
Functions
Triggers
Enumerated Types
Extensions
Indexes
Publications
ACCESS CONTROL
Feedback
Database Tables
schema: public
Search for a table
New table
Name
Description
Rows (Estimated)
Size (Estimated)
Realtime Enabled
registros
No description
1
24 kB
5 columns
usuarios
No description
2
24 kB
3 columns


Database Settings
Connection string
Source: Primary Database
Documentation
URI
PSQL
Golang
JDBC
.NET
Nodejs
PHP
Python
Display connection pooler
Mode: Transaction
Supavisor
Resolves to IPv4
postgres://postgres.ywbmjausnbqaquaqftea:[YOUR-PASSWORD]@aws-0-eu-central-1.pooler.supabase.com:6543/postg
Copy
How to connect to a different database or switch to another user
Connection parameters
Source: Primary Database
Display connection pooler
Mode: Transaction
Supavisor
Resolves to IPv4
Host
aws-0-eu-central-1.pooler.supabase.com
Copy
Database name
postgres
Copy
Port
6543
Copy
User
postgres.ywbmjausnbqaquaqftea
Copy
Password
[The password you provided when you created this project]

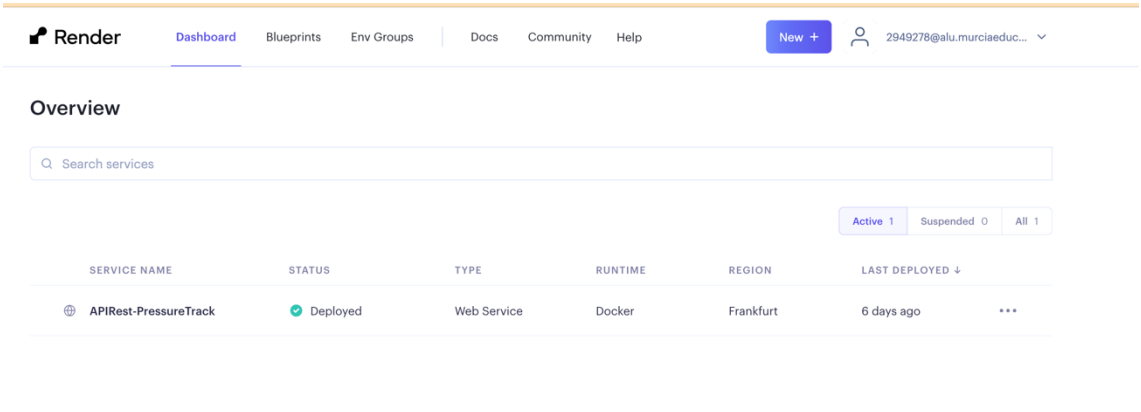
4.2. Despliegue del servicio web.

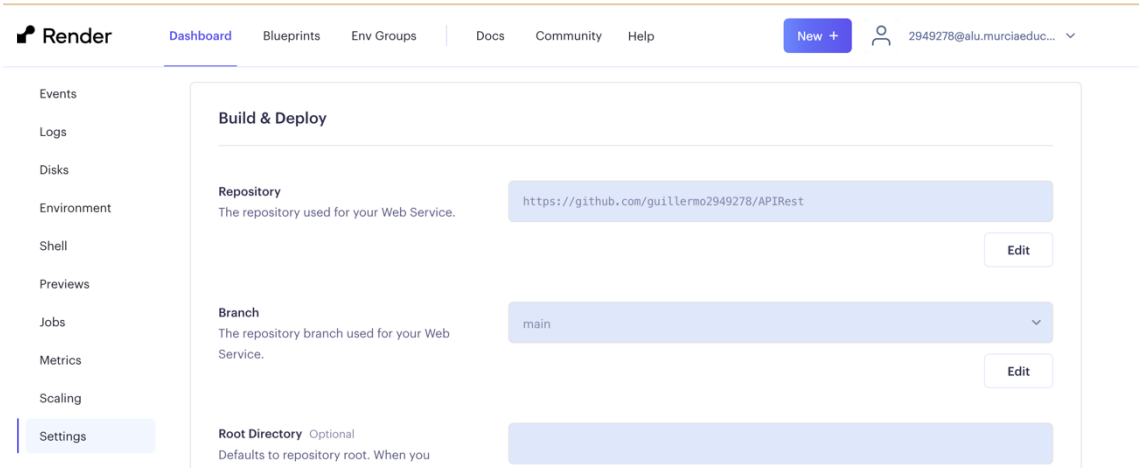
Para el despliegue del servicio web estuvimos barajando opciones clásicas como Heroku, pero al final por los servicios que se ofrecían de manera gratuita y por su simplicidad se ha optado por Render.

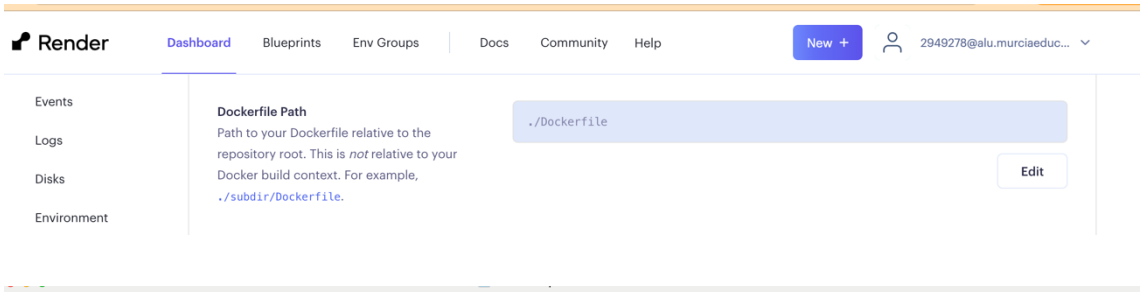
Render es una plataforma de la nube que ofrece servicios de alojamiento y despliegue de aplicaciones web, entre otras funciones. Una de sus cualidades es su sencillez para implementar estos despliegues, tener un servicio web gratuito para poder desplegar algunos de estos servicios con pocos recursos y poder escalar estos en función de las necesidades.

Por tanto, para poder desplegar este servicio nos hemos tenido que crear también en render. En nuestro caso hemos creado esta cuenta utilizando para ello nuestra cuenta GitHub ya que son servicios que van a estar interconectados, ya que la forma que va a tener render es desplegar un proyecto que se encuentra en Github (indicándole repositorio y rama que queremos que despliegue del mismo, que generalmente será main).

En primer lugar, hemos generado el servicio que se llama APIRest-PressureTrack. Lo hemos configurado indicando el repositorio de Github. Para poder desplegar estos servicios (API Rest creados en Java), es necesario también tener un fichero Dockerfile en la raíz del proyecto, donde le indique cómo compilar el proyecto API Rest y el fichero .jar correspondiente, por lo que esta información también se subirá a Github en el fichero Dockerfile y aquí le indicaremos la existencia de dicho fichero para que render sea capaz de leerlo y llevar a cabo la funcionalidad descrita en el mismo.







4.3. Desarrollo del API Rest

xxxx

4.3.1 Spring Boot

xxx

4.3.2 Estructura de paquetes

xxxx

4.3.3 Principales clases del proyecto

Para

5. Bibliografía

- https://www.paho.org/es/noticias/19-9-2023-oms-detalla-primer-informe-sobre-hipertension-arterial-devastadores-efectos-esta
- https://www.cnmc.es/prensa/panel-hogares-usos-internet-20231103