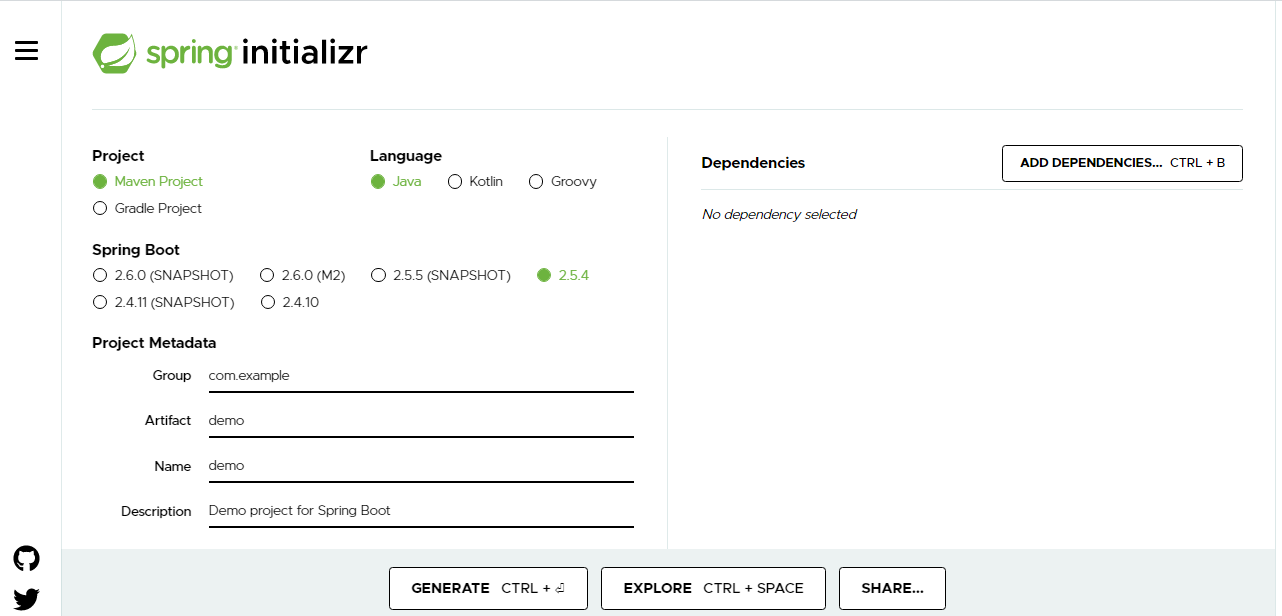
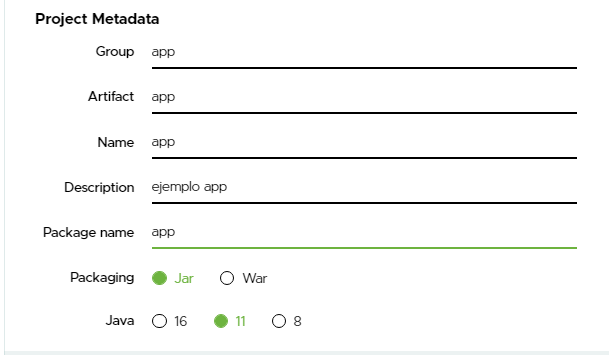
**Creando un proyecto de java springboot**

Vamos a utilizar la herramienta de <https://start.spring.io/> para crear nuestro primer proyecto.



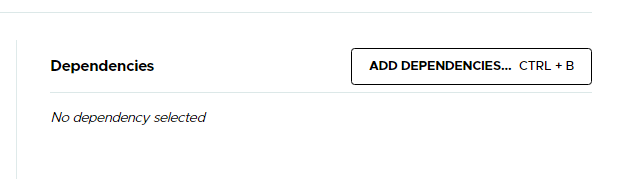
Debemos configurar nuestro proyecto de la siguiente forma:

* **Project:** Seleccionaremos Maven Project.
* **Lenguaje:** Java.
* **Spring Boot:** podemos dejar la que está marcada por defecto o seleccionar la que mejor se ajuste a nuestro requerimiento.
* **Project Metadata**: En esta parte ira el nombre descripción, grupo, Artifact...etc. recomiendo usar un nombre corto como se observa en la imagen de ejemplo.
* **Packaging:** Usaremos la opción de Jar.
* **Java:** Acá seleccionamos en que versión de java vamos a trabajar recomiendo verificar cual tenemos instalada para evitar problemas de compatibilidad

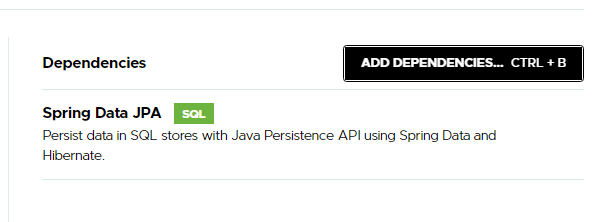


**Agregando dependencias**

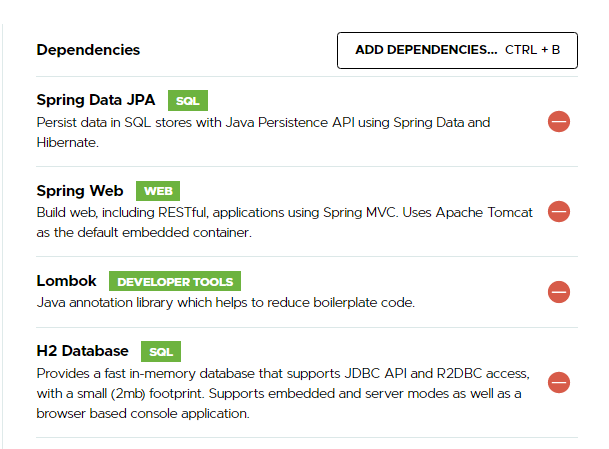
En el panel derecho podemos agregar las dependencias que consideramos que vamos a utilizar.



Al hacer click en ADD DEPENDENCIES se nos desplegara un menú en el cual podremos buscar las dependencias para nuestro proyecto.

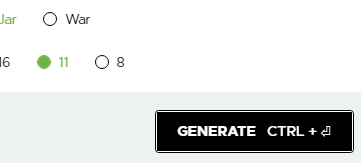


Para este ejemplo que vamos a realizar considero necesario agregar las cuatro dependencias que se observan en la siguiente imagen.

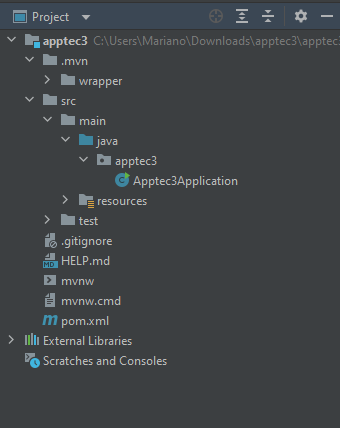


* **JPA** es la propuesta estándar que ofrece Java para implementar un Framework Object Relational Mapping (ORM), que permite interactuar con la base de datos por medio de objetos, de esta forma, **JPA** es el encargado de convertir los objetos Java en instrucciones para el Manejador de Base de Datos (MDB).
* **Spring** Web **MVC** es un sub-proyecto **Spring** que está dirigido a facilitar y optimizar el proceso creación de aplicaciones web utilizando el patrón **MVC** (Modelo-Vista-Controlador), donde el Modelo representa los datos o información que manejará la aplicación web
* **Lombok** es una librería que va a tunear nuestro IDE, que tras instalarla transformará el mismo, ya sea **Spring** Tool Suite o Eclipse. Además, será una dependencia que tenemos que incluir en nuestro proyecto para que podamos añadir una serie de anotaciones.
* **H2 Database** es un sistema de administrador de base de datos relacionales hecho con Java creado principalmente para entornos de desarrollo. Esto quiere decir que no reemplaza de ninguna manera a MySQL, SQL Server u Oracle que deberán seguir siendo usados en ambientes productivos.

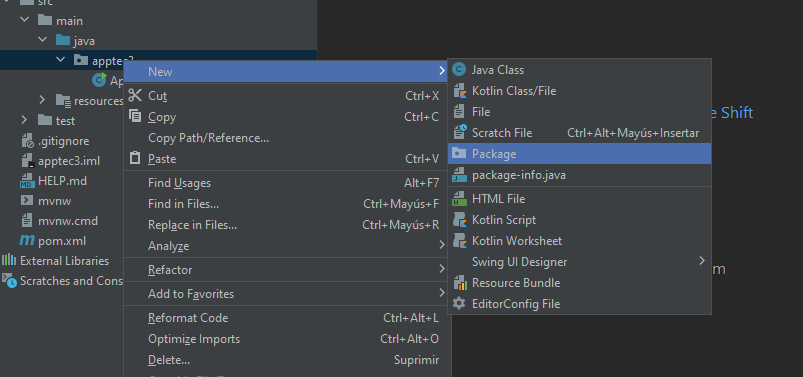
Al finalizar hacemos click el botón GENERATE y nos descargara un archivo comprimido donde estará nuestro proyecto.



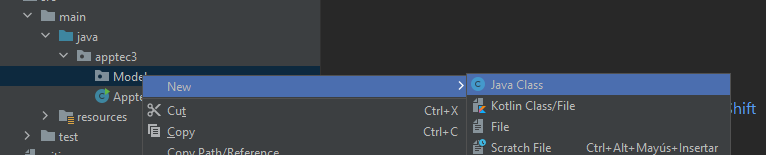
Al descomprimir nuestro proyecto y abrir la carpeta que hay dentro se verá con la siguiente estructura. En mi caso estoy utilizando Intellij IDEA como idea de desarrollo en java, también podemos usar Eclipse.

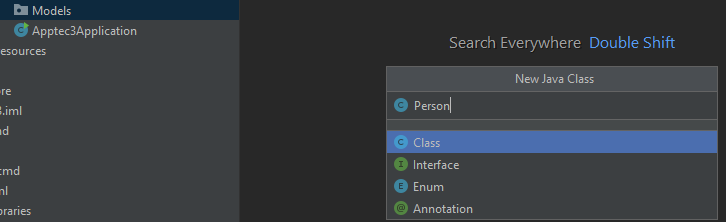


Vamos hacer click derecho y crear los Package’s necesarios para nuestro proyecto (directorios).

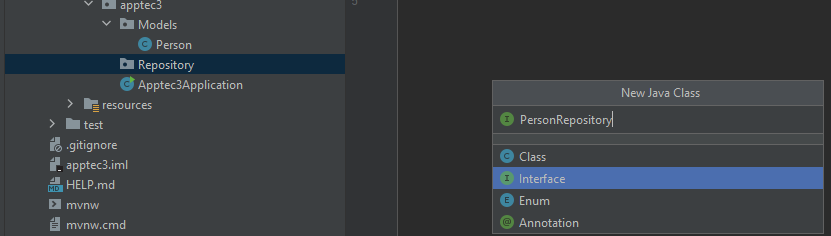


Crearemos un Package llamado Model y dentro del una clase de java ( Java Class) con el nombre de Person que será la que utilizaremos en este ejemplo.

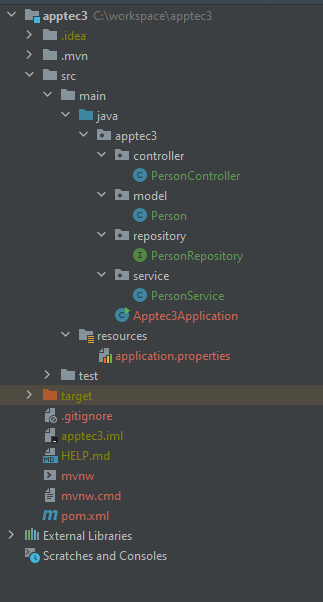




Luego crearemos otro Package con el nombre Repository y dentro del una interface llamada PersonRepository, aclaración para poder crear la interface hacemos clic derecho sobre el directorio (package) seleccionamos new -🡪 Java Class y en el menú desplegable 🡪Interface como se muestra en la siguiente imagen.

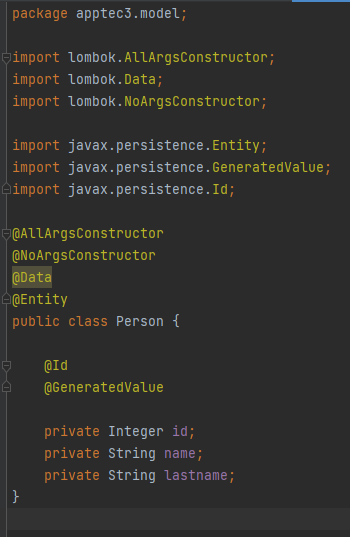


Nuestro proyecto deberá quedar con la siguiente estructura, con la excepción del Repository que será una interface, todas las demás serán clases de java.



**Models**

Dentro de este package vamos a crear los modelos de nuestro proyecto, en este ejemplo tenemos creado uno llamado Person, y para hacerlo de manera sencilla se vera de la siguiente forma:



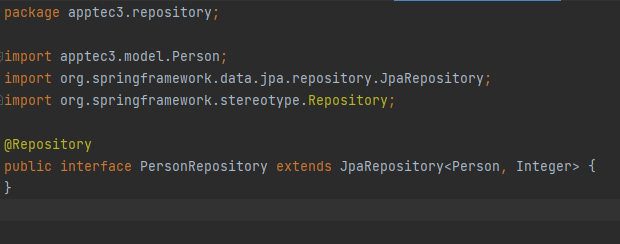
Analicemos los argumentos del Modelo Person:

Se observa la notación lombok, @AllArgsContrtuctor, @NoArgsConstructor, @Data. Seguido de la persistencia de datos (Argumentos para base de datos) @Entity, @Id, @GeneratedValue. Estos argumentos nos creara de manera automática la estructura de un modelo métodos getters and setters y el constructor de dicha clase, y los argumentos de persistencia de datos @Entity mapeara y creara la tabla en nuestra BD, en todos los ejemplos recomiendo siempre ir a la documentación oficial para analizar con más detalle su uso y funcionamiento.

**Repository**

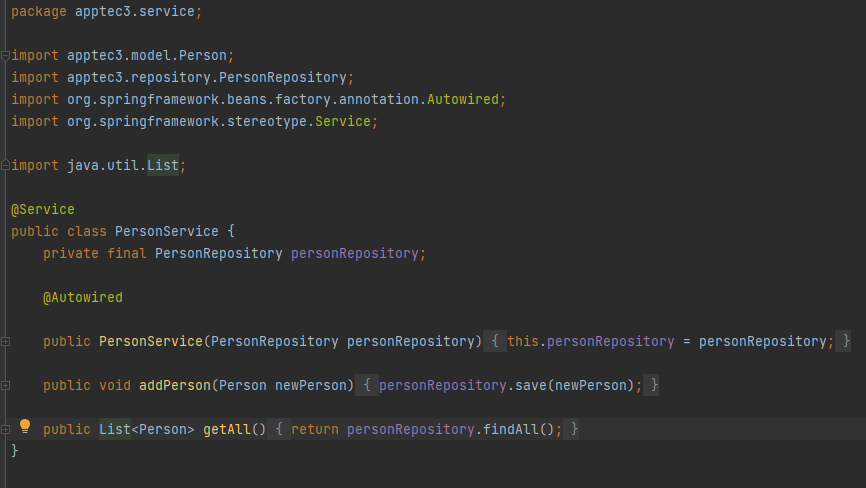
El concepto de **Repository** como clase que se encarga de gestionar todas las operaciones de persistencia contra una tabla de la base de datos es hace ya mucho tiempo un clásico. La mayor parte de los lenguajes de programación soportan el uso de clases Genéricas. Nuestro repositorio extiende (extends) o hereda de la clase JpaRepository y le pasamos la clase y el tipo como argumentos <Person, Integer>

**repository**: Es el package que contiene las interfaces que extienden de JPA para que estas clases se conecten a la base de datos. Estas gestionan información ya sea de buscar, borrar, actualizar o crear un registro en la base de datos.

****

**Service**

@**Service** : Este estereotipo se encarga de gestionar las operaciones de negocio más importantes a nivel de la aplicación y aglutina llamadas a varios repositorios de forma simultánea. Su tarea fundamental es la de agregador.

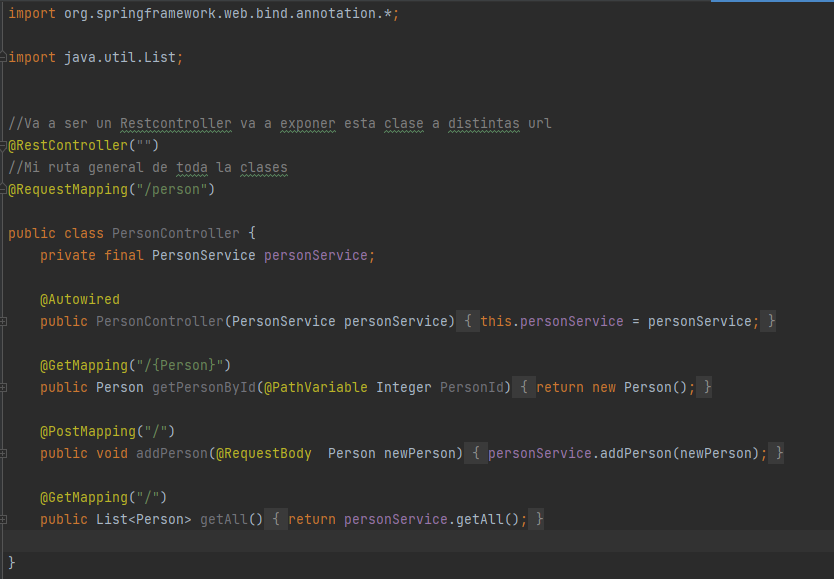
****

**Controller**

Un **Controller** es una clase que es responsable de preparar los datos que serán mostrados por la vista así como seleccionar la vista correcta a mostrar.

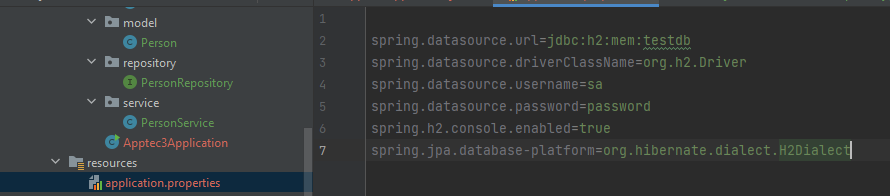
La **anotación** **@RestController** se usa para indicar **que** la clase es un controlador Spring. Esta **anotación** se puede utilizar para identificar controladores para Spring MVC o Spring WebFlux.

**@RequestMapping** Anotación que se encarga de relacionar un método con una petición http. El uso de anotaciones no implica que únicamente tengamos un único controlador sino que nos permite agrupar un conjunto de urls que estén asociadas a nivel de negocio en un controlador especifico.

****

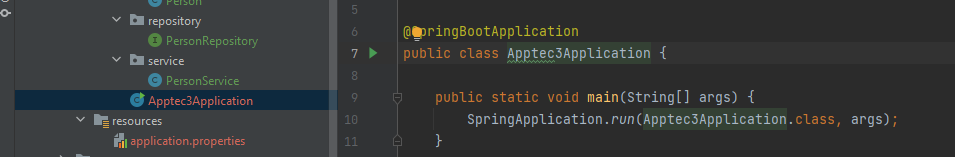
**Configurando nuestra base de datos H2**

En la carpeta (package) resorces vamos a encontrar un archivo llamado application.properties ahí vamos a editarlo como se observa en la imagen podemos cambiar los datos para personalizar nuestra conexión yo por defecto voy a dejarlo tal cual esta.

****

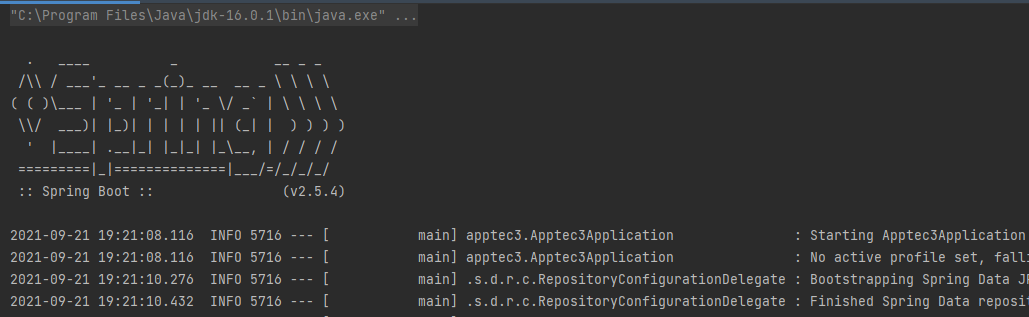
**Corriendo nuestra aplicación**

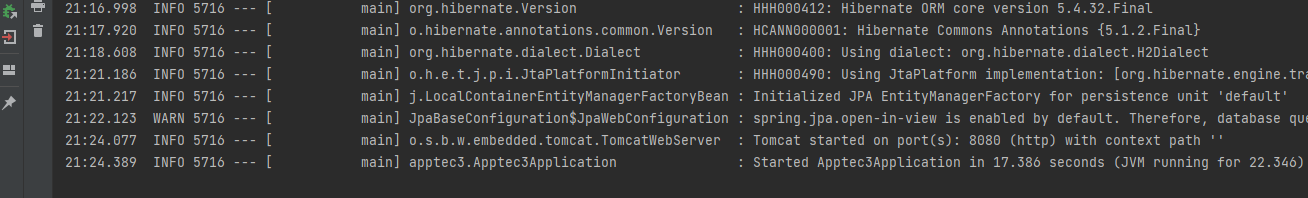
Al finalizar vamos a ejecutar nuestra aplicación para eso vamos a hacer clic en la flechita de color verde en la barra superior debemos estar posicionados en el archivo app que es el que contiene nuestro método main como se observa en la imagen





Y si todo salió bien nuestra aplicación debería verse como en la siguiente imagen:

****

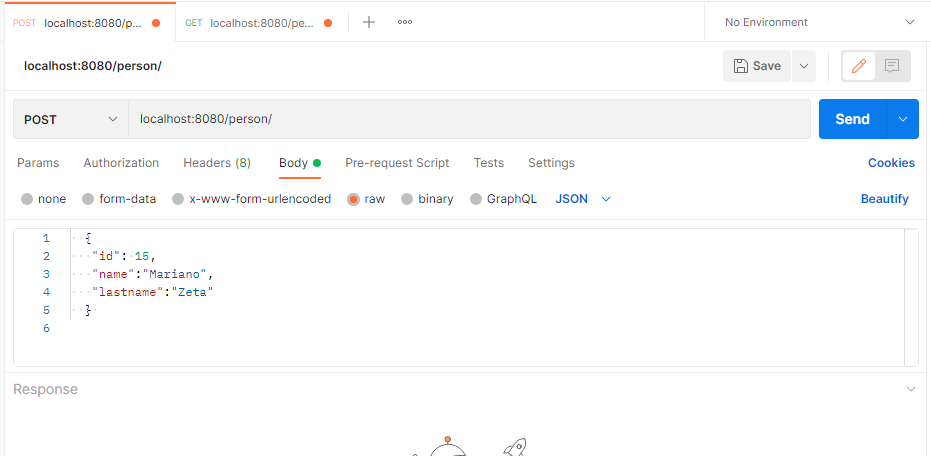
****

**Probando nuestra aplicación**

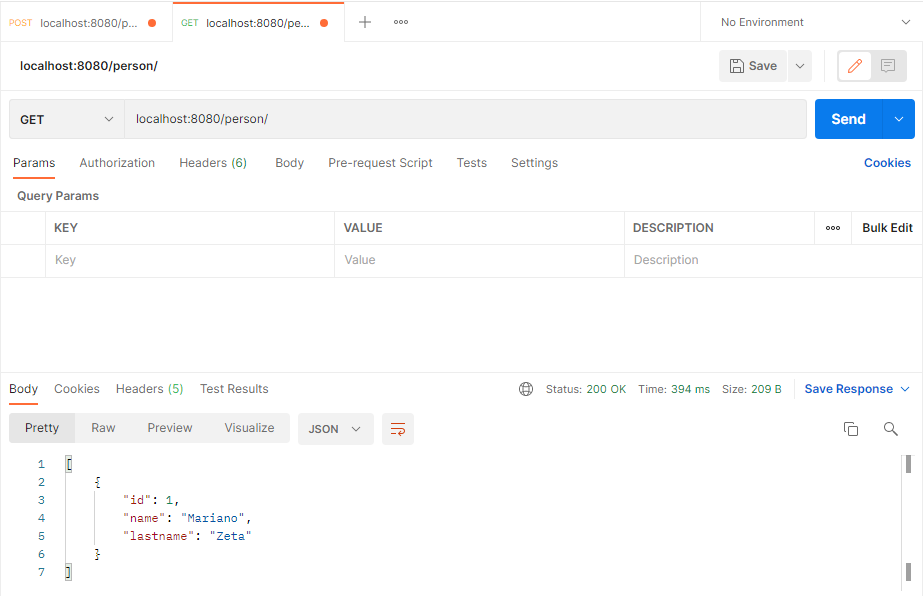
Dado que nuestra aplicación no tiene un front-end un entorno visual para poder probar, vamos a utilizar una herramienta de prueba llamada Postman. Con ella vamos a poder pegarle a nuestros endpoints.

Nuestros endopoints son las url (direcciones web) que exponemos para poder consumir un recurso de nuestro servicio Rest, si vamos a los controladores en este ejemplo tenemos uno llamado PersonController ahí dentro están definidas nuestras url.

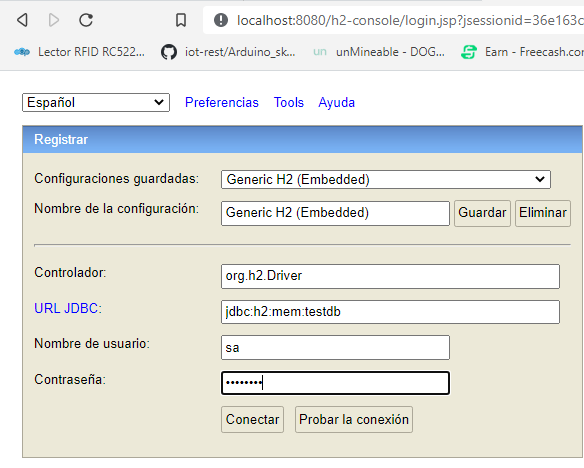
Sprinboot corre de manera local en nuestro equipo utilizando el servicio Tomcat del servidor apache y nos da como dirección localhost:8080, a eso le ponemos la dirección de nuestro endpoint /person y ya sabemos que podemos enviar peticiones get y post, en principio vamos a enviar una petición POST para poder agregar una persona a nuestra aplicación y lo hacemos usando un formato llamado JSON como se observa en la imagen siguiente:

****

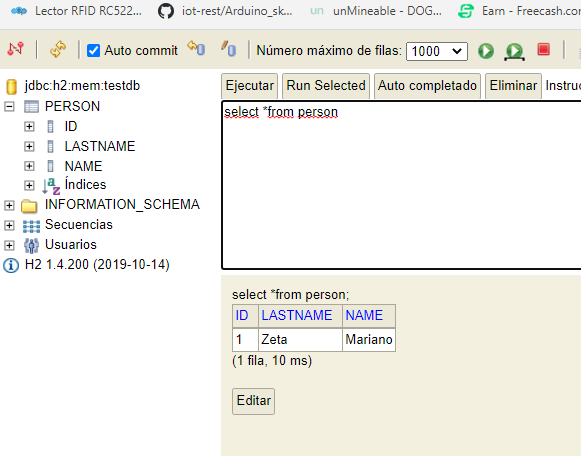
Si todo salió bien volvemos a pegarle a la misma url pero cambiando la petición POST por GET y el resultado será el dato que hemos agregado a nuestra BD de datos H2 que cargamos en memoria con sprinboot.

****

Para una mayor comprobación podemos entrar a la siguiente url con nuestra aplicación corriendo localhost:8080/h2-console y podremos acceder a nuestra base de datos como se observa en la imagen:

****

Y si ejecutamos una consulta SQL podremos ver en el resultado que el dato enviado en formato JSON fue insertado correctamente.

****

**Requisitos necesarios para este proyecto**

JDK – Java Developer Kit

[https://www.oracle.com/java/technologies/downloads/#jdk17-windows](https://www.oracle.com/java/technologies/downloads/" \l "jdk17-windows)

Maven repositorio

<https://maven.apache.org/download.cgi>