*## Spring boot*Permite desarrollar y arrancar de forma muy rápida aplicaciones basadas en Spring.  
  
*## Desarrollo del proyecto*En primer lugar, creamos un proyecto maven. Para indicar que queremos utilizar Spring Web con Spring Boot; en el archivo pom.xml las dependencias del programa deben de ser:  
~~~  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-mail</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>   
 <dependency>  
 <groupId>com.h2database</groupId>  
 <artifactId>h2</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 <!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt -->  
 <dependency>  
 <groupId>io.jsonwebtoken</groupId>  
 <artifactId>jjwt</artifactId>  
 <version>0.9.0</version>  
 </dependency>  
 <!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-security -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
 <version>2.3.5.RELEASE</version>  
 </dependency>  
 </dependencies>  
~~~  
  
*#Comandos Maven*| Maven Command | Descripción |   
|----------|:-------------:|  
| mvn --versión | Imprime la versión de Maven que está ejecutando. |  
| nvn clean | Borra el targetdirectorio en el que Maven normalmente construye su proyecto. |  
| mvn package| Genera el proyecto y empaqueta el archivo JAR resultante en el targetdirectorio |  
| mvn package -Dmaven.test.skip=true | Compila el proyecto y empaqueta el archivo JAR resultante en el targetdirectorio, sin ejecutar las pruebas unitarias durante la compilación.|  
|mvn clean package|Borra el targetdirectorio y compila el proyecto y empaqueta el archivo JAR resultante en el targetdirectorio.|  
|mvn clean package -Dmaven.test.skip=true|Borra el targetdirectorio y compila el proyecto y empaqueta el archivo JAR resultante en el targetdirectorio, sin ejecutar las pruebas unitarias durante la compilación|  
|mvn verify| Ejecuta todas las pruebas de integración que se encuentran en el proyecto|  
|mvn clean verify|Limpia el directorio de destino y ejecuta todas las pruebas de integración que se encuentran en el proyecto.|  
|mvn install|Crea el proyecto descrito por su archivo POM de Maven e instala el artefacto resultante (JAR) en su repositorio local de Maven|  
|mvn install -Dmaven.test.skip=true| Compila el proyecto descrito por su archivo POM de Maven sin ejecutar pruebas unitarias e instala el artefacto resultante (JAR) en su repositorio local de Maven|  
|mvn clean install|Borra el targetdirectorio y crea el proyecto descrito por su archivo POM de Maven e instala el artefacto resultante (JAR) en su repositorio local de Maven|  
|mvn clean install -Dmaven.test.skip=true|Borra el targetdirectorio y crea el proyecto descrito por su archivo Maven POM sin ejecutar pruebas unitarias, e instala el artefacto resultante (JAR) en su repositorio local de Maven|  
|mvn dependency:tree|Imprime el árbol de dependencias para su proyecto, basado en las dependencias configuradas en el archivo pom.xml.|  
|mvn dependency:tree -Dverbose|Imprime el árbol de dependencias para su proyecto, basado en las dependencias configuradas en el archivo pom.xml. Incluye dependencias transitivas repetidas.|  
|mvn dependency:tree -Dincludes=com.fasterxml.jackson.core|Imprime las dependencias de su proyecto que dependen del artefacto com.fasterxml.jackson.core.|  
|mvn dependency:tree -Dverbose -Dincludes=com.fasterxml.jackson.core|Imprime las dependencias de su proyecto que dependen del artefacto com.fasterxml.jackson.core. Incluye dependencias transitivas repetidas.|  
|mvn dependency:build-classpath|Imprime la ruta de clase necesaria para ejecutar su proyecto (aplicación) según las dependencias configuradas en el archivo pom.xml.|  
  
  
*# Creación SSL*En el terminal colocar `keytool -genkeypair -alias tomcat -keyalg RSA -keysize 2048 -storetype PKCS12 -keystore keystore.p12 -validity 3650 -storepass password`  
, donde:  
- genkeypair: genera un par de claves;  
- alias: el nombre de alias del elemento que estamos generando;  
- keyalg: el algoritmo criptográfico para generar el par de claves;  
- keysize: el tamaño de la clave. Hemos utilizado 2048 bits, pero 4096 sería una mejor opción para la producción;  
- storetype: el tipo de almacén de claves;  
- keystore: el nombre del almacén de claves;  
- validity: número de días de validez;  
- storepass: una contraseña para el almacén de claves.  
> Para verificar el almacen de claves `keytool -list -v -keystore keystore.jks` y escribir la contraseña que se establecio'  
  
*# Creación de una API Rest generica  
  
## Anotaciones de los archivos:*- @Autowired: A través de esta anotación Spring será capaz de llevar a cabo la inyección de dependencias sobre el atributo marcado. En este caso, estamos inyectando la capa de servicio, y por eso no tenemos que instanciarla.  
- @RequestMapping: Con esta anotación especificamos la ruta desde la que escuchará el servicio, y qué método le corresponde.  
- @ResponseBody: Con ella definimos lo que será el cuerpo de la respuesta del servicio.  
- @PathVariable: Sirve para indicar con qué variable de la url se relaciona el parámetro sobre el que se esté usando la anotación  
- La etiqueta @Configuration, indica que la clase en la que se encuentra contiene la configuración principal del proyecto.  
- La anotación @EnableAutoConfiguration indica que se aplicará la configuración automática del starter que hemos utilizado. Solo debe añadirse en un sitio, y es muy frecuente situarla en la clase main  
- La anotación @EnableAutoConfiguration indica que se aplicará la configuración automática del starter que hemos utilizado. Solo debe añadirse en un sitio, y es muy frecuente situarla en la clase main  
 > Para no llenar nuestra clase de anotaciones, podemos sustituir las etiquetas @Configuration, @EnableAutoConfiguration y @ComponentScan por @SpringBootApplication, que engloba al resto.   
- Podemos también utilizar la etiqueta @RestController en lugar de @Controller, que sustituye al uso de @Controller + @ResponseBody,  
- Los componentes de servicio son el archivo de clase que contiene la anotación @Service. Estos archivos de clase se utilizan para escribir lógica empresarial en una capa diferente, separada del archivo de clase @RestController.  
   
 *## Funciones:*1. Crud de datos usando MySQL  
 >  
 *#### Creación del proyecto   
 #### Creación del CRUD inicial* 1.1 Creación de los packages: Controller, Exception, Model, Repository y Service  
 \* Controller: Tendrá las rutas y método HTTP por los cuales se va a acceder a las peticiones determinadas  
 \* Exception: Mostrará mensajes al no encontrar una petición determinada  
 \* Model: Tendrá los archivos que contienen la estructura de datos de la DB junto con los metodos SET y GET para   
 modificar los mismos  
 \* Repository: Tendrá el archivo mediante el cual se tendrá acceso a los datos establecidos por el modelo de dats creado  
 \* Service: El cual tiene los archivos que generan la lógica de las consultas HTTP  
 >  
 El proyecto se crea en la secuencia de:  
   
 \* Modelo → Java Class  
 \* Repositorio → Java Interface  
 \* ResourceNotFound → Java Class  
 \* DataModelService → Java Interface  
 \* Controller → Java Class  
 \* DataModelServiceImpl → Java Class   
   
 >Dentro del ResourceNotFound se encuentra la constante `private static final long serialVersionUID = 1L;`, la cual   
 significa: El serialVersionUID es un identificador de versión universal para una Serializableclase.   
 La deserialización usa este número para garantizar que una clase cargada corresponda exactamente a un objeto   
 serializado.  
 >Dentro del archivo DataModelServiceImpl, implementa el DataModelService con lo cual se  
   
  
*# Ejecución del programa*Generar los permisos de ejecución para los archivos .sh  
``chmod +x [nombre del archivo].sh``  
Ejuctar los archivos en el orden:  
\* ./compile.sh  
 ![compile.sh](Doc/Imagenes/compile.png)  
 Si al momento de ejecutar el script aparce Build Success, la compilación a sido un exito, de lo contrario revisar los mensajes que se muestran en consola.  
\* ./run.sh  
 ![run.sh](Doc/Imagenes/Run.png)   
 Si al momento de ejecutar el script aparce :::RUN:::, la ejecución del programa ha sido un exito, de lo contrario revisar los mensajes que se muestran en consola.  
   
\* Ó también puedes usar el archivo `./default.sh`, el cual ejecutará compile.sh y run.sh de la forma `./compile.sh && ./run.sh` que es otra forma de ejecutar los dos scripts a la vez.  
  
  
*# Autenticación y uso de Json Web Token*\* UsuarioDao, esta clase de entidad se utiliza para la información del usuario.  
\* UsuarioDto, es responsable de obtener los valores del usuario y pasarlos a la capa DAO para ser insertados en la base de datos.  
\* UsuarioRepository, este repositorio nos permite acceder a la información almacenada en la base de datos utilizando JPA. Hemos utilizado CrudRepository. CrudRepository proporciona principalmente funciones CRUD, mientras que JpaRepository proporciona algunos métodos relacionados con JPA, como vaciar el contexto persistente, eliminar registros en un lote.