Descripción general de la API

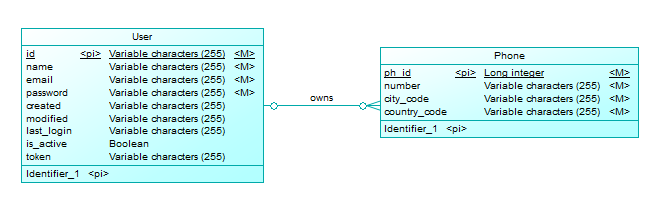
Los endpoints se realizaron teniendo en cuenta el ejemplo puesto en el PDF, por lo cual algunos nombres se dejaron tal cual sin importar errores de ortografía/estilo por si tienen datos de prueba listos con esas características. Por ejemplo, el campo “contrycode” debería llamarse “countryCode” pero se conservó así.

La API utiliza JWT para dar seguridad a los endpoints, sin embargo, se dejaron tres que tienen acceso libre para poder realizar las pruebas:

* /**authenticate**: permite a los usuarios existentes autenticarse y obtener un token para poder acceder a otros endpoints.
* /**registerUser**: es usado para crear nuevos usuarios. Es el primero que debe accederse ya que la API quedó configurada para generar las tablas necesarias para su funcionamiento desde el arranque, pero sin datos.
* /**h2**-**console**: es el que permite acceder a la consola de H2 para revisar los datos, tablas y demás elementos existentes en la base de datos.

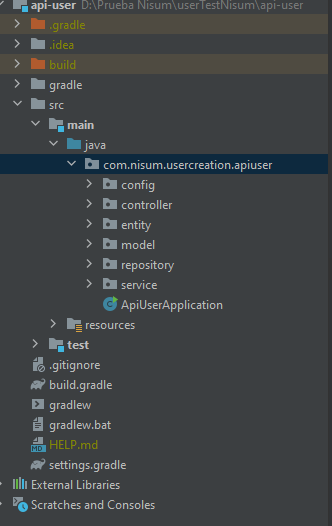
Cualquier otro endpoint debe venir acompañado en su header de un adecuado token que se obtiene con el endpoint /authenticate.

Las tablas creadas son 2, “**User**” y “**Phone**”, cada uno con sus datos correspondientes y adicional a esto existe una relación de uno a muchos, un usuario puede tener varios teléfonos y al momento de crear un usuario, si se envía uno o más objetos teléfono, estos serán creados de forma automática en la base de datos, con lo cual no es necesario crear un teléfono para crear un usuario o viceversa, aunque es posible.



Note que los tipos de datos para los atributos de “**Phone**” son **Varchar**, esto también debido al ejemplo en el PDF, de otra forma serían números.

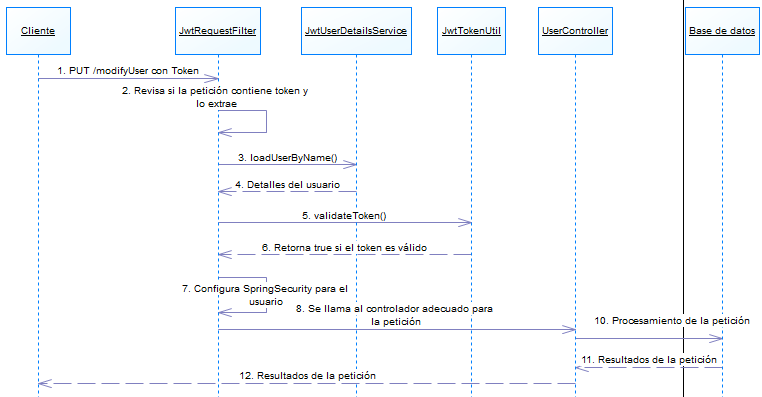
Distribución de paquetes de la aplicación



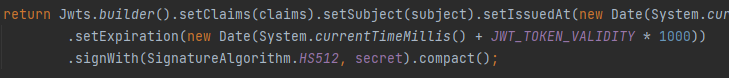
La aplicación está construida con los siguientes paquetes:

* **Config**: contiene clases de apoyo y archivos de configuración necesarios para el uso de JWT.
* **Controller**: contiene los controladores que definen los endpoints a los que se puede acceder en la API.
* **Entity**: contiene clases que mapean las tablas existentes en la base de datos, en este caso la entidad **User** y la entidad **Phone**.
* **Model**: contiene clases que sirven de modelo para las peticiones y respuestas de la funcionalidad de JWT pero que no corresponden a tablas en la base de datos.
* **Repository**: en este paquete se encuentran interfaces que heredan de **JpaRepository**, permitiendo así tener funcionalidades que facilitan operaciones de CRUD sobre las tablas de la base de datos.
* **Service**: este paquete está creado para contener servicios que se fueran a usar en la API, aquellos con la anotación @**Service** de Springboot. Para este caso sólo se encuentra un servicio relacionado con la funcionalidad de JWT.

Proceso de autenticación



El proceso de autenticación sigue este diagrama de secuencia. Este diagrama muestra el caso en el que se envía una petición con un token válido, de no ser así el proceso se cortaría en el paso 6 evitando que el usuario acceda a recursos no permitidos. El tiempo de sesión, es decir, la validez del token es definida en JwtTokenUtil y es definida en milisegundos a partir de la hora actual:



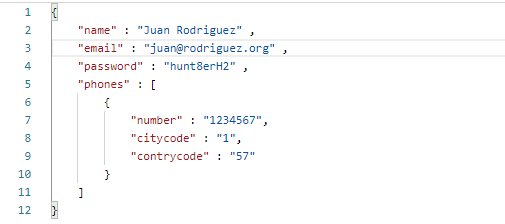
Adicional a la seguridad de JWT, se adicionó una funcionalidad para no guardar los password de los usuarios directamente en la base de datos. Estos password son cifrados utilizando el **PasswordEncoder** de Springboot.

Endpoints

**URL**: <http://localhost:8080/registerUser>

**Método**: POST

**Parámetros**: un objeto JSON que contenga los datos del usuario a crear como, por ejemplo:



**Respuesta**: en caso de ser exitosa la operación se retorna un JSON conteniendo el registro creado en la base de datos, en caso contrario, un JSON con un mensaje de error explicando el por qué la operación falló, ejemplo:

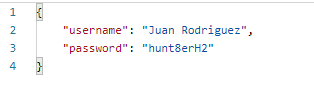


**Restringido**: No

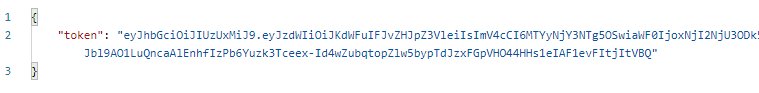
**URL**: <http://localhost:8080/authenticate>

**Método**: POST

**Parámetros**: un objeto JSON que contenga los datos del usuario que se quiere autenticar como, por ejemplo:



**Respuesta**: un JSON conteniendo un nuevo token válido para el usuario, ejemplo:



**Restringido**: No

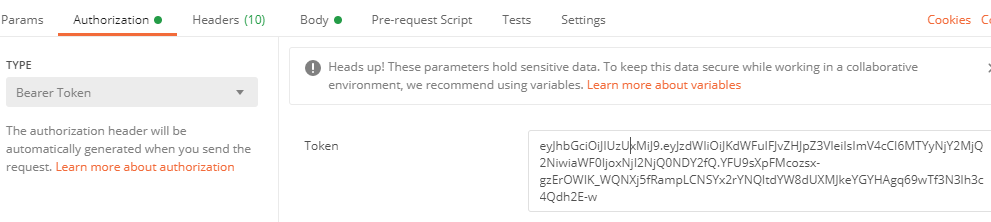
**URL**: <http://localhost:8080/modifyUser>

**Método**: PUT

Parámetros: un objeto JSON conteniendo los cambios que se quieren aplicar (solamente se tienen en cuenta los atributos name, password, email y active). Ejemplo:



También se debe adicionar en los **header** un token válido obtenido con el endpoint **“/authenticate”** o **“/registerUser”,** ejemplo:



**Respuesta**: un JSON que contiene el objeto como quedó en la base de datos, ejemplo:

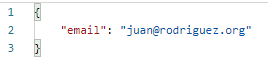


**Restringido**: Sí

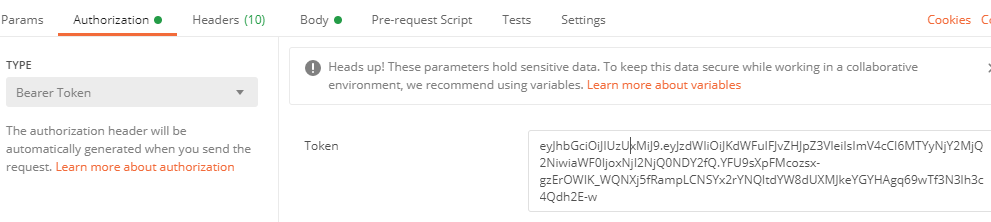
**URL**: <http://localhost:8080/findUserByEmail>

**Método**: GET

**Parámetros**: un JSON conteniendo el email del usuario a buscar, ejemplo:



También se debe adicionar en los **header** un token válido obtenido con el endpoint **“/authenticate”** o **“/registerUser”,** ejemplo:

**Respuesta**: un objeto JSON conteniendo el usuario encontrado, ejemplo:

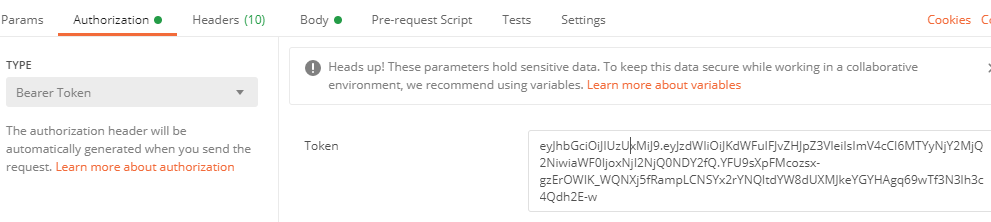


**Restringido**: Sí

**URL**: <http://localhost:8080/findAllUsers>

**Método**: GET

**Parámetros**: solamente un token válido en el header como en el anterior endpoint:



**Respuesta**: un JSON conteniendo un listado de objetos de la tabla USER, ejemplo:



**Restringido**: Sí