|  |
| --- |
| **SPRINT 2: Desarrollo del Backend** |

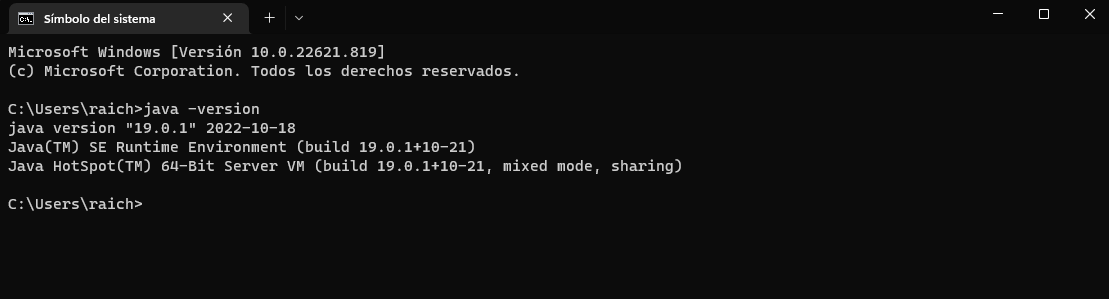
|  |  |
| --- | --- |
| **Identificación Proyecto** | |
| Nombre Proyecto: | Desarrollo de una aplicación web para el registro y  visualización de marcadores deportivos. |
| Número Equipo: | 117 |
| **Integrantes del equipo** | |
| Rol  (Líder-Desarrollador – Cliente) | Nombre |
| Desarrollador | Ray Andrés Támara Sáez |
|  |  |

|  |
| --- |
| **Evidencia construcción del Backend** |

Para la creación del API REST emplearemos Java + Spring Boot. Lo anterior se llevará acabo en el entorno de ***Visual Studio Code (VSC)***.

De manera que requeriremos el *Java Developer Kit (jdk),* que para este caso usaremos y verificaremos que este instalado en su versión 19.

<https://www.oracle.com/java/technologies/downloads/#jdk19-windows>



Para el correcto funcionamiento del VSC con nuestro objetivo de desarrollo en la tecnología deseada se necesitarán las siguientes extensiones:

*Extensión Pack for Java:*

<https://marketplace.visualstudio.com/items?itemName=vscjava.vscode-java-pack>

*Spring Boot Extension Pack:*

<https://marketplace.visualstudio.com/items?itemName=Pivotal.vscode-boot-dev-pack>

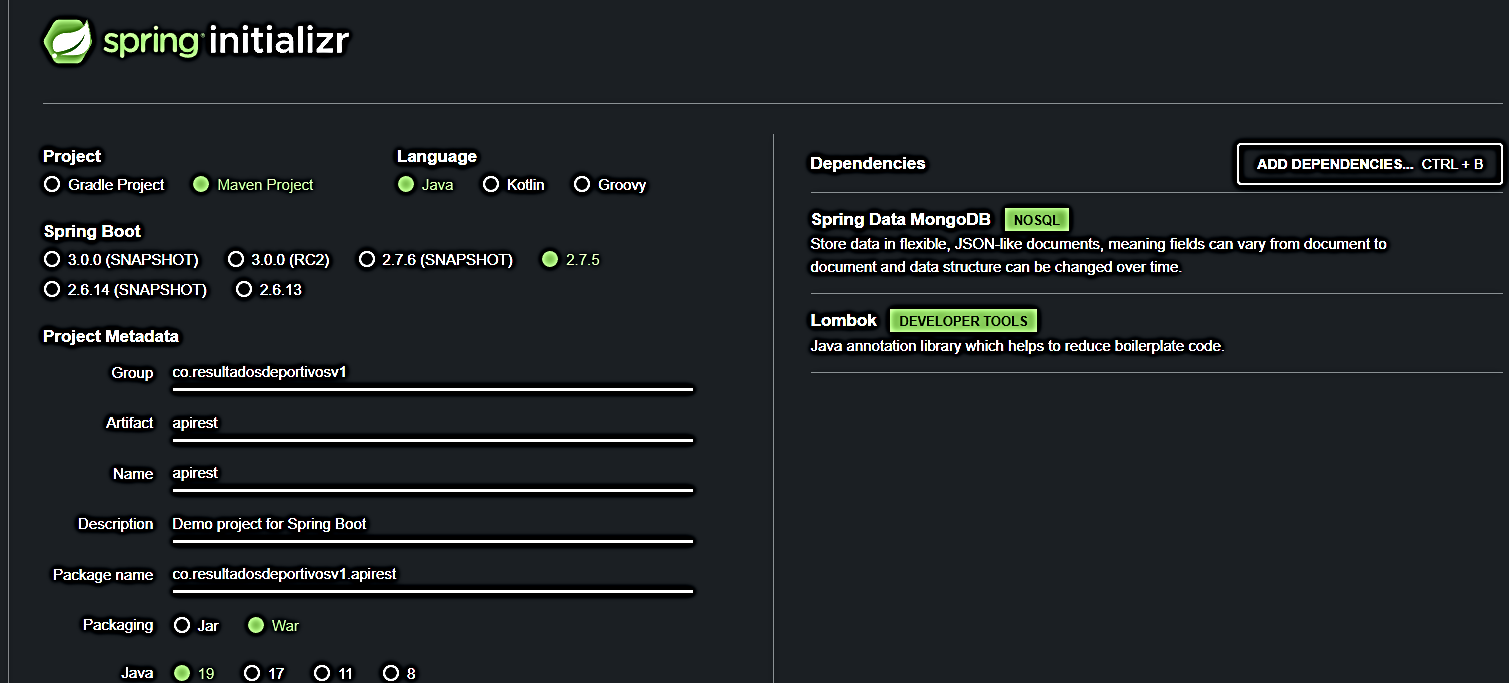
*Thunder Client:* (Para realizar las peticiones de cliente a nuestra API REST).

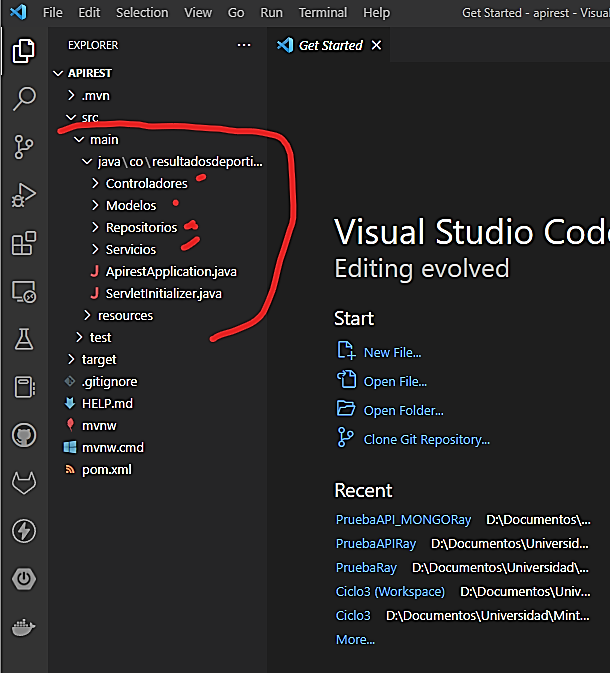
<https://marketplace.visualstudio.com/items?itemName=rangav.vscode-thunder-client>

*Mongo DB Community Server:* Cómo se mencionó en el Sprint 2 se empleará este sistema de base de datos no SQL basado en documentos pero que recibirá información estructurada.

<https://www.mongodb.com/try/download/community>

El proyecto base del API REST lo creamos utilizando Spring initializer ([Spring Initializr](https://start.spring.io/)) , en donde nos aseguraremos de crear un proyecto Maven en lenguaje Java en su versión 19, en la ultima versión estable de spring boot (2.7.5) con un empaquetado tipo WAR y con las dependencias *Spring Data MongoDB* y *Lombok*.

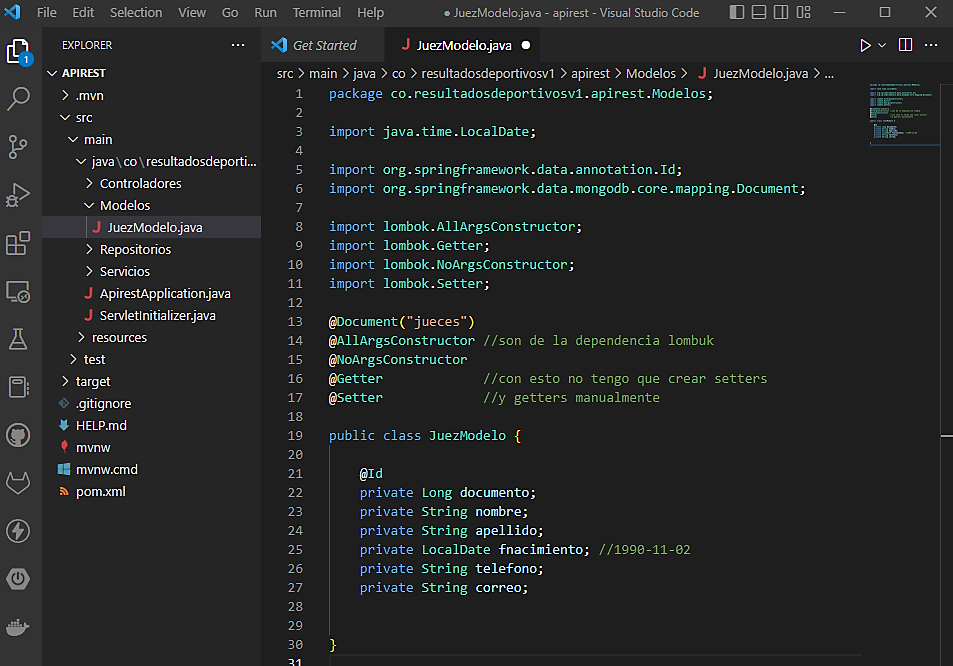




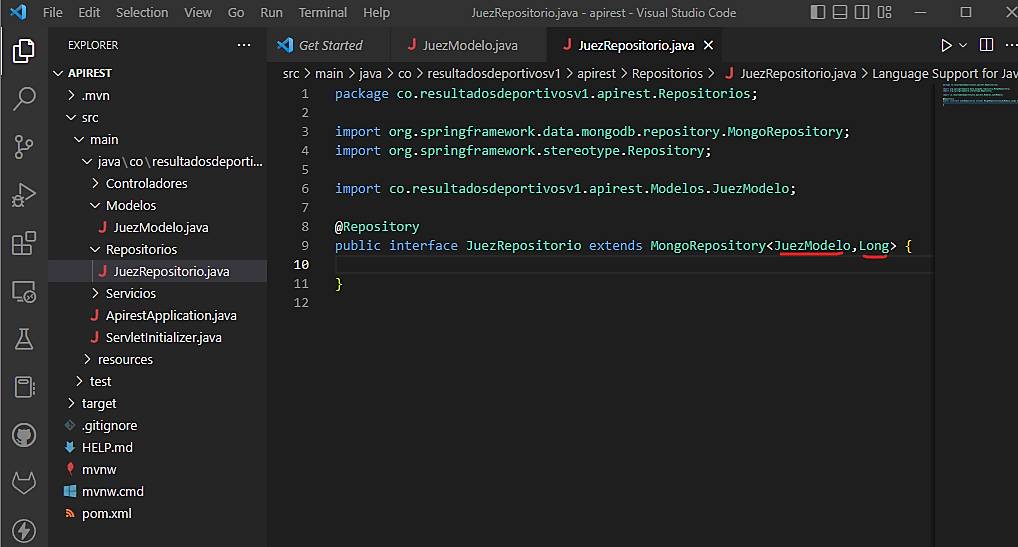
Generamos y descargamos el comprimido .zip de Spring initializer con el proyecto base del APIrest el cual descomprimimos en el lugar de preferencia.

Abrimos la carpeta de nuestro proyecto en *VSC,* y creamos las 4 carpetas correspondientes a las capas de nuestro backend (modelo, servicios, controlador, Repositorios). en la carpeta /APIrest/src/main/javaXXXX.

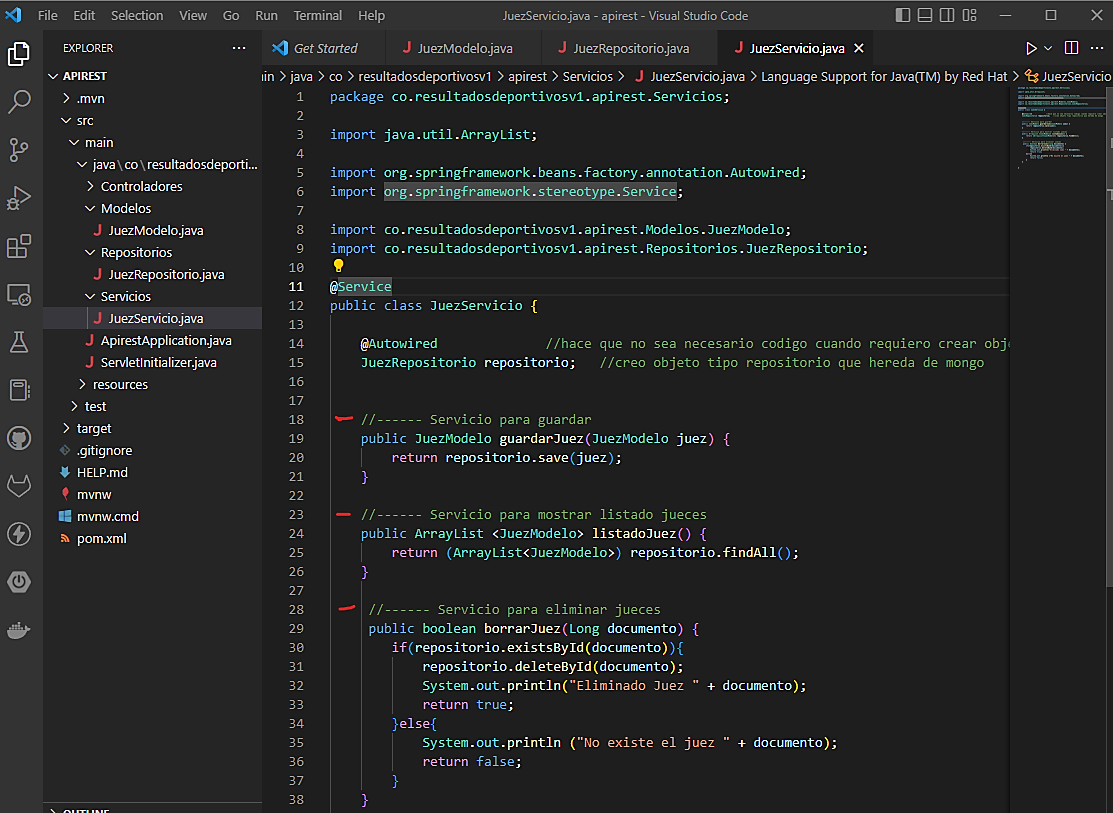
Creamos un *modelo juez* o persona habilitada para ingresar los resultados deportivos, se crea nueva archivo tipo Java en la carpeta Modelos y se crea el modelo como una clase pública con la información que consideramos conveniente.



Creamos el repositorio con los métodos del CRUD para el juez como una clase de tipo interfaz que hereda de un repositorio de mongo y tendrá como parámetros de entrada el *Modelo de tipo juez*, y el *id de tipo Long* (documento).



A continuación en la capa de servicios, declaramos y creamos los servicios de los que dispondrá el Modelo de Juez, creamos 3 servicios correspondientes al *guardado del juez* en la base de datos, *la eliminación del juez y* el listado de todos los jueces en la base de datos.



Finalmente procedemos a crear el controlador con cada uno de los endpoints que llamarán a los servicios de los que dispone el modelo juez.

Endpoint principal

<http://localhost:8080/juez/guardar>

Guardar Juez.

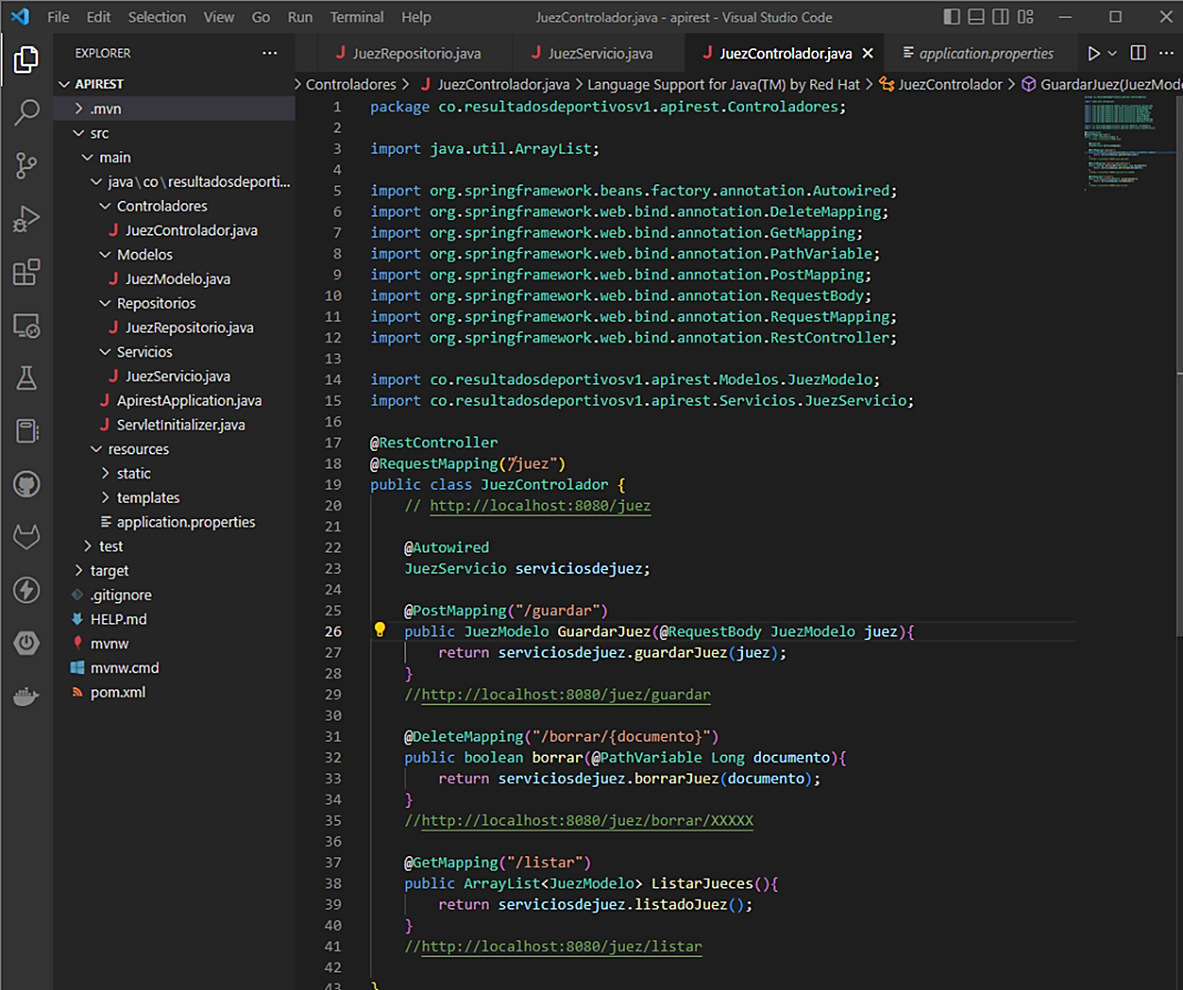
<http://localhost:8080/juez/guardar>

Borrar Juez.

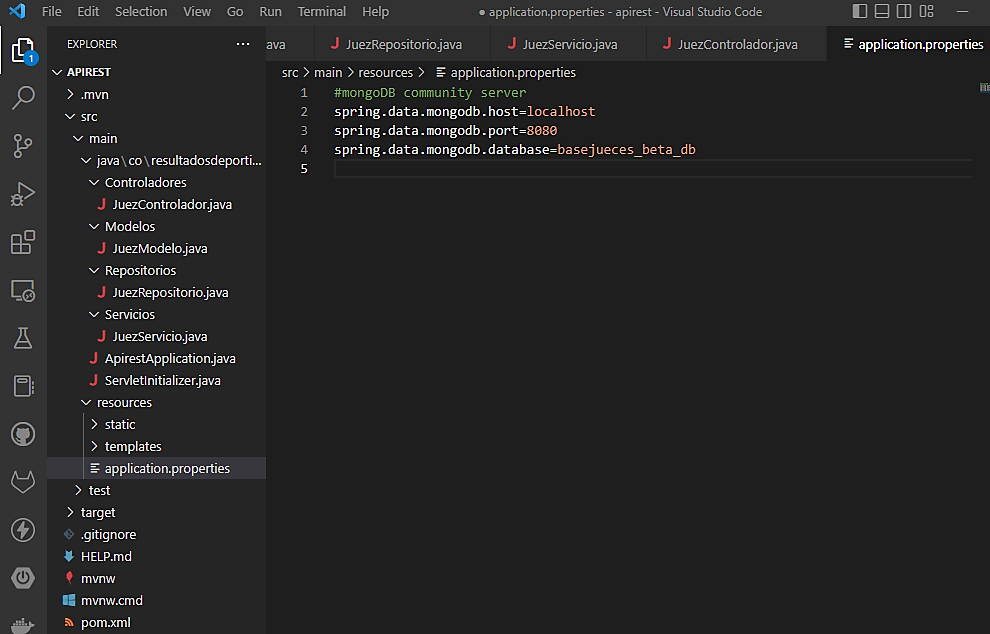
<http://localhost:8080/juez/borrar>/#documentoaborrar

Listar Jueces.

<http://localhost:8080/juez/listar>

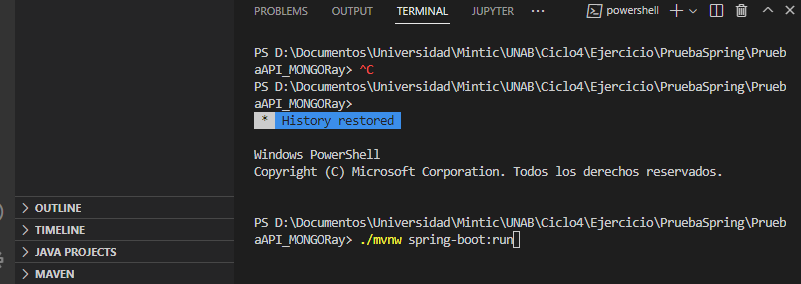


Finalizamos con la cadena de conexión a la base de datos.



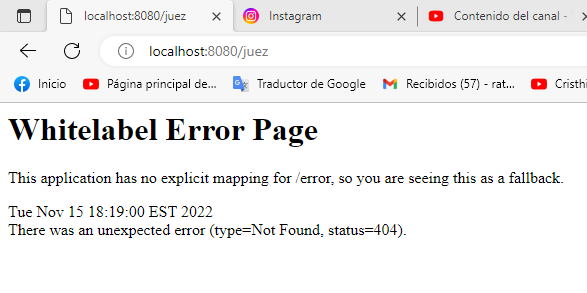
Ahora se debe inicializar el servidor local spring boot con el comando.

./mvnw spring-boot:run en la terminal.



Y verificamos en el navegador que el endpoint principal del backend está respondiendo con el navegador.

[http://localhost:8080/juez/](http://localhost:8080/juez/guardar)

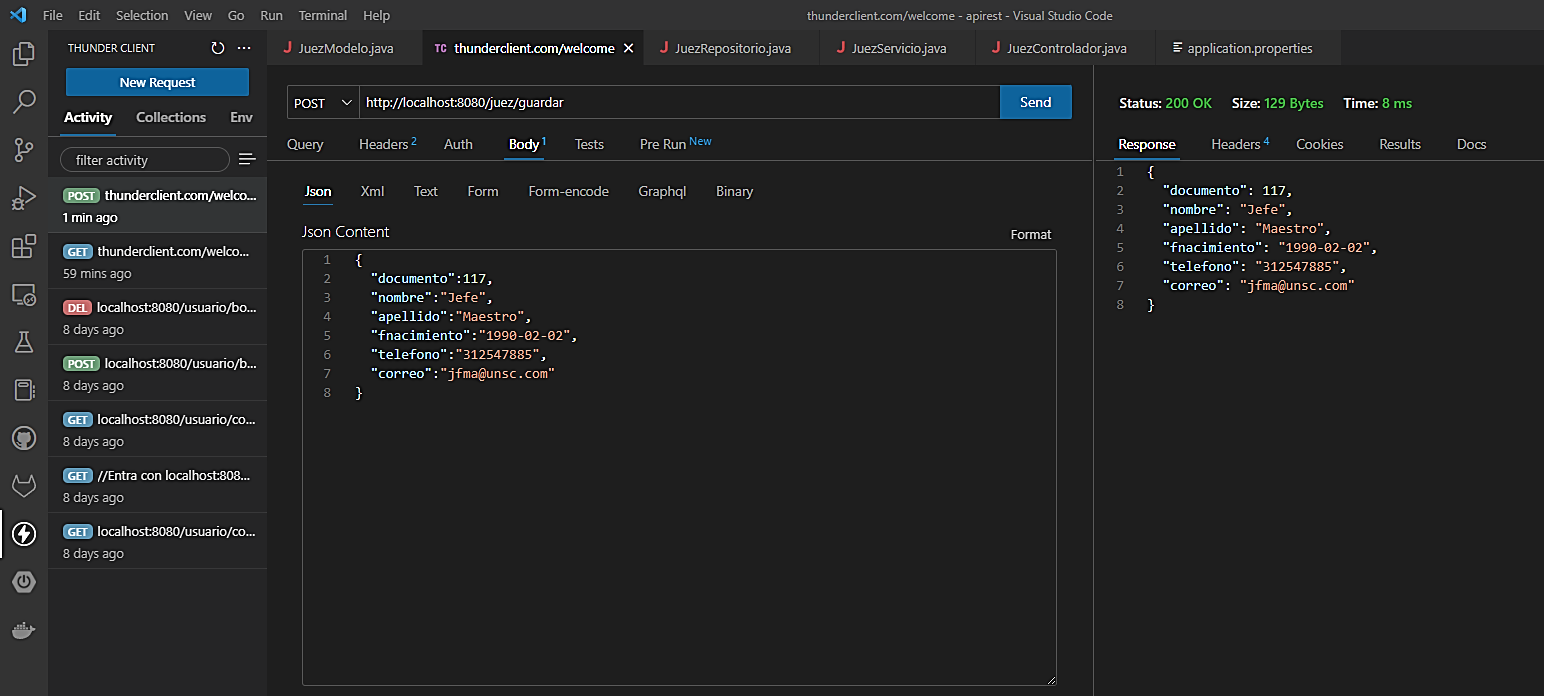


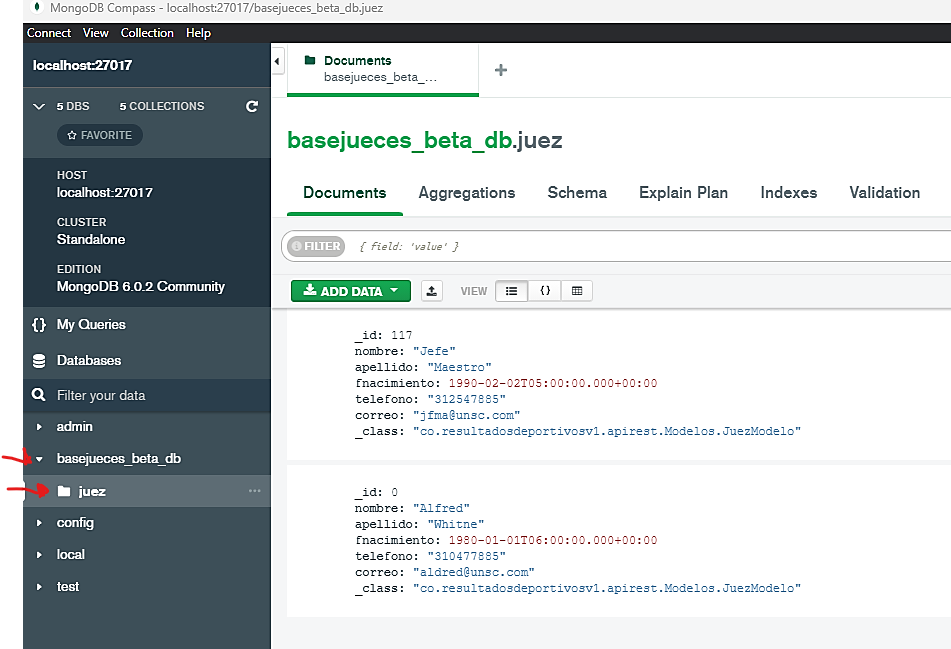
|  |
| --- |
| **Evidencias de los “endpoint” con el consumo de recursos del API REST** |

Como evidencia de los “endpoint” donde se visualice el consumo de recursos del API REST.

Haciendo uso de la extensión *ThunderClient* probaremos que el CRUD se este realizando de manera correcta en nuestro APIRest con los métodos **GET**, **POST** y **DELETE.**

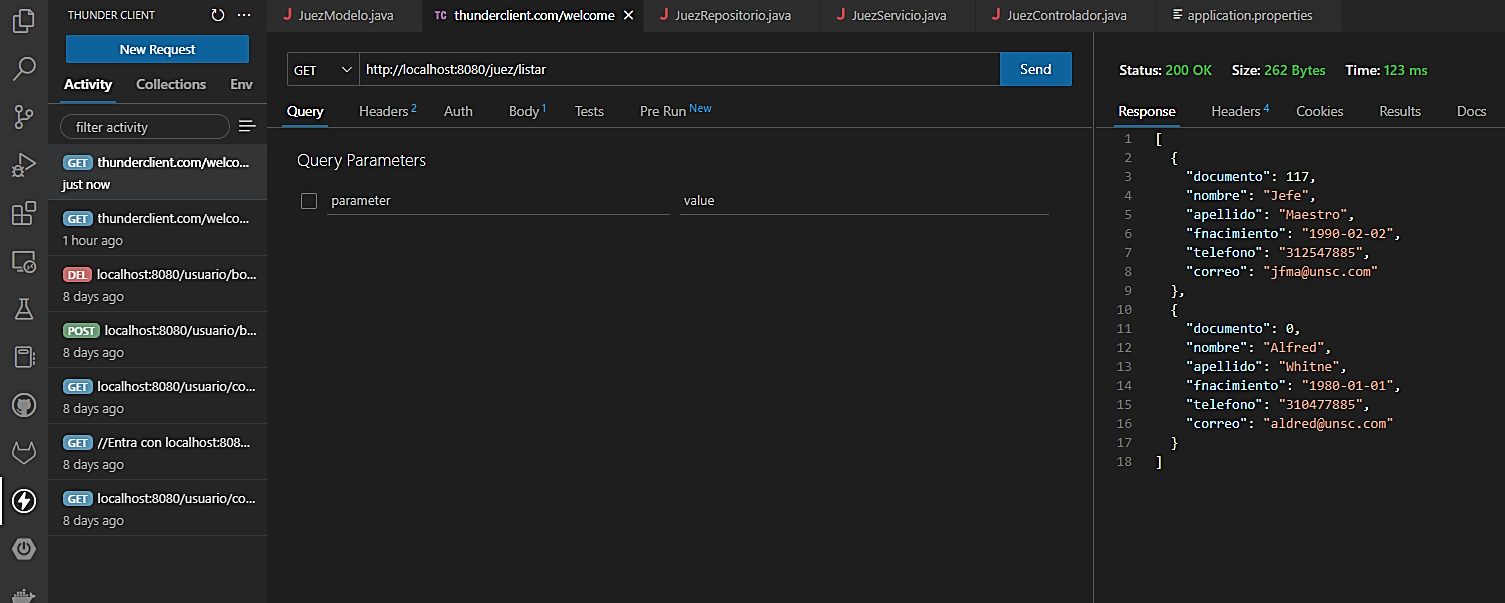
Creamos una **petición de tipo POST** con un body tipo JSON que será dirigida al servicio de guardado de jueces y que nos devuelve un **resultado 200** que indica la petición fue realizada con éxito por la APIRest y el servidor.



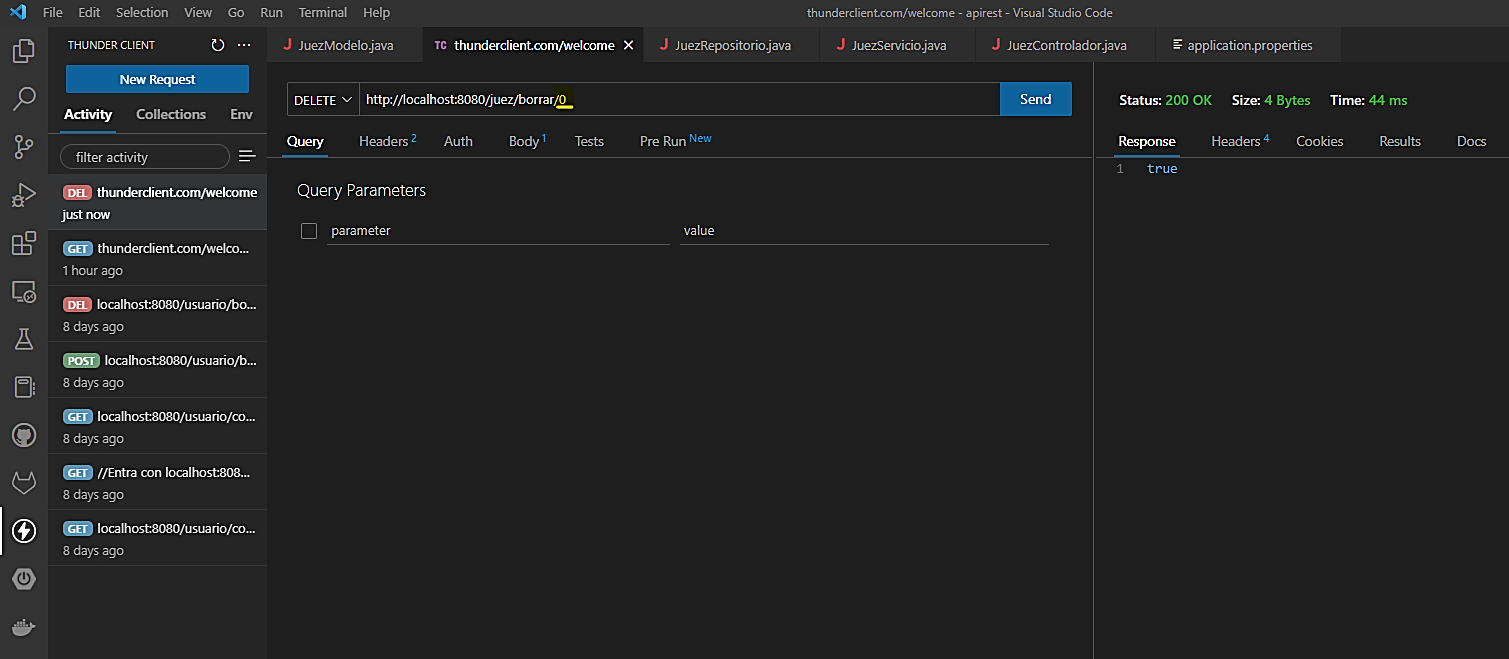


Adicional se verifica en la interfaz gráfica de ***Mongo Compass*** que **Spring-Boot** efectivamente realizo la creación de una base de datos llamada *basejueces\_beta\_db* y dentro de la misma se ha creado el *modelo juez* en donde estarán contenidos todos los jueces habilitados para ingresar al sistema y en dónde ya se encuentra con éxito la información contenida en el JSON de la petición previamente enviada a la APIRest.

Ahora creamos una petición de **tipo GET** que llame al endpoint con el servicio de listar y que debería listar los elementos contenidos en la base de datos. La petición se realiza con éxito con un código 200 y nos muestra los elementos de la base de datos como se mostró en Mongo Compass.



Para finalizar se realiza una petición de **tipo DELETE**, que llama al endpoint con el servicio de borrar juez y que debería eliminar el registro que tenga un *id* o en este caso *documento* “0”. El servidor devuelve un código 200 con éxito y un mensaje de TRUE.



Para verificar que el registro con documento “0” fue eliminado volvemos a realizar la petición con el endpoint al servicio de listado de jueces que corre con éxito y ahora sólo muestra un registro demostrando que el id ”0” fue eliminado.

