

Sistema CRUD con Spring Boot y MongoDB

Estructura Básica del proyecto CRUD

src/main/java/com.example.CRUD :

En esta Url contiene todo el código fuente Java del proyecto.

- **model:** Define las entidades que representan los datos (Clase Spartan).
- **repository:** Interfaces que permiten la conexión y operaciones sobre MongoDB.
- **Service:** Aquí se implementa la lógica como es (guardar, buscar, actualizar, eliminar).
- **rest:** Controladores REST Aquí estarán todos los endpoints de la API.
- **CrudApplication.java:** Aquí esta la aplicación Spring Boot.

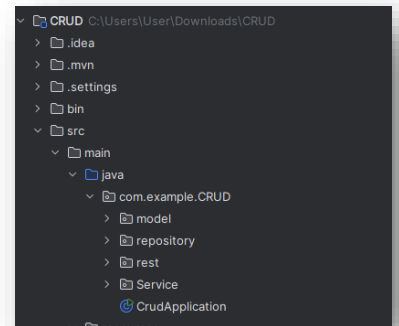
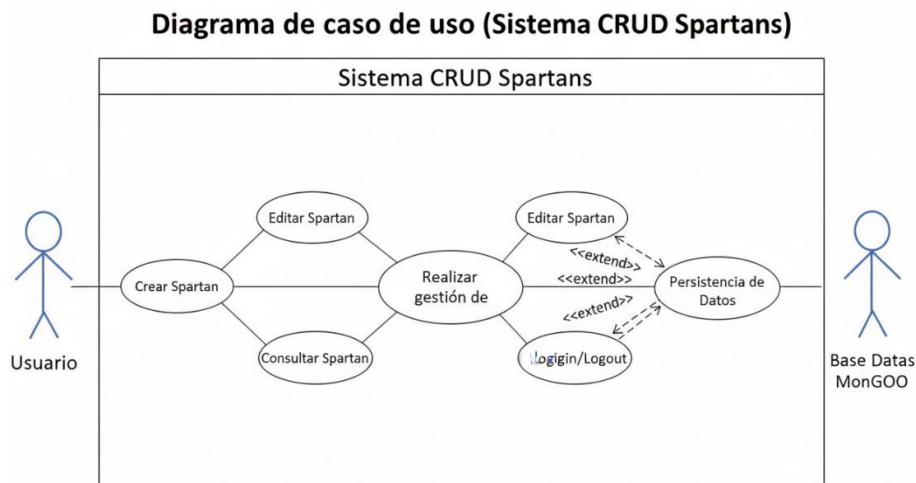


Diagrama de Caso de Uso



Actores

- **Usuario:** interactúa directamente con el sistema para gestionar la información del dato proporcionado por el usuario
- **MongoDB:** Es la base de datos que recibe toda la información y responde con los datos solicitados, a través del servicio REST.

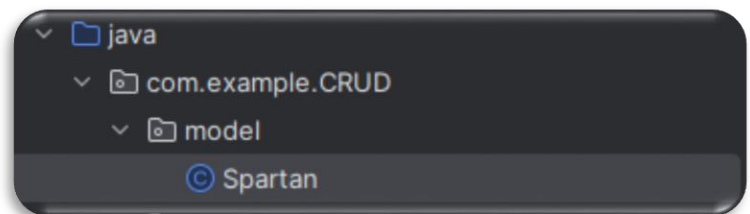
Casos de Uso

- **Crear** Este caso de uso permite al usuario agregar un nuevo Spartan a la base de datos.
- **Editar**: Permite al usuario modificar la información de un Spartan que ya existe.
- **Consultar**: El usuario puede buscar y ver los detalles de uno o varios Spartans.
- **Eliminar Spartan**: Este caso de uso le da la opción al usuario de borrar un Spartan del sistema.

Descripción de la Clase Spartan

Esta clase define cómo se organiza y almacena la información de un dato dentro de nuestra aplicación de gestión.

Este es un modelo que usaremos para representar cada dato en la base de datos MongoDB.



Anotaciones Clave

- `@Document(collection = "Spartan")`: Le indica a Spring Data que esta clase corresponde a la colección "Spartan" en MongoDB.
- `@Id`: Marca el campo id como el identificador único de cada documento.
- `@Indexed(unique = true)`: Garantiza que el campo gamertag sea único, evitando duplicados en la colección.

Atributos de la Clase

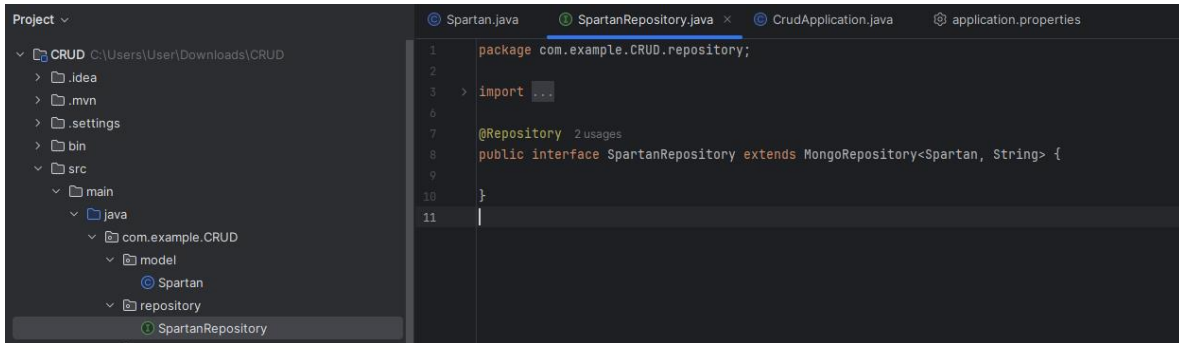
- `id`: Identificador único generado automáticamente por MongoDB.
- `nombreCodigo`: Nombre de operación o código
- `gamertag`: Nombre de usuario único
- `escuadra`: Escuadrón al que pertenece.
- `nivelExperiencia`: Valor numérico que indica la experiencia.
- `fechaAlistamiento`: Fecha y hora en que fue registrado en el sistema.

Constructores y Métodos

- **Constructor sin argumentos**: Inicializa `nivelExperiencia` en 1 y asigna automáticamente la fecha de alistamiento.
- **Getters y Setters**: Métodos para acceder y modificar los valores de los atributos de manera segura.

Descripción del Repository SpartanRepository

Este código define el repositorio que nos permite interactuar con la base de datos MongoDB para la entidad Spartan.



Descripción del Controller SpartanRest

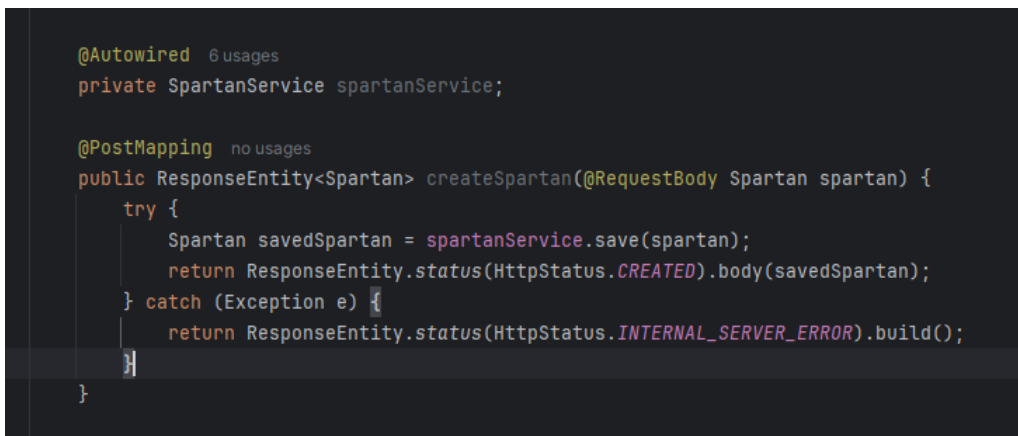
Esta clase es el controlador REST que expone los endpoints de la API para manejar operaciones sobre los Spartans. Aquí es donde las solicitudes HTTP del cliente son recibidas y procesadas.

Anotaciones Clave

- `@RestController` : Marca la clase como controlador REST.
- `@RequestMapping("/api/spartans")` Define la ruta base de todos los endpoints
- `@Autowired` : Inyecta automáticamente la instancia de `SpartanService`.

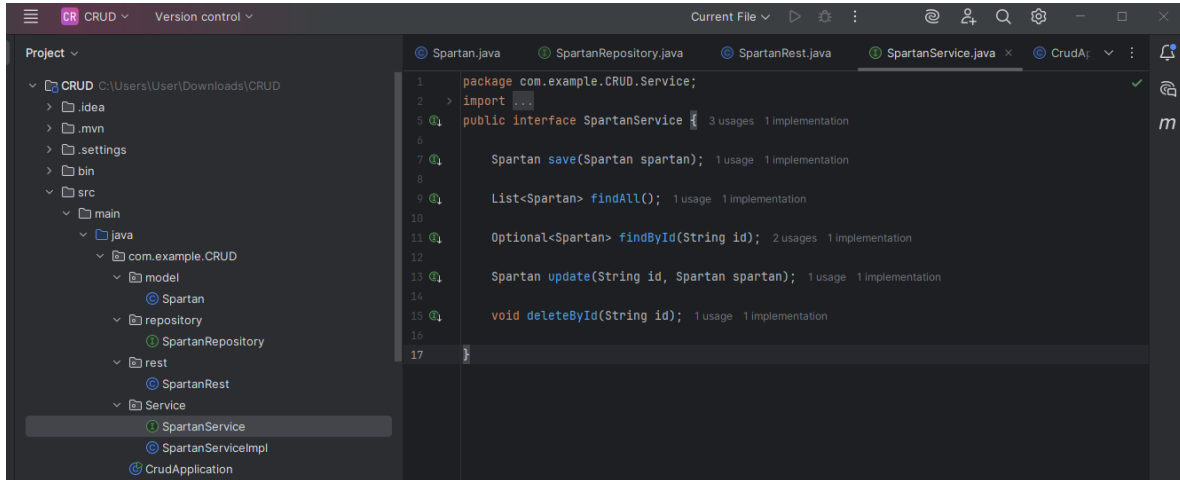
Ejemplo: Método Post

- Recibe en el cuerpo de la solicitud (`@RequestBody`) y lo guarda usando el servicio.
- Devuelve 201 CREATED si se guarda correctamente.



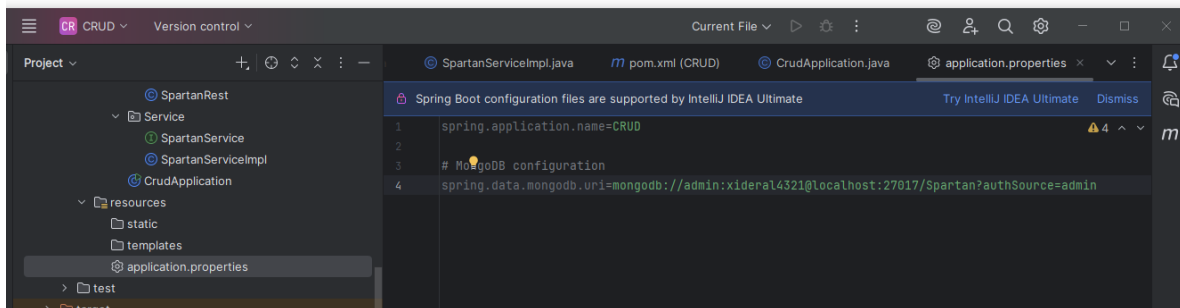
Descripción de la Interfaz SpartanService

En esta Interfaz podemos ver las operaciones que se realizarán, que recibe las solicitudes del usuario a la hora de ingresar un dato y en el repository que encargará de manejarlo a la base de datos.



Descripción de la interfaz Application.Properties.

En esta parte se declara la configuración sobre el puerto de MongoDB anteriormente levantado con Docker

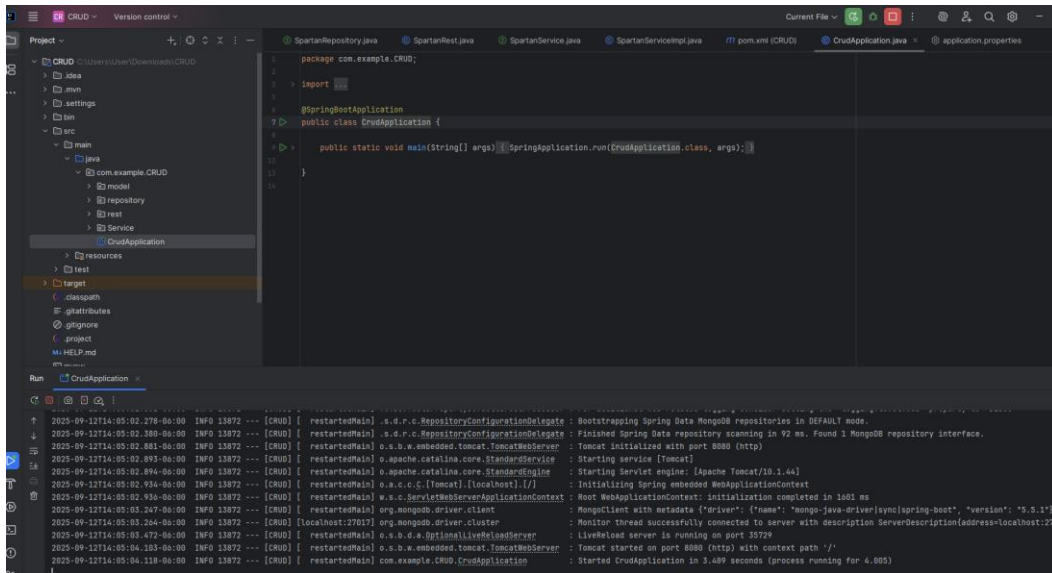


1. spring.application.name=CRUD Le da un nombre a la aplicación.
2. spring.data.mongodb.uri= Indica cómo conectarse a MongoDB, con:
 - Usuario y contraseña (admin:xideral4321)
 - Dirección y puerto (localhost:27017) . Como levantamos MongoDB con Docker en ese puerto, usamos la misma dirección y puerto para que Spring Boot se conecte correctamente a la base de datos.
 - Base de datos (Spartan)
 - Autenticación en la DB de admin (authSource=admin)

Prueba de Caja Negra

Crear un Spartan (POST /api/spartans)

1.- Levantamos el Servicio y se inicia en el puerto 8080



The screenshot shows an IDE with a project named 'CRUD'. The main file is 'CrudApplication.java', which is a Spring Boot application. The code is as follows:

```
package com.example.CRUD;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class CrudApplication {

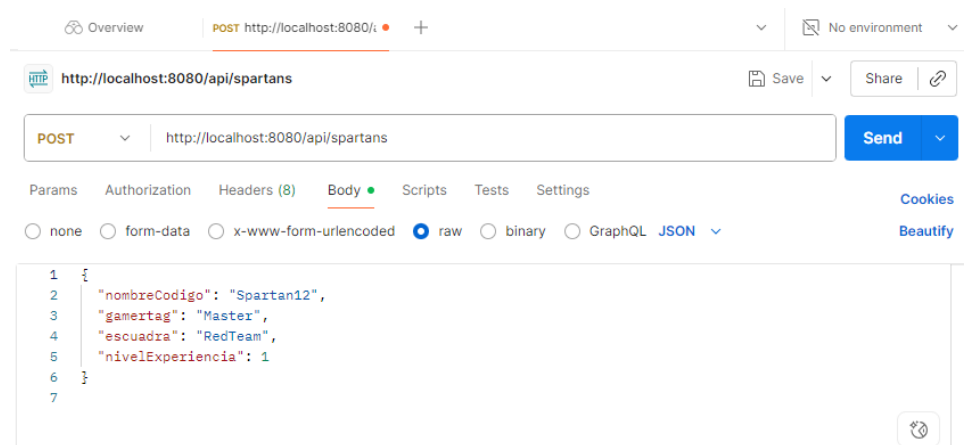
    public static void main(String[] args) {
        SpringApplication.run(CrudApplication.class, args);
    }
}
```

The 'Run' console at the bottom shows the following logs:

```
2025-09-12T14:05:02.278-04:00 INFO 13872 --- [CRUD] [ restartedMain] s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data MongoDB repositories in DEFAULT mode.
2025-09-12T14:05:02.380-04:00 INFO 13872 --- [CRUD] [ restartedMain] s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 92 ms. Found 1 MongoDB repository interface.
2025-09-12T14:05:02.881-04:00 INFO 13872 --- [CRUD] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 (http)
2025-09-12T14:05:02.893-04:00 INFO 13872 --- [CRUD] [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2025-09-12T14:05:02.894-04:00 INFO 13872 --- [CRUD] [ restartedMain] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.44]
2025-09-12T14:05:02.934-04:00 INFO 13872 --- [CRUD] [ restartedMain] o.s.c.s.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2025-09-12T14:05:02.936-04:00 INFO 13872 --- [CRUD] [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1601 ms
2025-09-12T14:05:03.247-04:00 INFO 13872 --- [CRUD] [ restartedMain] org.mongodb.driver.client : MongoClient with metadata {'driver': 'mongo-java-driver|sync|spring-boot', 'version': '3.5.1'}
2025-09-12T14:05:03.264-04:00 INFO 13872 --- [CRUD] [ restartedMain] org.mongodb.driver.cluster : Monitor thread successfully connected to server with description ServerDescription(address=localhost:2702)
2025-09-12T14:05:03.422-04:00 INFO 13872 --- [CRUD] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : LiveReload server is running on port 35729
2025-09-12T14:05:04.163-04:00 INFO 13872 --- [CRUD] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path '/'
2025-09-12T14:05:04.118-04:00 INFO 13872 --- [CRUD] [ restartedMain] com.example.CRUD.CrudApplication : Started CrudApplication in 3.489 seconds (process running for 4.005)
```

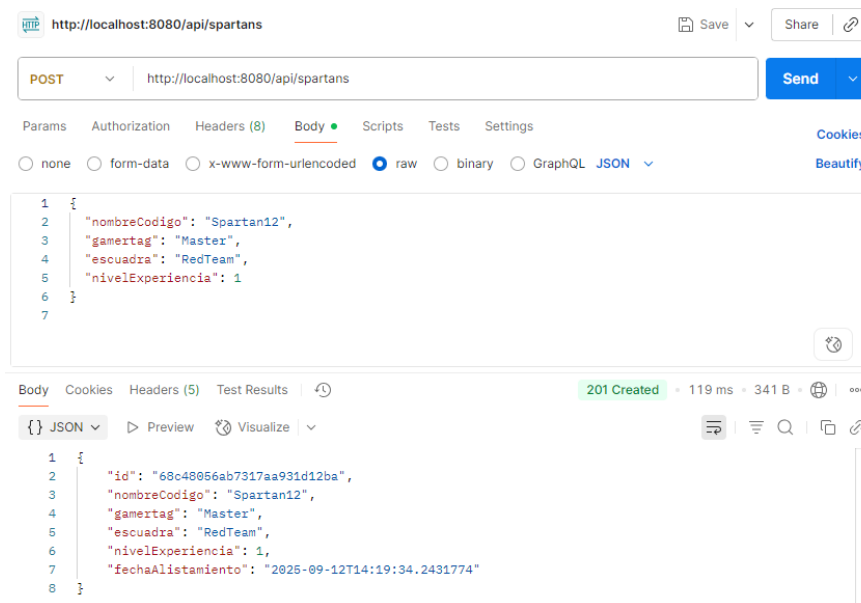
2.- Nos vamos a Postman para poder probar nuestro servicio levantado en el puerto 8080 y creamos una nueva petición POST hacia la ruta:

<http://localhost:8080/api/spartans>



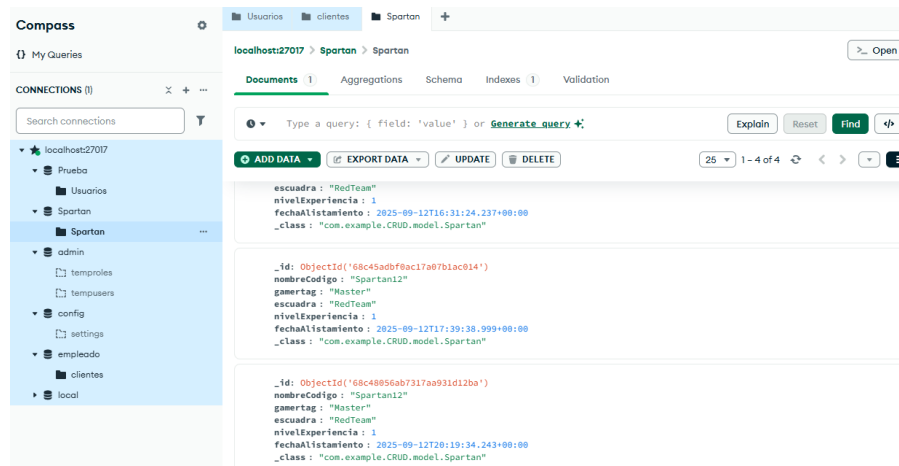
3.- Enviamos la petición para poder registrar un nuevo dato

El servicio debe responder con un HTTP 201 (Created).



Conclusión de las pruebas en Postman y con MongoDB

Ya que probamos con Postman el registro de un Spartan, podemos decir que nuestro servicio en el puerto 8080 está funcionando correctamente y guarda la información en MongoDB.



De la misma manera, con Postman también podemos hacer el resto de operaciones:

- **GET** para consultar los Spartans,
- **PUT** para actualizarlos,
- **DELETE** para eliminarlos.