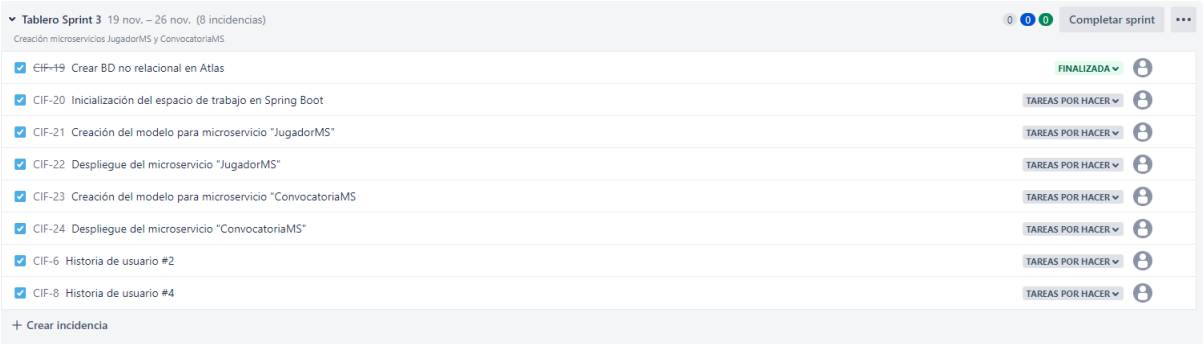
FUTMATCH

Informe Sprint #3

Jira

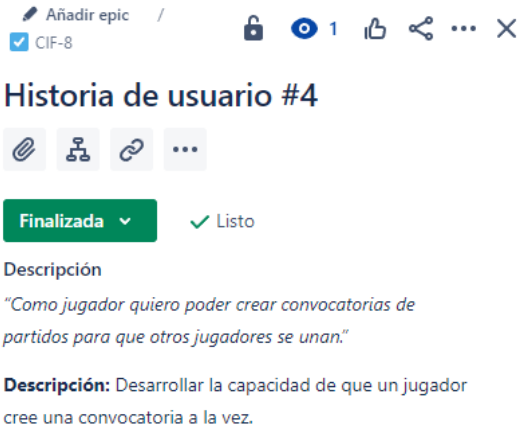
Para el sprint #3 se crearon las siguientes incidencias en la plataforma de Jira:

- Crear BBDD no relacional en Atlas
- Inicialización del espacio de trabajo en Spring Boot
- Creación del modelo para microservicio “JugadorMS”
- Despliegue del microservicio “JugadorMS”
- Creación del modelo para microservicio “ConvocatoriaMS”
- Despliegue del microservicio “ConvocatoriaMS”
- Y se añadieron las historias de usuario #2 y #4



Se realizaron las respectivas tareas y dentro de cada incidencia se añadieron las respectivas pruebas, y archivos adjuntos.

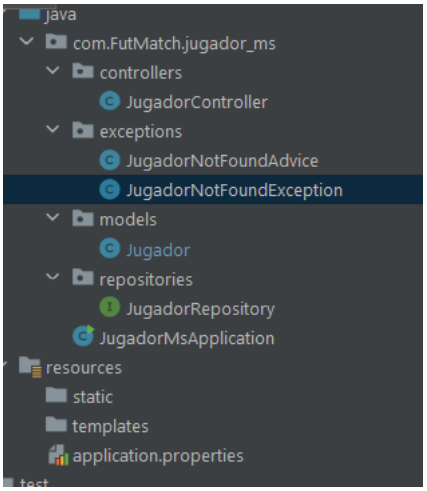
En este sprint se desarrollaron las historias de usuario #2 y #4, que permiten a un usuario crear un perfil de jugador para acceder a las convocatorias y crear convocatorias para atraer jugadores.



Código (Microservicio Jugador)

Nuestro segundo microservicio (el primero fue el de autenticación), es el de jugador. Este microservicio permite la creación de un perfil de jugador en la que el usuario podrá ingresar su nombre (público), fecha de nacimiento y aparecerán las convocatorias jugadas en las que ha participado en la app - este valor no es modificable por el usuario (esto con el propósito de generar confianza en los demás jugadores).

La estructura del código de nuestro microservicio es:

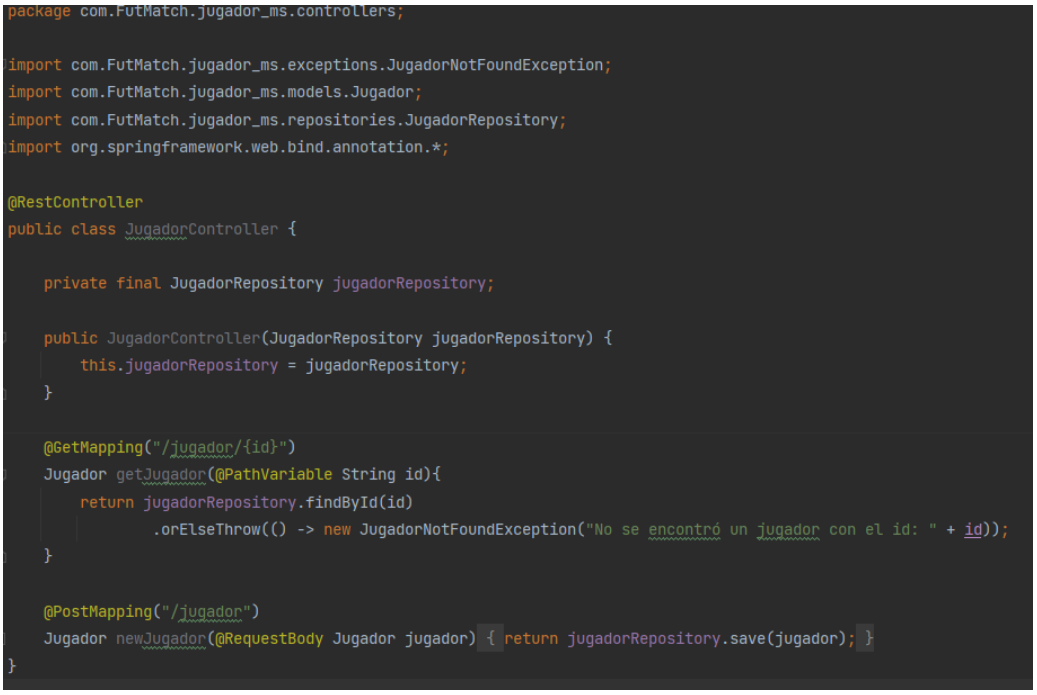


Consiste inicialmente en el modelo Jugador con el siguiente código:

```java
package com.FutMatch.jugador_ms.models;

import org.springframework.data.annotation.Id;
import java.util.Date;

public class Jugador {

    @Id
    private String nombre;
    private Date fechaNacimiento;
    private Integer convJugadas;


    public Jugador(String nombre, Date fechaNacimiento, Integer convJugadas){
        this.nombre = nombre;
        this.fechaNacimiento = fechaNacimiento;
        this.convJugadas = convJugadas;
    }

    public String getNombre() { return nombre; }
    public void setNombre(String nombre) { this.nombre = nombre; }
    public Date getFechaNacimiento() { return fechaNacimiento; }
    public void setFechaNacimiento(Date fechaNacimiento) { this.fechaNacimiento = fechaNacimiento; }
    public Integer getConvJugadas() { return convJugadas; }
    public void setConvJugadas(Integer convJugadas) { this.convJugadas = convJugadas; }

}
```

En el que se establecen los atributos y los getter y los setters. (Esto sigue el esquema planteado en el primer sprint - documento de excel)

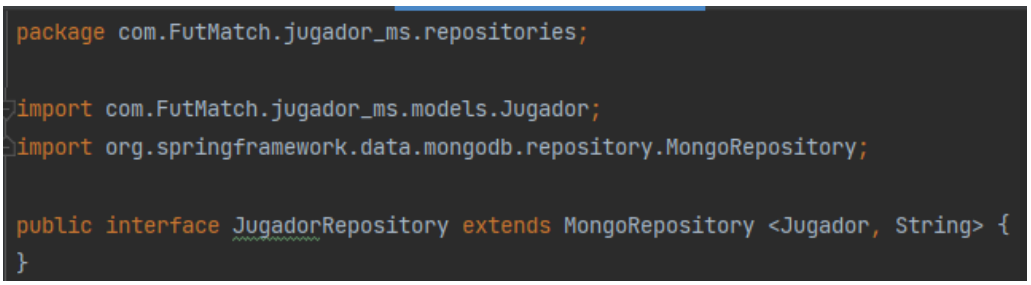
Se crea el controlador de Jugador con el siguiente código:



```java
package com.FutMatch.jugador_ms.controllers;

import com.FutMatch.jugador_ms.exceptions.JugadorNotFoundException;
import com.FutMatch.jugador_ms.models.Jugador;
import com.FutMatch.jugador_ms.repositories.JugadorRepository;
import org.springframework.web.bind.annotation.*;

@RestController
public class JugadorController {

    private final JugadorRepository jugadorRepository;

    public JugadorController(JugadorRepository jugadorRepository) {
        this.jugadorRepository = jugadorRepository;
    }

    @GetMapping("/jugador/{id}")
    Jugador getJugador(@PathVariable String id){
        return jugadorRepository.findById(id)
                .orElseThrow(() -> new JugadorNotFoundException("No se encontró un jugador con el id: " + id));
    }

    @PostMapping("/jugador")
    Jugador newJugador(@RequestBody Jugador jugador) { return jugadorRepository.save(jugador); }
}
```

En el que se permite la creación y consulta de la información del jugador. (Falta adicionar la funcionalidad de la eliminación del perfil. No se permitirá su modificación).

El repositorio de Jugador solo extiende del de Mongo:



```java
package com.FutMatch.jugador_ms.repositories;

import com.FutMatch.jugador_ms.models.Jugador;
import org.springframework.data.mongodb.repository.MongoRepository;

public interface JugadorRepository extends MongoRepository <Jugador, String> {
}
```

Finalmente, se maneja, por ahora, una excepción, la de un usuario no encontrado, con el siguiente código para el Advice y para la Exception:

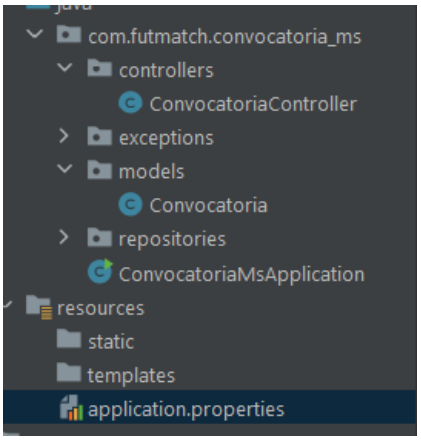
```java
package com.FutMatch.jugador_ms.exceptions;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.ResponseStatus;

@ControllerAdvice
@ResponseBody

public class JugadorNotFoundAdvice {
    @ResponseBody
    @ExceptionHandler(JugadorNotFoundException.class)
    @ResponseStatus(HttpStatus.NOT_FOUND)
    String EntityNotFoundAdvice(JugadorNotFoundException ex) { return ex.getMessage(); }
}
```

```java
package com.FutMatch.jugador_ms.exceptions;

public class JugadorNotFoundException extends RuntimeException {
    public JugadorNotFoundException(String message) {
        super(message);
    }
}
```

Código (Microservicio Convocatoria)

Nuestro tercer microservicio es el de convocatoria. Este busca crear y consultar convocatorias (falta adicionarle la funcionalidad de eliminar convocatoria, que será restringida solo para el usuario que lo crea; y la funcionalidad de consulta según localidad o fecha).

La estructura de nuestro microservicio es la siguiente:



Para las excepciones y el repositorio se maneja el código similar el microservicio de Jugador.

El código del modelo de convocatoria es el siguiente:

```java
package com.futmatch.convocatoria_ms.models;

import org.springframework.data.annotation.Id;
import java.time.LocalDate;
import java.time.LocalTime;
import java.util.ArrayList;


public class Convocatoria{
    @Id
    private String id;
    private ArrayList<String> jugEquip1;
    private ArrayList<String> jugEquip2;
    private LocalDate fecha;
    private LocalTime hora;
    private String localidad;
    private ArrayList<String> posiciones;

    public Convocatoria(String id, ArrayList<String> jugEquip1, ArrayList<String> jugEquip2, LocalDate fecha, LocalTime hora, String localidad, ArrayList<String> posiciones) {
        this.id = id;
        this.jugEquip1 = jugEquip1;
        this.jugEquip2 = jugEquip2;
        this.fecha = fecha;
        this.hora = hora;
        this.localidad = localidad;
        this.posiciones = posiciones;
    }
```

En el modelo aparecen todos los atributos planteados inicialmente en la estructura (excel del sprint #1), y los respectivos getters y setters que no aparecen en la imagen pero se encuentran en el código.

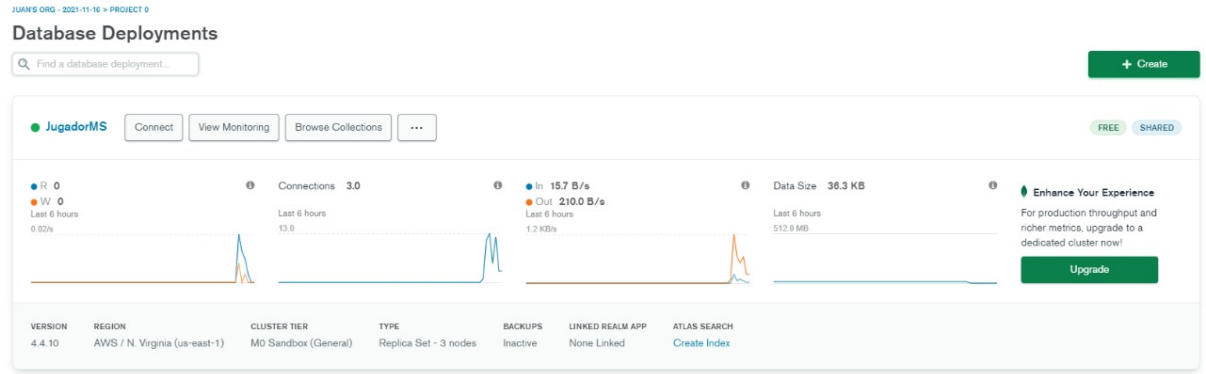
El código del controlador de convocatoria es:

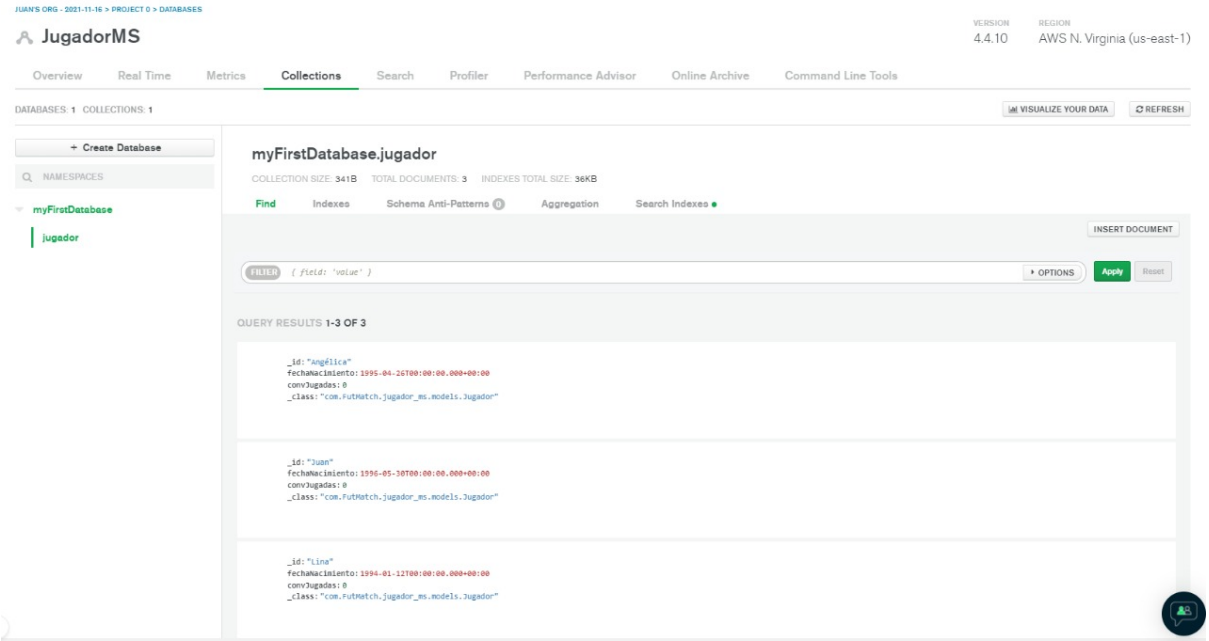
```java
package com.futmatch.convocatoria_ms.controllers;

import com.futmatch.convocatoria_ms.exceptions.ConvocatoriaNotFoundException;
import com.futmatch.convocatoria_ms.models.Convocatoria;
import com.futmatch.convocatoria_ms.repositories.ConvocatoriaRepository;
import org.springframework.web.bind.annotation.*;

@RestController /*para que esa clase sea un controlador rest, crear vistas para rtas de http*/
public class ConvocatoriaController {

    private final ConvocatoriaRepository convocatoriaRepository;

    public ConvocatoriaController(ConvocatoriaRepository convocatoriaRepository) {
        this.convocatoriaRepository = convocatoriaRepository;
    }

    @GetMapping("/convocatorias/{id}")
    Convocatoria getConvocatoria(@PathVariable String id){
        return convocatoriaRepository.findById(id)
                .orElseThrow(() -> new ConvocatoriaNotFoundException("No se encontro ninguna convocatoria con el id" + id));
    }

    @PostMapping("/convocatorias")
    Convocatoria newConvocatoria(@RequestBody Convocatoria convocatoria){
        return convocatoriaRepository.save(convocatoria);
    }
}
```

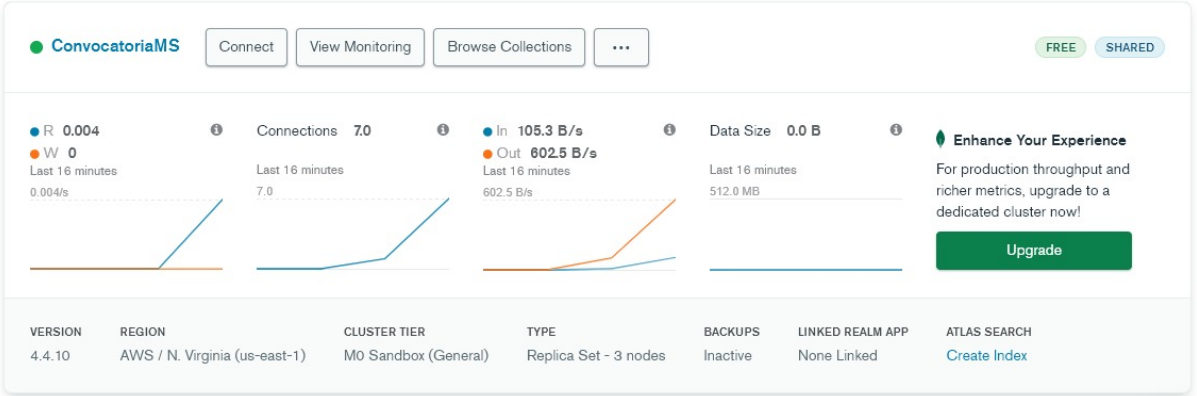
En dónde se encuentran los métodos para crear y consultar la convocatoria por Id (falta adicionar la consulta por localidad y por fecha, ya que son filtros que queremos adicionar para la consulta de las convocatorias en la app).

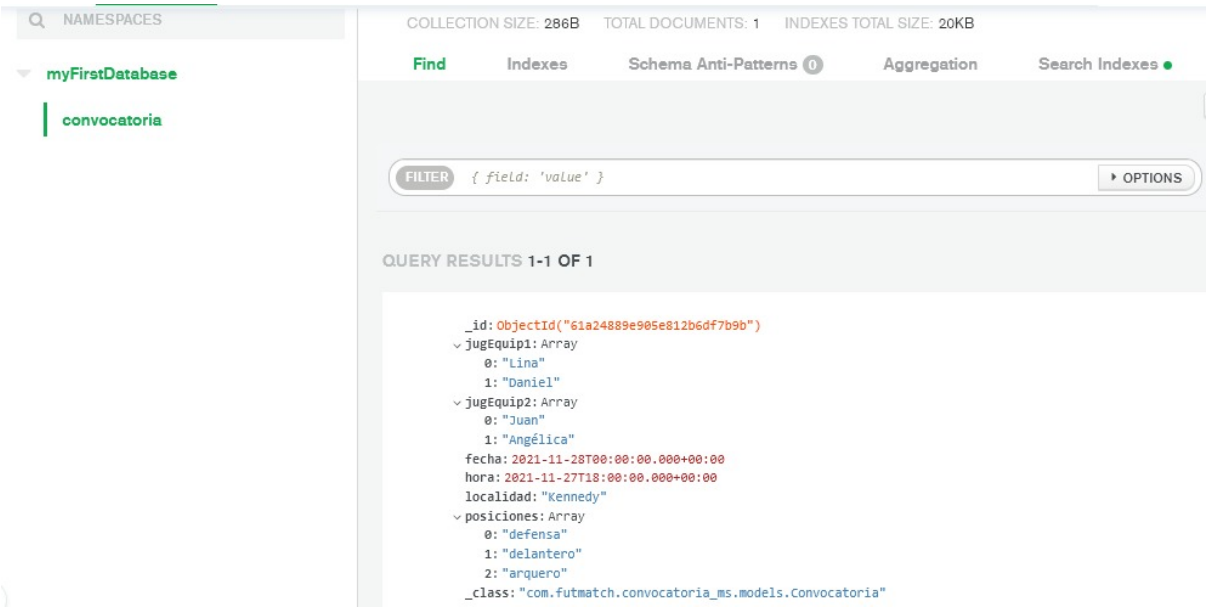
Bases de Datos (No relacionales)

Haciendo uso de la plataforma Atlas de MongoDB realizamos el despliegue de la base de datos para cada uno de los microservicios que implementaremos *Jugador* y *Convocatoria*, habilitando las conexiones externas para que puedan ser utilizadas desde cualquier dispositivo. Teniendo en cuenta también el driver y la versión que estamos utilizando en la creación de nuestros microservicios *(Java, 4.3 or later)*.









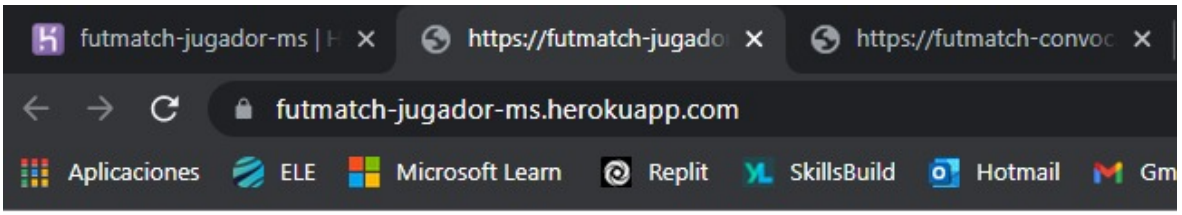
Las URLs que fueron generadas con la base de datos y que le brinda la información necesaria a Spring Boot para realizar la conexión con MongoDB fueron alojadas en el archivo *src/main/resources/application.properties*, respectivamente.

```
spring.data.mongodb.uri=${URL_DB:mongodb+srv://futmatch-jugador:1234@jugadorms.uspwl.mongodb\
  .net/myFirstDatabase?retryWrites=true&w=majority}
server.port=${PORT:8080}
```

```
spring.data.mongodb.uri=${URL_DB:mongodb+srv://futmatch-convocatoria:1234@convocatoriams.aut8k.mongodb\
  .net/myFirstDatabase?retryWrites=true&w=majority}
server.port=${PORT:8080}
```

Heroku (despliegue)

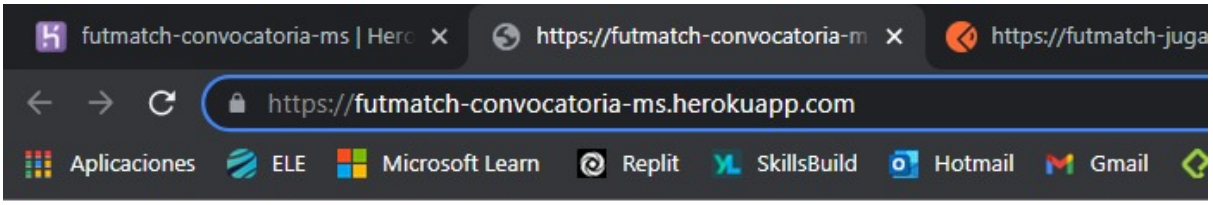
Al hacer el despliegue en Heroku y ejecutar la aplicación no presenta errores y en su respectiva URL presenta el resultado esperado para cada uno de los microservicios:



Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Sat Nov 27 14:58:38 GMT 2021
There was an unexpected error (type=Not Found, status=404).



Whitelabel Error Page

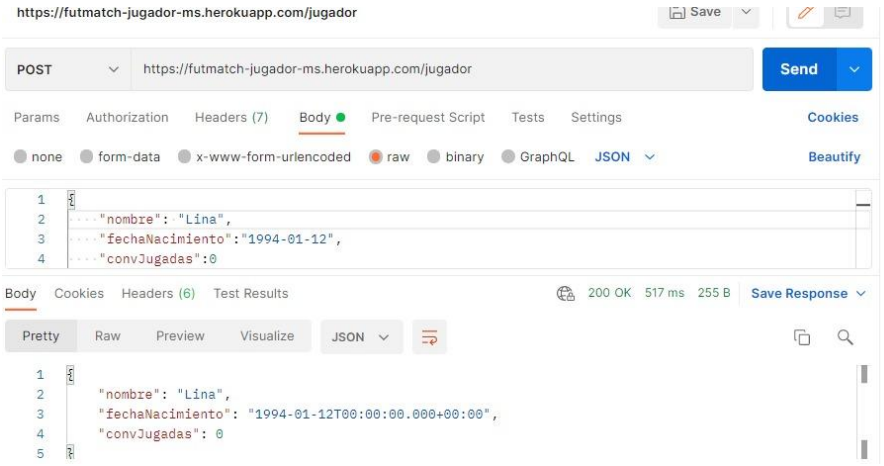
This application has no explicit mapping for /error, so you are seeing this as a fallback.

Sat Nov 27 14:59:11 GMT 2021
There was an unexpected error (type=Not Found, status=404).

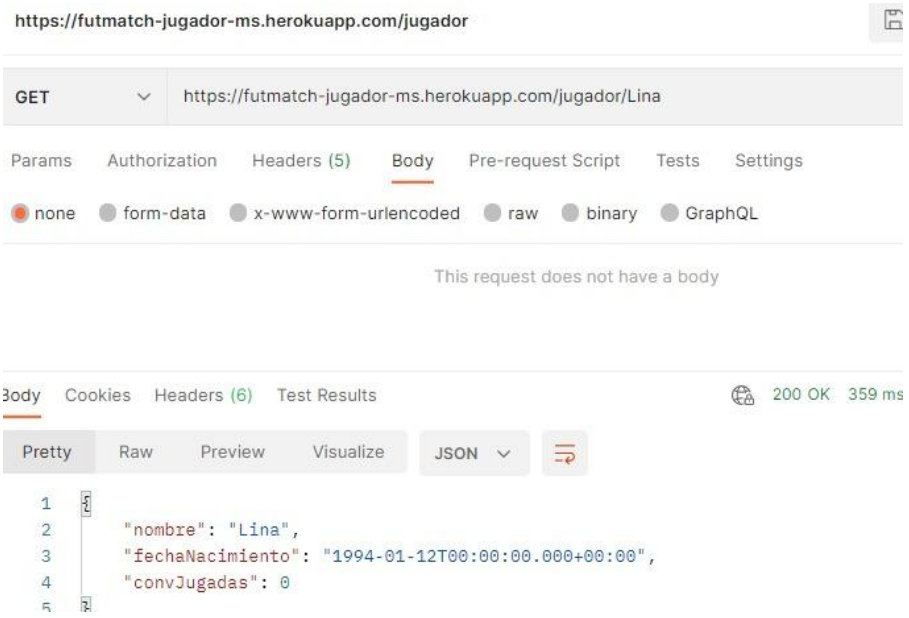
Pruebas (postman)

Después del despliegue de la aplicación en Heroku realizamos las respectivas pruebas en el aplicativo web de Postman.

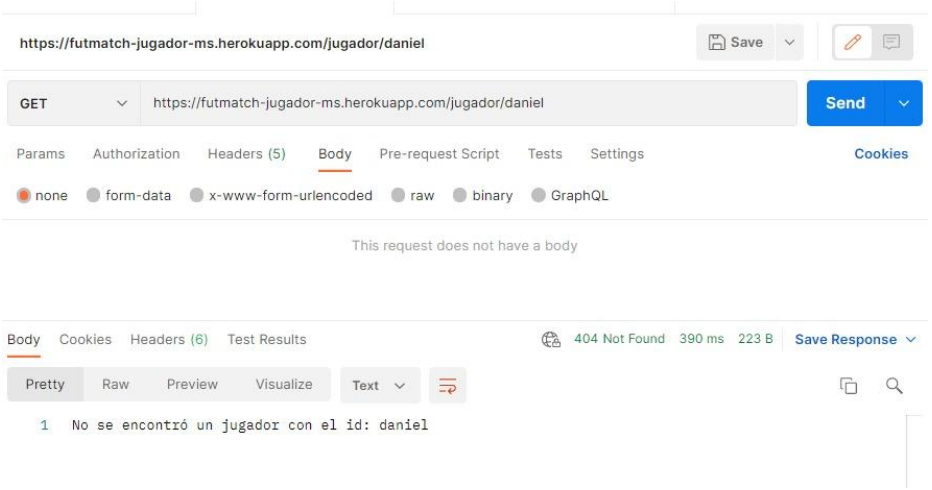
Primero registramos un jugador nuevo, con su respectiva información:



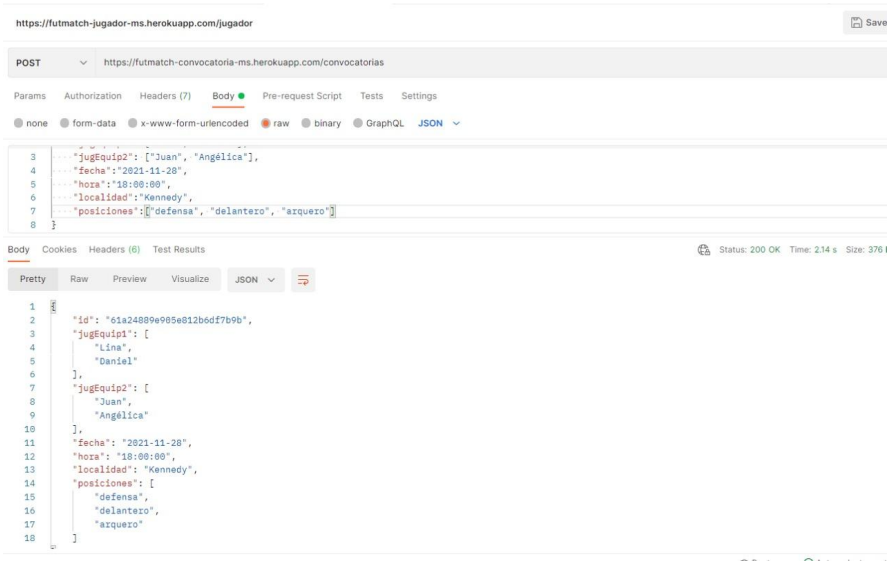
Esta información quedó guardada en la BBDD de jugador. Una vez creado accedimos al enlace en el cual podemos verificar la información de un jugador



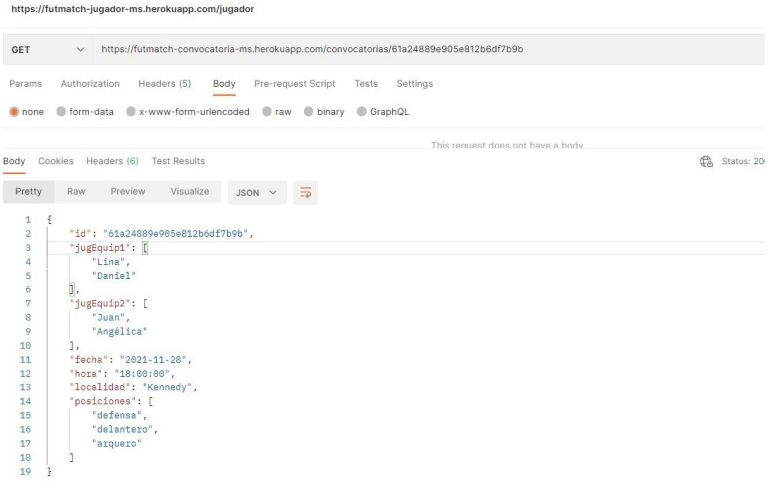
También, mediante el uso de excepciones, verificamos el mensaje que sale cuando se busca un jugador del cual no existe registro



Realizamos el mismo procedimiento con las convocatorias; primero creamos una convocatoria con los datos requeridos



Y accedimos a la información de una convocatoria puntual



Repositorio

Hemos mantenido nuestro repositorio en línea actualizado con los commits correspondientes para cada punto importante del desarrollo.
Además, desarrollamos en nuestra rama develop y cuando se tiene listo el microservicio, se fusiona con la rama principal.