



**UNIVERSIDAD
FRANCISCO GAVIDIA**
Tecnología, Innovación y Calidad

Facultad: Ingeniería y Sistemas

Módulo: Desarrollo de Aplicaciones Web

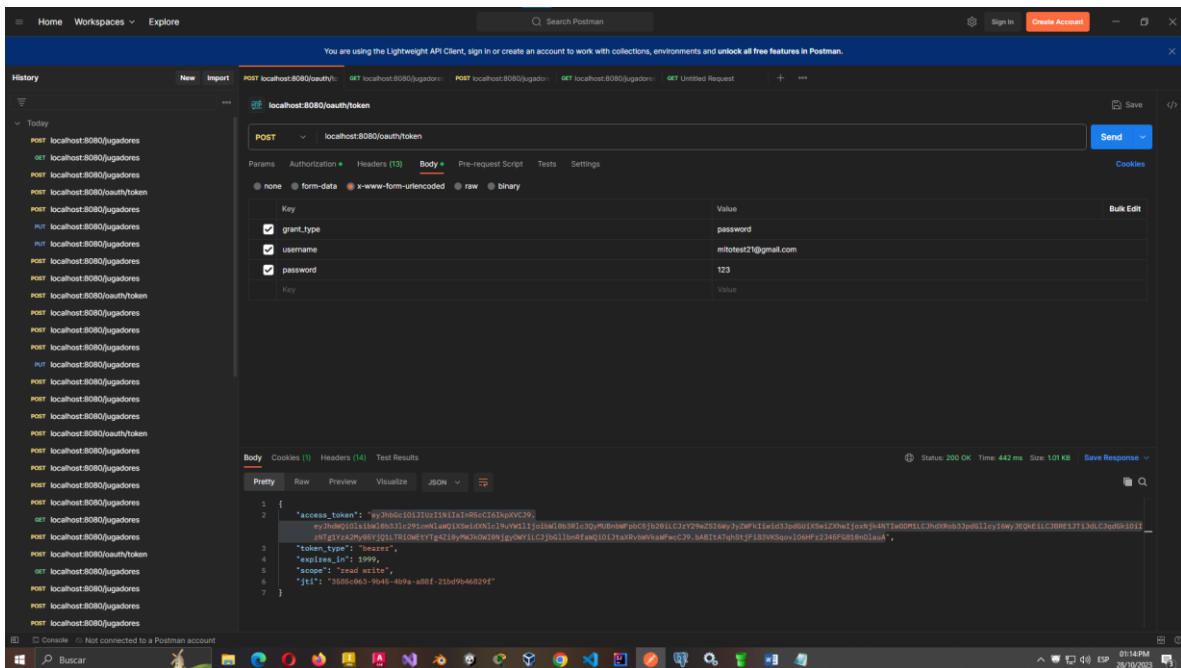
Docente: Carlos Boris Martinez Calzadia

Alumno: Garby Edenilson Álvarez Velásquez

Actividad: Laboratorio 3

Fecha de Entrega: 22 de octubre de 2023

Postman funcionando:



En POST se colocan los datos de username y password que se especifican en el application.properties y el link del token de oauth.

A screenshot of the Postman application interface. At the top, the HTTP method is 'POST' and the URL is 'localhost:8080/oauth/token'. Below this, there are tabs for 'Params', 'Authorization', 'Headers (13)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Authorization' tab is selected and highlighted with an orange underline. In the 'Authorization' section, the 'Type' is set to 'Basic Auth'. Below this, a text box explains: 'The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)'. To the right, there are two input fields: 'Username' with the value 'mitomedlapp' and 'Password' with the value 'mito89codexS'. A warning icon is visible next to the password field.

```
security.oauth2.resource.filter-order=3

#https://www.allkeysgenerator.com/Random/Security-Encryption-Key-Generator.aspx
security.signing-key=MaYzkSjmkzPC57L
security.encoding-strength=256
security.security-realm=Spring Boot JWT

security.jwt.client-id=mitomediapp
security.jwt.client-secret=mito89codex
security.jwt.grant-type=password
security.jwt.scope-read=read
security.jwt.scope-write=write
security.jwt.resource-ids=mitoresourceid
```

En el body se colocan los datos de usuario en la base de datos:

Params

Authorization

Headers (13)

Body

Pre-request Script

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

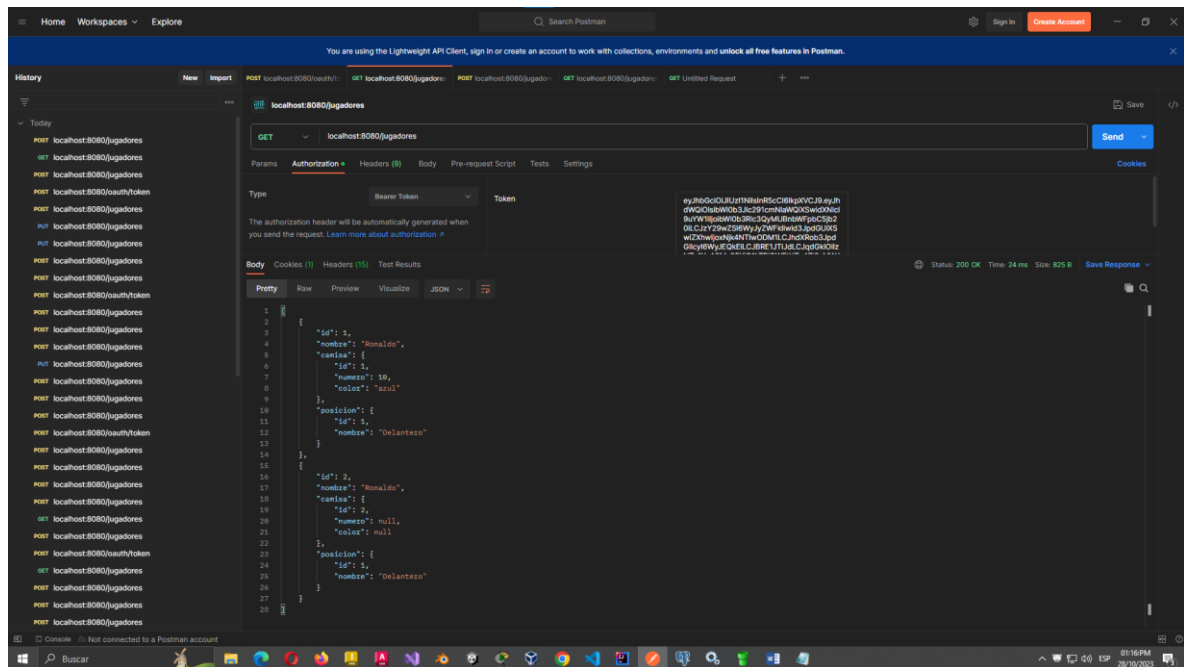
binary

Key	Value
<input checked="" type="checkbox"/> grant_type	password
<input checked="" type="checkbox"/> username	mitotest21@gmail.com
<input checked="" type="checkbox"/> password	123
Key	Value

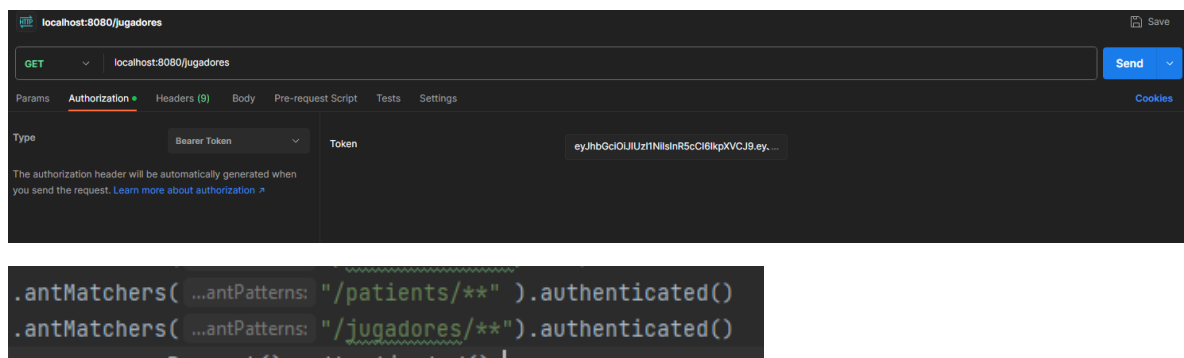
Acá se visualiza el token siendo generado totalmente funcional:

```
Body Cookies (1) Headers (14) Test Results 200 OK 442 ms 1.01 KB Save Response
Pretty Raw Preview Visualize JSON
{
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOiJ1b3Jlc291cmNlYWQIXSwidXN1c19uYW11IjoibWl0b3Rlc3QyMUBnbWVpbC5jb20iLCJzY29wZSI6WyJyZWFrIiwid3JpdGUiXSwiZXhwIjoxNjk4NTIwODM1LCJhdXRob3JpdGllcyI6WyJEQkEiLCJBRE1JTjJdLCJqdGkiOiIzNTg1YzA2My85YjQ1LTRiOWEtyTg4Zi8yMWJkOWI0NjgyOWYiLCJjbGllbnRfaWQiOiJtaXRvbnVkaWwFwcC3J9.bABiTA7qhStJfI83VKSqovl06HFr2J45FG810nDlauA",
  "token_type": "bearer",
  "expires_in": 1999,
  "scope": "read write",
  "jti": "3585c063-9b45-4b9a-a88f-21bd9b46829f"
}
```

Obtenemos los datos de la base mediante el token:



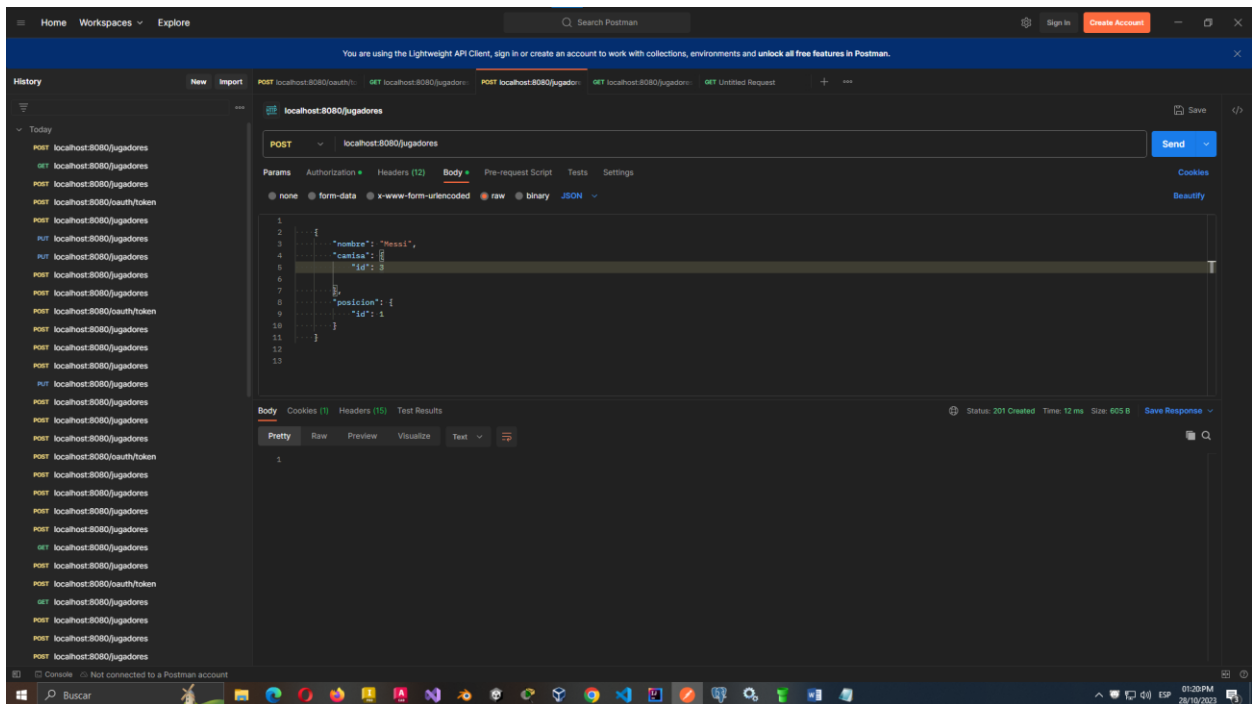
Arriba en el GET se pone el link de la api, luego se especifica Bearer Token como método de Autorización y se coloca el token anteriormente generado.



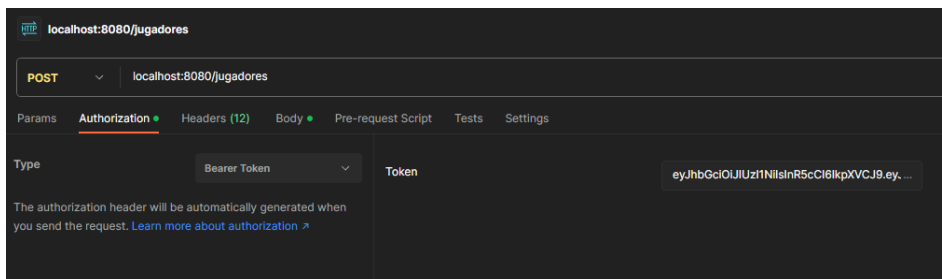
Datos obtenidos:

```
Body Cookies (1) Headers (15) Test Results
Pretty Raw Preview Visualize JSON
1
2 {
3   "id": 1,
4   "nombre": "Ronaldo",
5   "camisa": {
6     "id": 1,
7     "numero": 10,
8     "color": "azul"
9   },
10  "posicion": {
11    "id": 1,
12    "nombre": "Delantero"
13  },
14 },
15 {
16   "id": 2,
17   "nombre": "Ronaldo",
18   "camisa": {
19     "id": 2,
20     "numero": null,
21     "color": null
22   },
23   "posicion": {
24     "id": 1,
25     "nombre": "Delantero"
26   }
27 }
28 }
```

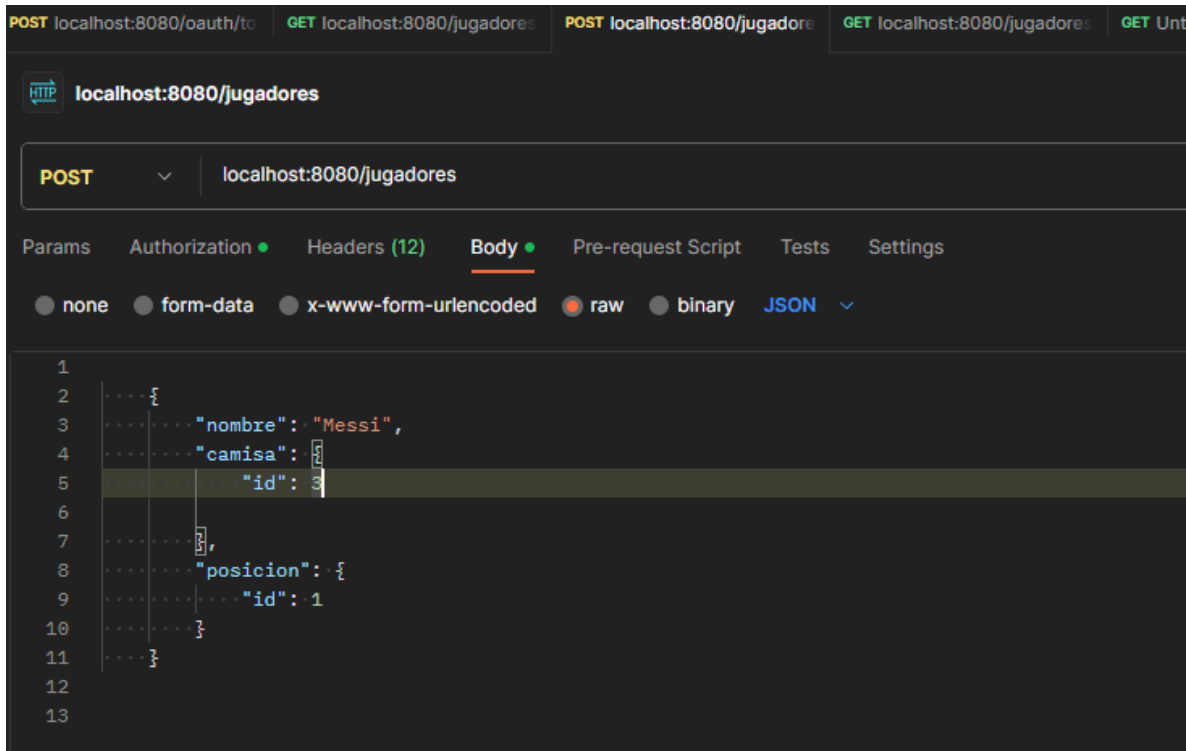
Insertar datos en la base mediante la API:



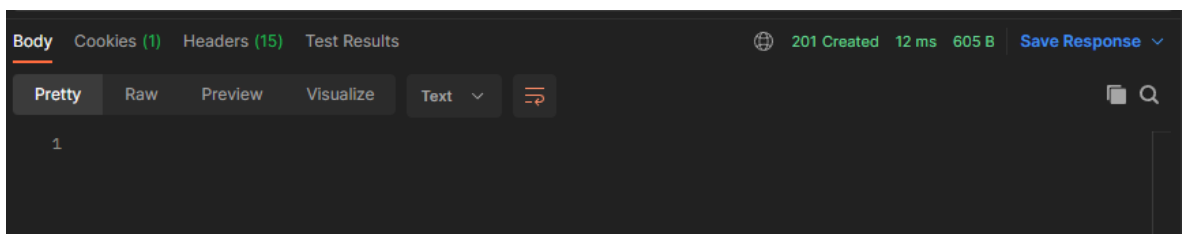
En POST se coloca el link y el token de acceso.

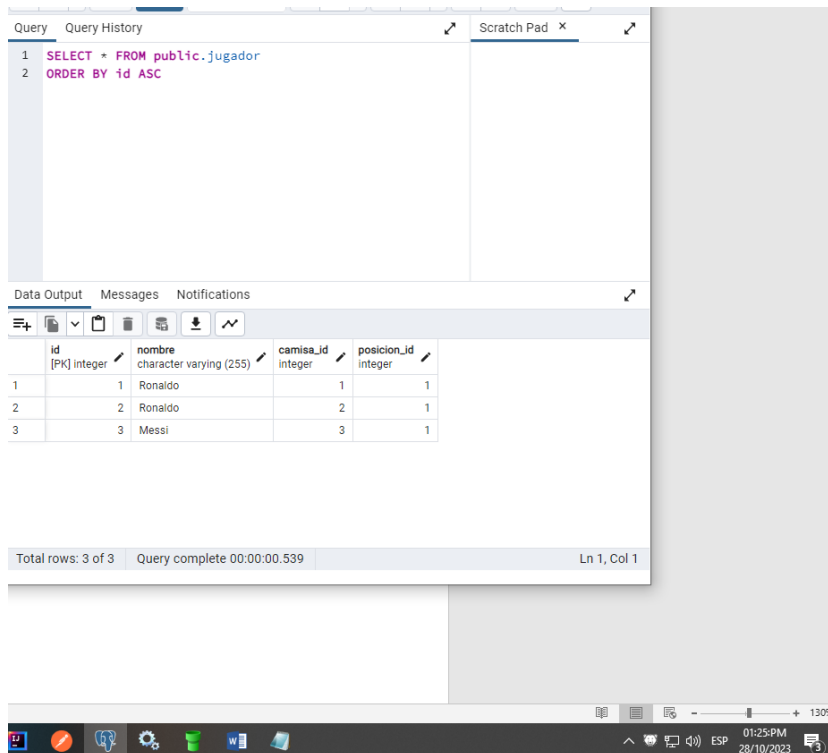


En body, se selecciona raw y se selecciona en JSON, en el editor de texto copiamos la estructura que nos genero anteriormente el GET, quitamos los corchetes y se colocan los datos de la manera como se ve ahí.



Para saber que funciona, veremos a la derecha, 20 Created, y en la base datos ya saldrá.

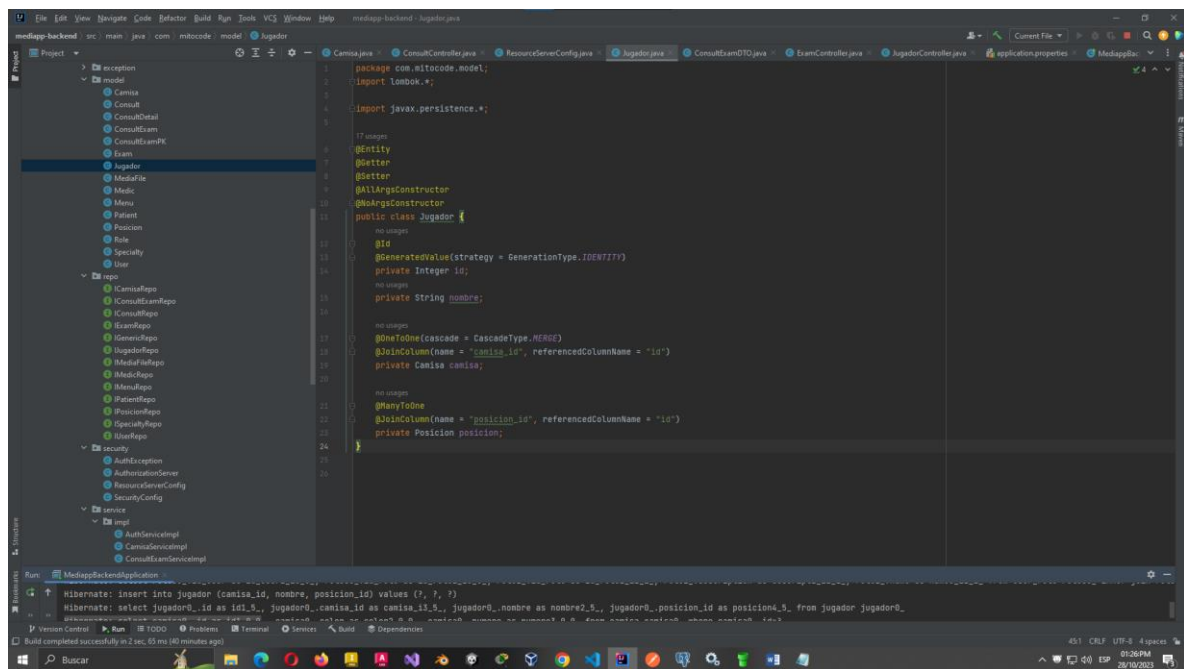




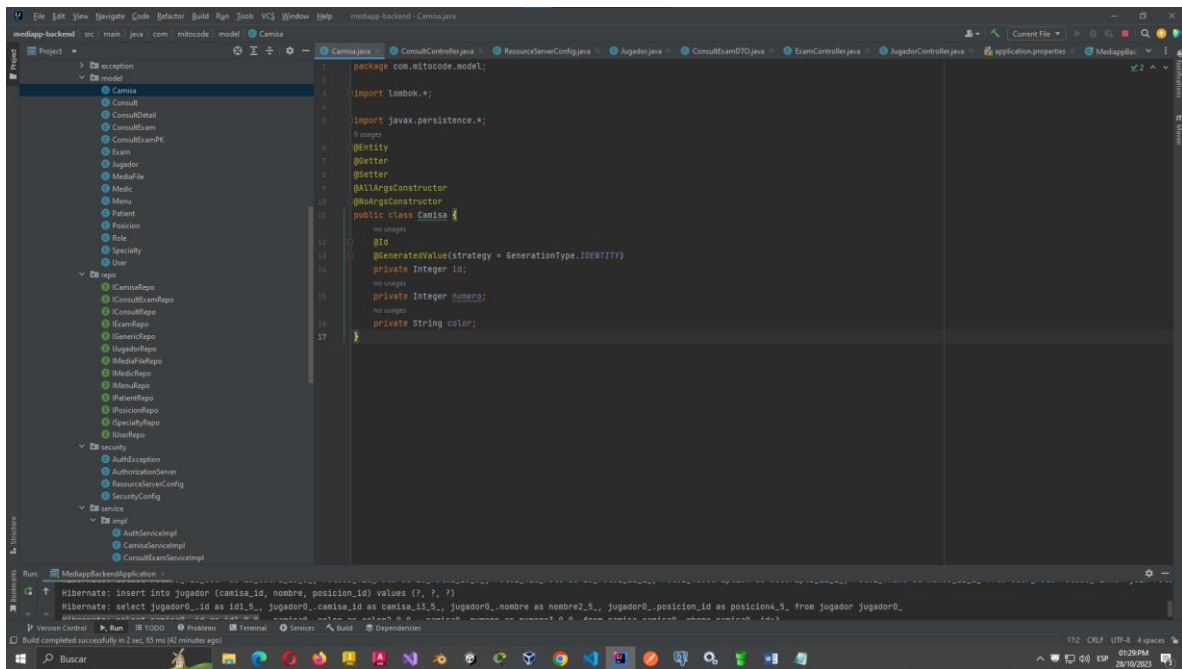
Modelos:

Modelo de Jugador:

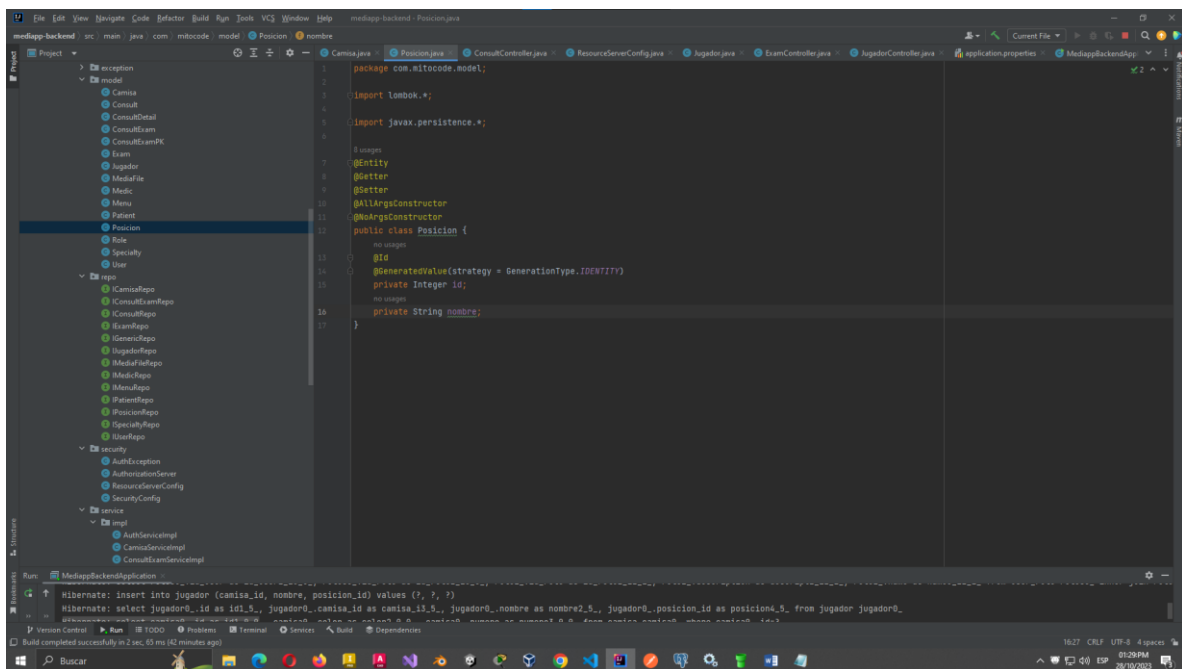
Es uno a uno con camisa ya que una camisa solo la puede llevar un jugador y muchos a uno en posición ya que un jugador puede jugar en más de una posición.



Modelo de Camisa:

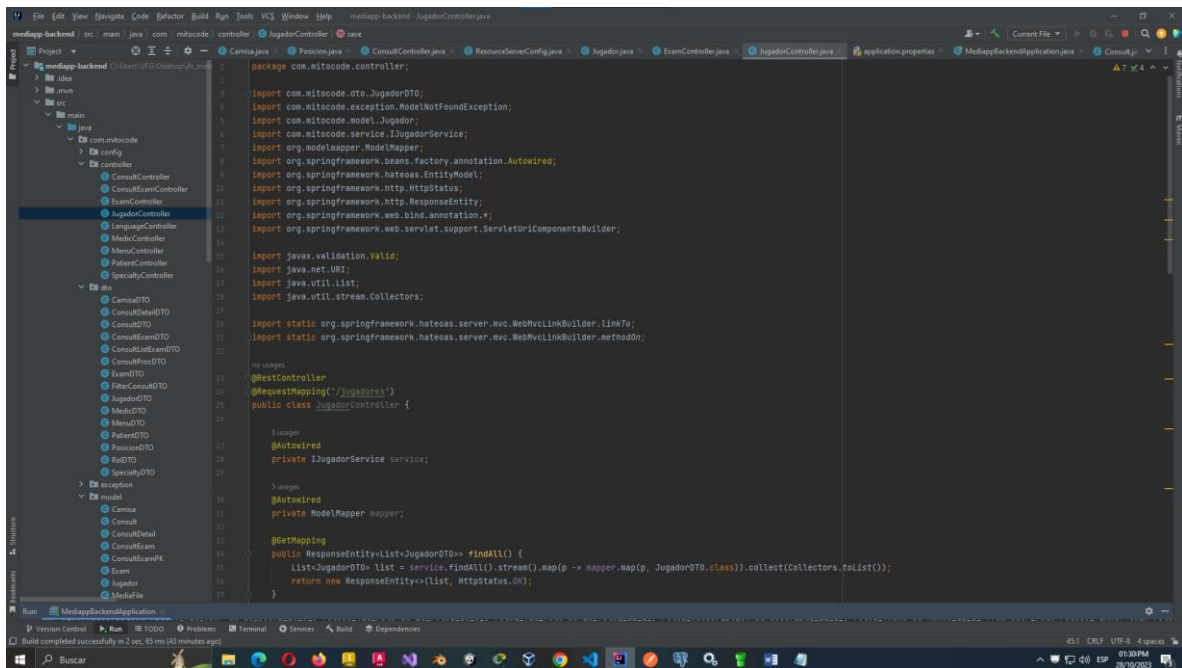


Modelo de posición:



Controlador:

Este es el controlador:



```
package com.mitocode.controller;

import com.mitocode.dto.JugadorDTO;
import com.mitocode.exception.ModelNotFoundException;
import com.mitocode.model.Jugador;
import com.mitocode.service.IJugadorService;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import javax.validation.Valid;
import java.net.URI;
import java.util.List;
import java.util.stream.Collectors;

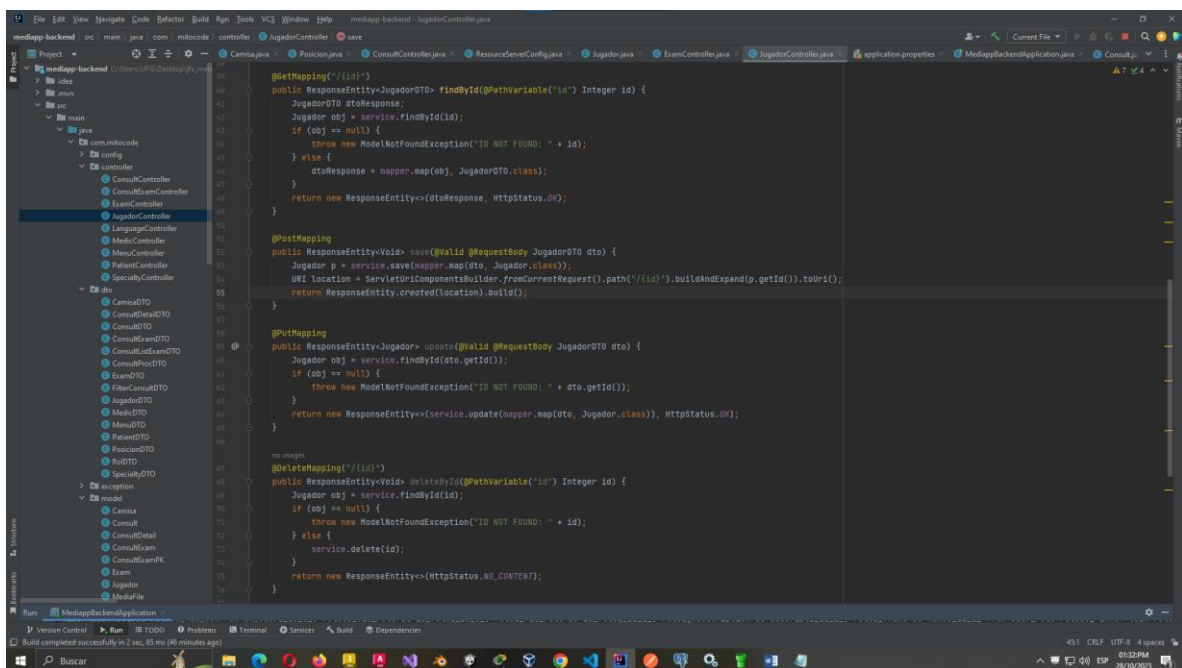
import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.LinkTo;
import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.methodOn;

@RestController
@RequestMapping("/jugadores")
public class JugadorController {

    @Autowired
    private IJugadorService service;

    @Autowired
    private IMapper mapper;

    @GetMapping
    public ResponseEntity<List<JugadorDTO>> findAll() {
        List<JugadorDTO> list = service.findAll().stream().map(p -> mapper.map(p, JugadorDTO.class)).collect(Collectors.toList());
        return new ResponseEntity<List<JugadorDTO>>(list, HttpStatus.OK);
    }
}
```



```
    @GetMapping("/{id}")
    public ResponseEntity<JugadorDTO> findById(@PathVariable("id") Integer id) {
        JugadorDTO dtoResponse;
        Jugador obj = service.findById(id);
        if (obj == null) {
            throw new ModelNotFoundException("ID NOT FOUND: " + id);
        } else {
            dtoResponse = mapper.map(obj, JugadorDTO.class);
        }
        return new ResponseEntity<JugadorDTO>(dtoResponse, HttpStatus.OK);
    }

    @PostMapping
    public ResponseEntity<Void> save(@Valid @RequestBody JugadorDTO dto) {
        Jugador p = service.save(mapper.map(dto, Jugador.class));
        URI location = ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}").buildAndExpand(p.getId()).toURI();
        return ResponseEntity.created(location).build();
    }

    @PutMapping
    public ResponseEntity<Jugador> update(@Valid @RequestBody JugadorDTO dto) {
        Jugador obj = service.findById(dto.getId());
        if (obj == null) {
            throw new ModelNotFoundException("ID NOT FOUND: " + dto.getId());
        }
        return new ResponseEntity<Jugador>(service.update(mapper.map(dto, Jugador.class)), HttpStatus.OK);
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteById(@PathVariable("id") Integer id) {
        Jugador obj = service.findById(id);
        if (obj == null) {
            throw new ModelNotFoundException("ID NOT FOUND: " + id);
        } else {
            service.delete(id);
        }
        return new ResponseEntity<Void>(HttpStatus.NO_CONTENT);
    }
}
```

Importaciones y creaci3n del controler:

```

1 package com.mitocode.controller;
2
3 import com.mitocode.dto.JugadorDTO;
4 import com.mitocode.exception.ModelNotFoundException;
5 import com.mitocode.model.Jugador;
6 import com.mitocode.service.IJugadorService;
7 import org.modelmapper.ModelMapper;
8 import org.springframework.beans.factory.annotation.Autowired;
9 import org.springframework.hateoas.EntityModel;
10 import org.springframework.http.HttpStatus;
11 import org.springframework.http.ResponseEntity;
12 import org.springframework.web.bind.annotation.*;
13 import org.springframework.web.servlet.support.ServletUriComponentsBuilder;
14
15 import javax.validation.Valid;
16 import java.net.URI;
17 import java.util.List;
18 import java.util.stream.Collectors;
19
20 import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.LinkTo;
21 import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.methodOn;
22
23 no usages
24 @RestController
25 @RequestMapping("/jugadores")
26 public class JugadorController {
27
28     8 usages
29     @Autowired
30     private IJugadorService service;
31
32     5 usages
33     @Autowired
34     private ModelMapper mapper;

```

GET dentro del controlador:

```

35 @GetMapping
36 public ResponseEntity<List<JugadorDTO>> findAll() {
37     List<JugadorDTO> list = service.findAll().stream().map(p -> mapper.map(p, JugadorDTO.class)).collect(Collectors.toList());
38     return new ResponseEntity<>(list, HttpStatus.OK);
39 }
40
41 @GetMapping("/{id}")
42 public ResponseEntity<JugadorDTO> findById(@PathVariable("id") Integer id) {
43     JugadorDTO dtoResponse;
44     Jugador obj = service.findById(id);
45     if (obj == null) {
46         throw new ModelNotFoundException("ID NOT FOUND: " + id);
47     } else {
48         dtoResponse = mapper.map(obj, JugadorDTO.class);
49     }
50     return new ResponseEntity<>(dtoResponse, HttpStatus.OK);
51 }

```

POST dentro del controlador:

```

52 @PostMapping
53 public ResponseEntity<Void> save(@Valid @RequestBody JugadorDTO dto) {
54     Jugador p = service.save(mapper.map(dto, Jugador.class));
55     URI location = ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}").buildAndExpand(p.getId()).toUri();
56     return ResponseEntity.created(location).build();
57 }
58
59 @PutMapping

```

PUT dentro del controlador.

```
58     @PutMapping
59     @ public ResponseEntity<Jugador> update(@Valid @RequestBody JugadorDTO dto) {
60         Jugador obj = service.findById(dto.getId());
61         if (obj == null) {
62             throw new ModelNotFoundException("ID NOT FOUND: " + dto.getId());
63         }
64         return new ResponseEntity<>(service.update(mapper.map(dto, Jugador.class)), HttpStatus.OK);
65     }
```

Delete dentro del controlador:

```
67     @DeleteMapping("/{id}")
68     public ResponseEntity<Void> deleteById(@PathVariable("id") Integer id) {
69         Jugador obj = service.findById(id);
70         if (obj == null) {
71             throw new ModelNotFoundException("ID NOT FOUND: " + id);
72         } else {
73             service.delete(id);
74         }
75         return new ResponseEntity<>(HttpStatus.NO_CONTENT);
76     }
77 }
```