



TDW

DOCUMENTACIÓN

DE LA API EN POSTAM

Integrantes:

ANDRADE GRANADOS DANIELA
LAGUNES VÁZQUEZ MILDRED VALERIA
RODRÍGUEZ MÉNDEZ DENYS MONSERRAT
VALENCIA SAN ROMAN JOEL

4BM1

Ajedrez



Get started here

This template guides you through CRUD operations (GET, POST, PUT, DELETE), variables, and tests.



How to use this template

Step 1: Send requests

RESTful APIs allow you to perform CRUD operations using the POST, GET, PUT, and DELETE HTTP methods.

This collection contains each of these request types. Open each request and click "Send" to see what happens.

Step 2: View responses

Observe the response tab for status code (200 OK), response time, and size.

Step 3: Send new Body data

Update or add new data in "Body" in the POST request. Typically, Body data is also used in PUT request.

Plain Text



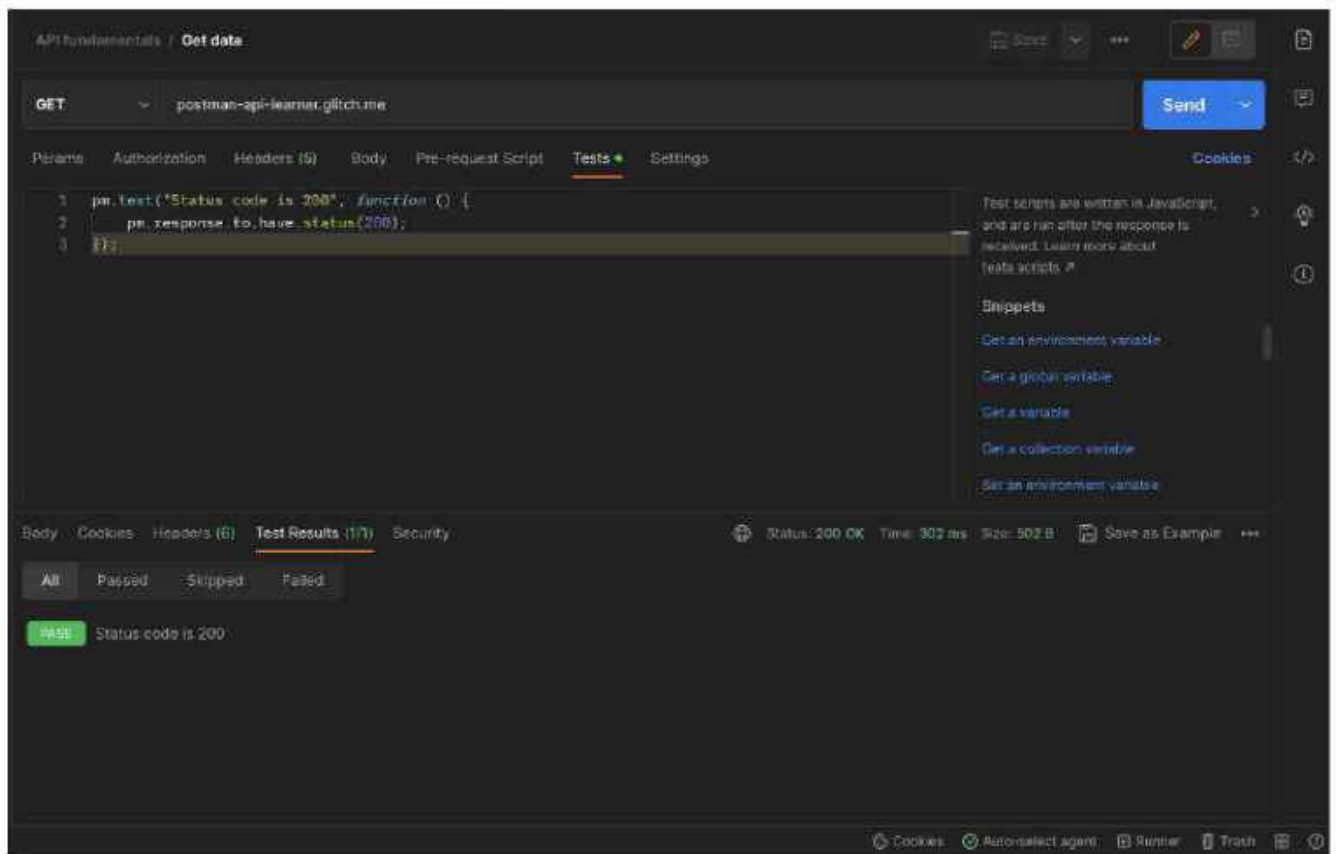
```
{  
  "name": "Add your name in the body"  
}
```

Step 4: Update the variable

Variables enable you to store and reuse values in Postman. We have created a variable called `base_url` with the sample request <https://postman-api-learner.glitch.me>. Replace it with your API endpoint to customize this collection.

Step 5: Add tests in the "Tests" tab

Tests help you confirm that your API is working as expected. You can write test scripts in JavaScript and view the output in the "Test Results" tab.



INITIALIZING

```
import { createClient } from '@supabase/supabase-js'
const supabaseUrl = 'https://rucfylvlgxlinvfwkxsdj.supabase.co'
const supabaseKey = process.env.SUPABASE_KEY
const supabase = createClient(supabaseUrl, supabaseKey)
```

Database Tables

Name	Description	Rows (Estimated)	Size (Estimated)	Realtime Enabled	
jugador_partida	No description	0	8192 bytes	×	2 columns
partida	No description	6	24 kB	×	6 columns
jugador	No description	3	32 kB	×	6 columns
hotel	No description	3	32 kB	×	4 columns
arbitro	No description	2	32 kB	×	5 columns
sala	No description	3	24 kB	×	3 columns
psia	No description	3	24 kB	×	2 columns

Update rows

`update` lets you update rows. `update` will match all rows by default. You can update specific rows using horizontal filters, e.g. `eq`, `lt`, and `is`.

`update` will also return the replaced values for UPDATE.

[Learn more](#)

Delete rows

`delete` lets you delete rows. `delete` will match all rows by default, so remember to specify your filters!

[Learn more](#)

Subscribe to changes

Supabase provides realtime functionality and broadcasts database changes to authorized users depending on Row Level Security (RLS) policies.

[Learn more](#)

UPDATE MATCHING ROWS

```
const [ data, error ] = await supabase
  .from('pairs')
  .update({ other_column: 'otherValue' })
  .eq('some_column', 'someValue')
  .select()
```

DELETE MATCHING ROWS

```
const { error } = await supabase
  .from('pairs')
  .delete()
  .eq('some_column', 'someValue')
```

SUBSCRIBE TO ALL EVENTS

```
const pairs = supabase.channel('custom-all-channel')
.on(
  'postgres_changes',
  { event: '*', schema: 'public', table: 'pairs' },
  (payload) => {
    console.log('Change received!', payload)
  }
)
.subscribe()
```

GET get sala

[Open request](#) →

`https://db.rucifylgxlinvlwksdj.supabase.co/rest/v1/sala?select=*`

This is a POST request, submitting data to an API via the request body. This request submits JSON data, and the data is reflected in the response.

A successful POST request typically returns a `200 OK` or `201 Created` response code.

Request Headers

apikey	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJkdXBhYmFzIiwiaWF0IjoxNjU0MjM0MjE2fQ.
Authorization	bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJkdXBhYmFzIiwiaWF0IjoxNjU0MjM0MjE2fQ.

Query Params

`select` *

sala

DESCRIPTION

Click to edit.

COLUMN

numero

Required

TYPE

number

FORMAT

integer

DESCRIPTION

Click to edit.

COLUMN

capacidad

Optional

TYPE

number

FORMAT

integer

DESCRIPTION

Click to edit.

SELECT NUMERO

```
let { data: sala, error } = await supabase
  .from('sala')
  .select('numero')
```

SELECT CAPACIDAD

```
let { data: sala, error } = await supabase
  .from('sala')
  .select('capacidad')
```

hotelid

DESCRIPTION

Click to edit.

COLUMN

hotelid

Required

TYPE

number

FORMAT

integer

DESCRIPTION

Click to edit.

Read rows

To read rows in `sala`, use the `select` method.

[Learn more](#)

SELECT HOTELID

```
let { data: sala, error } = await supabase
  .from('sala')
  .select('hotelid')
```

READ ALL ROWS

```
let { data: sala, error } = await supabase
  .from('sala')
  .select('*')
```

READ SPECIFIC COLUMNS

```
let { data: sala, error } = await supabase
  .from('sala')
  .select('some_column,other_column')
```

READ REFERENCED TABLES

```
let { data: sala, error } = await supabase
  .from('sala')
  .select('
```

Filtering

Supabase provides a wide range of filters.

[Learn more](#)

WITH FILTERING

```
let { data, error } = await supabase
  .from('sala')
  .select('*')
  // filters
  .eq('column', 'Equal to')
  .gt('column', 'Greater than')
  .lt('column', 'Less than')
  .gte('column', 'Greater than or equal to')
  .lte('column', 'Less than or equal to')
  .like('column', '%CaseSensitive%')
  .ilike('column', '%CaseInsensitive%')
  .is('column', null)
  .in('column', ['Array', 'Values'])
  .neq('column', 'Not equal to')
  // Arrays
  .in('array_column', ['array', 'contains'])
  .on('array_column', ['contained', 'by'])
```

Insert rows

`insert` lets you insert into your tables. You can also insert in bulk and do UPSERT.

`insert` will also return the replaced values for UPSERT.

[Learn more](#)

INSERT A ROW

```
const { data, error } = await supabase
  .from('sala')
  .insert([
    { some_column: 'someValue', other_column: 'otherValue' },
  ])
  .select()
```

INSERT MANY ROWS

```
const { data, error } = await supabase
  .from('sala')
```

Update rows

`update` lets you update rows. `update` will match all rows by default. You can update specific rows using horizontal filters, e.g. `eq`, `lt`, and `is`.

`update` will also return the replaced values for UPDATE.

[Learn more](#)

UPDATE MATCHING ROWS

```
const { data, error } = await supabase
  .from('sala')
  .update({ other_column: 'otherValue' })
  .eq('some_column', 'someValue')
  .select()
```

Delete rows

`delete` lets you delete rows. `delete` will match all rows by default, so remember to specify your filters!

[Learn more](#)

DELETE MATCHING ROWS

```
const { error } = await supabase
  .from('sala')
  .delete()
  .eq('some_column', 'someValue')
```

Subscribe to changes

Supabase provides realtime functionality and broadcasts database changes to authorized users depending on Row Level Security (RLS) policies.

[Learn more](#)

SUBSCRIBE TO ALL EVENTS

```
const sala = supabase.channel('custom-all-channel')
const
  'postgres_changes',
  { event: '*', schema: 'public', table: 'sala' },
  (payload) => {
    console.log('Change received!', payload)
  }
)
.subscribe()
```


POST post partida

[Open request →](#)

https://db.rucfylvlglxlinvfwkxsdj.supabase.co/rest/v1/partida

This is a POST request, submitting data to an API via the request body. This request submits JSON data, and the data is reflected in the response.

A successful POST request typically returns a 200 OK or 201 Created response code.

Request Headers

apikey	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzI
Authorization	bearer eyJhbGciOiJIUzI1NiIsImtpZCI6Imtp1c2NVQ1INcjJFdDZYZEwiLCJ0e
Content-Type	application/json

Body raw (json)

json



```
{
  "numero": 3,
  "fecha": "2023-01-01",
  "numero_arbitro": 1,
  "numero_sala": 1,
  "numero_entradas": 25
}
```


POST post pais

[Open request→](#)

https://db.rucfyIgxlinvlwkxsdj.supabase.co/rest/v1/pais

This is a POST request, submitting data to an API via the request body. This request submits JSON data, and the data is reflected in the response.

A successful POST request typically returns a 200 OK or 201 Created response code.

Request Headers

apikey	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzI
Authorization	bearer eyJhbGciOiJIUzI1NiIsImtpZCI6Imt1c2NVQ1INcjJFdDZYZEwiLCJ0e
Content-Type	application/json

Body raw (json)

json

```
{
  "numero": 4,
  "nombre": "Turquia"
}
```


DELETE delete general

[Open request→](#)

https://db.rucifylgxlinvlwkxsdj.supabase.co/rest/v1/pais?numero=eq.3

This is a POST request, submitting data to an API via the request body. This request submits JSON data, and the data is reflected in the response.

A successful POST request typically returns a 200 OK or 201 Created response code.

Request Headers

apikey	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzI
Authorization	bearer eyJhbGciOiJIUzI1NiIsImtpZCI6Imlt1c2NVQ1INcjJFdDZYZEwiLCJ0e
Content-Type	application/json

Query Params

numero	eq.3
--------	------

Body raw (json)

json

```
{
  "nombre": "Polonia"
}
```

//solo cambias el nombre de la entidad el numero eq.'3' y en json pones que quieres cambiar

INTRODUCCIÓN

En el marco de los emocionantes campeonatos de ajedrez que se celebran anualmente en la Ciudad de México, surge la necesidad de crear un sistema integral de registro y gestión de participantes, alojamientos y partidas. Este proyecto se centra en la construcción de una sólida Base de Datos (BD) que aborde diversos aspectos clave para el adecuado desarrollo y seguimiento del torneo. A continuación, se presentan los elementos fundamentales a considerar:

Participantes y Árbitros:

En el campeonato, jugadores y árbitros desempeñan roles esenciales. Se requiere información detallada, como el número de asociado, nombre, país de origen y un número de teléfono de contacto obligatorio. Cabe destacar que los árbitros no pueden participar como jugadores y cada participante es enviado por un país específico.

Países:

La representación internacional se organiza mediante un sistema de numeración correlativa según el orden alfabético de los países. Se busca conocer el nombre de cada país y qué jugadores y árbitros han sido enviados por cada uno.

Partidas:

Cada enfrentamiento se identifica mediante un número único y se registran detalles cruciales como la fecha de celebración. Dos jugadores y un árbitro participan en cada partida, asegurando que el árbitro no pueda arbitrar a jugadores de su mismo país.

Alojamientos

Jugadores y árbitros son alojados en hoteles designados para el torneo. Se recopila información sobre el nombre del hotel, su dirección y número de teléfono.

Introducción

Salas y Hoteles:

Cada partida tiene lugar en una sala de uno de los hoteles, siendo fundamental conocer el número de entradas vendidas y la capacidad de cada sala. Un hotel puede disponer de varias salas, y se establece una relación entre hoteles y salas.

Análisis de Datos:

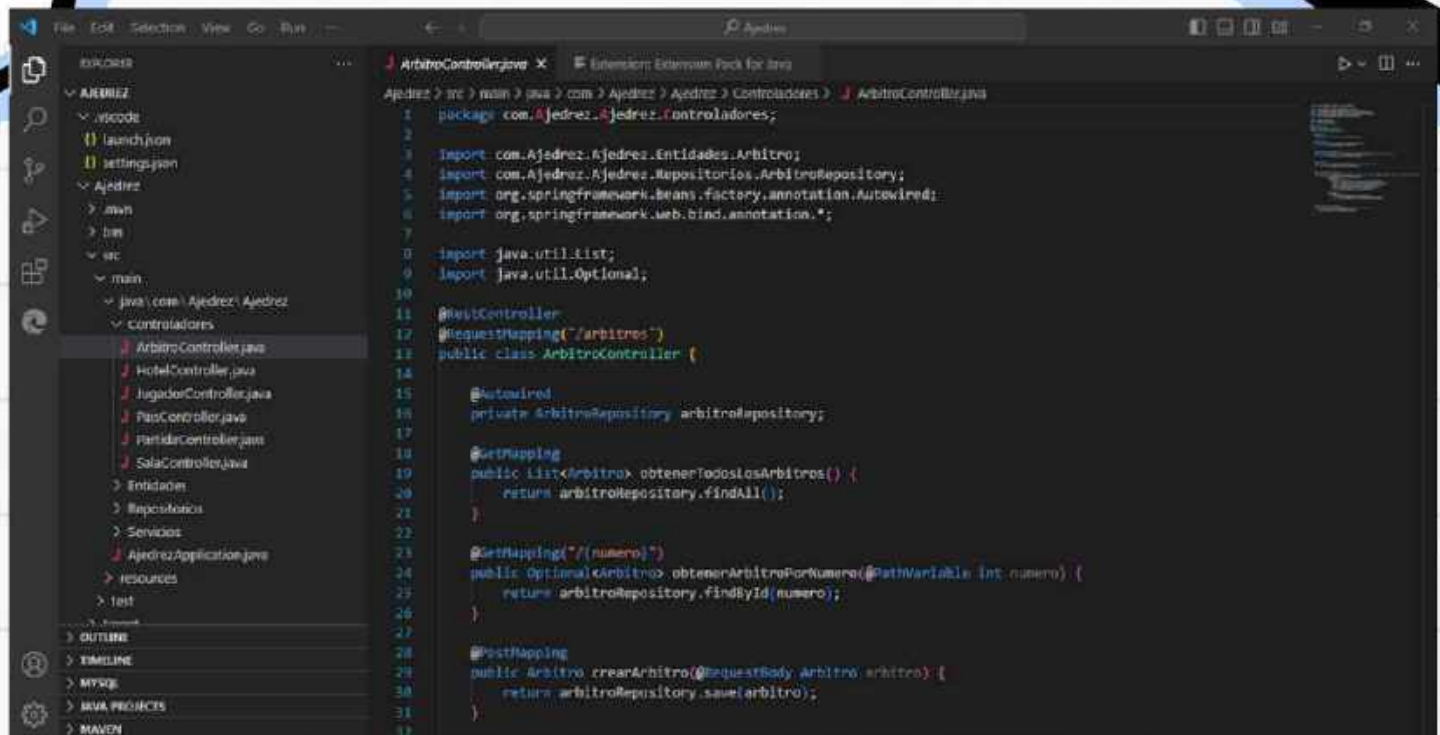
Para estructurar la BD, se definen entidades como Jugador, Arbitro, Partida, Hotel, Sala y País. Se asignan atributos específicos a cada entidad, destacando las claves primarias, llaves foráneas y las relaciones entre ellas. Además, se establecen suposiciones que simplifican el diseño, como la unicidad de árbitros por partida y la asignación de un participante por hotel.

En resumen, este proyecto busca ofrecer una solución eficiente y completa para la gestión de información en los campeonatos de ajedrez, garantizando un seguimiento detallado de participantes, alojamientos y partidas, contribuyendo así al éxito y organización de estos emocionantes eventos anuales en la Ciudad de México.

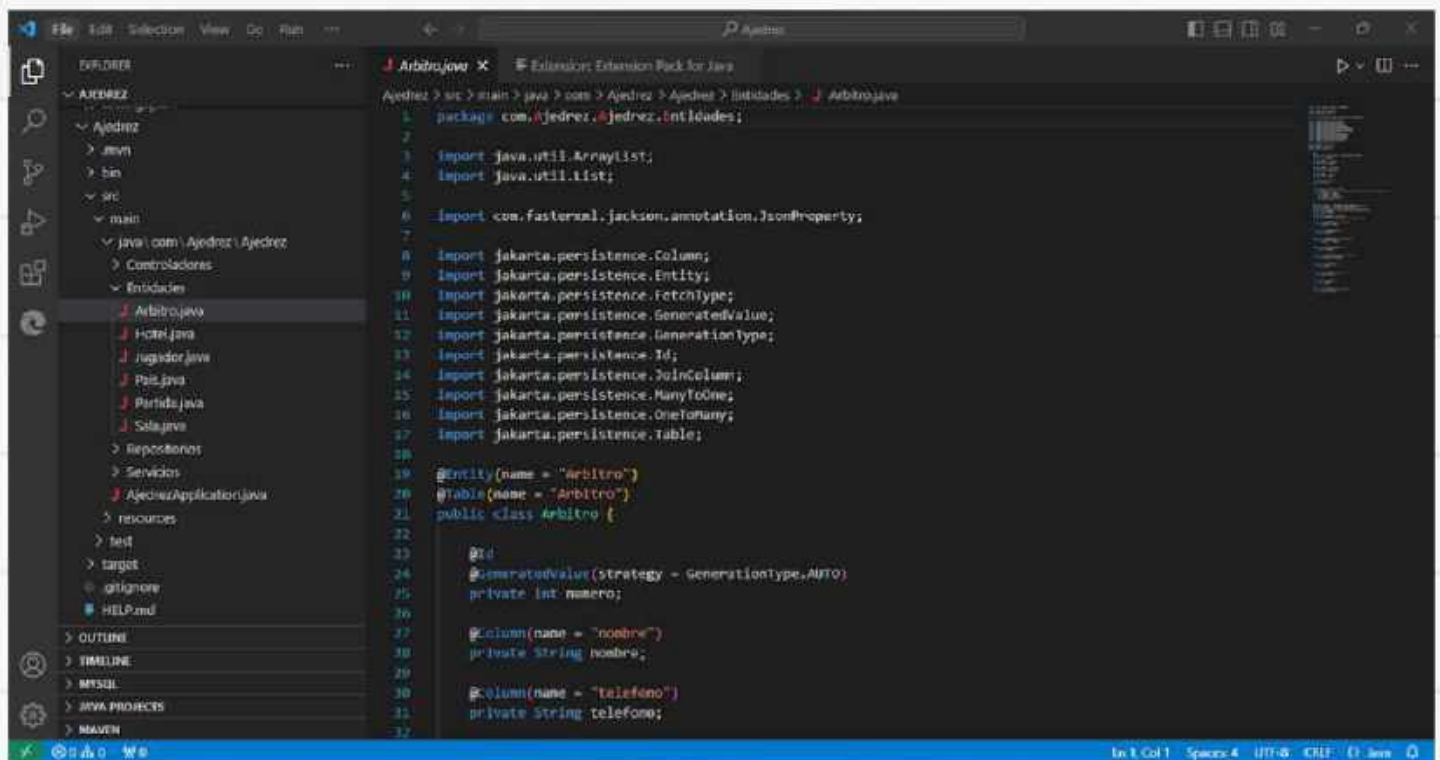
Contexto

En la Ciudad de México realizan campeonatos de ajedrez cada año y se desea llevar un registro de los participantes, alojamientos y partidas que intervendrán en este año. Para construir la BD hay que tomar en cuenta los siguientes aspectos:

1. En el campeonato participan jugadores y árbitros, de ambos se requiere conocer el número de asociado, nombre, país, teléfono de contacto, debe proporcionar uno al menos
2. Los árbitros no pueden participar como jugadores.
3. Los países envían al campeonato un conjunto de jugadores y árbitros, Cada jugador y árbitro es enviado por un único país.
4. Cada país se identifica por un número correlativo según su orden alfabético e interesa conocer además su nombre,
5. Cada partida se identifica por un número también se requiere la fecha, la juegan dos jugadores y la arbitra un árbitro. Hay que tener en cuenta que un árbitro no puede arbitrar a jugadores enviados por el mismo país que le ha enviado a él.
6. Todo participante participa en, al menos una partida.
7. Tanto jugadores como árbitros se alojan en uno de los hoteles en los que se desarrollan las partidas. De cada hotel se desea conocer el nombre, la dirección y el número de teléfono.
8. Cada partida se celebra en una de las salas de las que pueden disponer los hoteles. Se desea conocer el número de entradas vendidas en la sala para cada partida, de la sala interesa conocer su número, y capacidad.



```
ArbitroController.java
Ajedrez > src > main > java > com > Ajedrez > Ajedrez > Controladores > ArbitroController.java
1 package com.Ajedrez.Ajedrez.Controladores;
2
3 import com.Ajedrez.Ajedrez.Entidades.Arbitro;
4 import com.Ajedrez.Ajedrez.Repositorios.ArbitroRepository;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.web.bind.annotation.*;
7
8 import java.util.List;
9 import java.util.Optional;
10
11 @RestController
12 @RequestMapping("/arbitros")
13 public class ArbitroController {
14
15     @Autowired
16     private ArbitroRepository arbitroRepository;
17
18     @GetMapping
19     public List<Arbitro> obtenerTodosLosArbitros() {
20         return arbitroRepository.findAll();
21     }
22
23     @GetMapping("/{numero}")
24     public Optional<Arbitro> obtenerArbitroPorNumero(@PathVariable int numero) {
25         return arbitroRepository.findById(numero);
26     }
27
28     @PostMapping
29     public Arbitro crearArbitro(@RequestBody Arbitro arbitro) {
30         return arbitroRepository.save(arbitro);
31     }
32 }
```

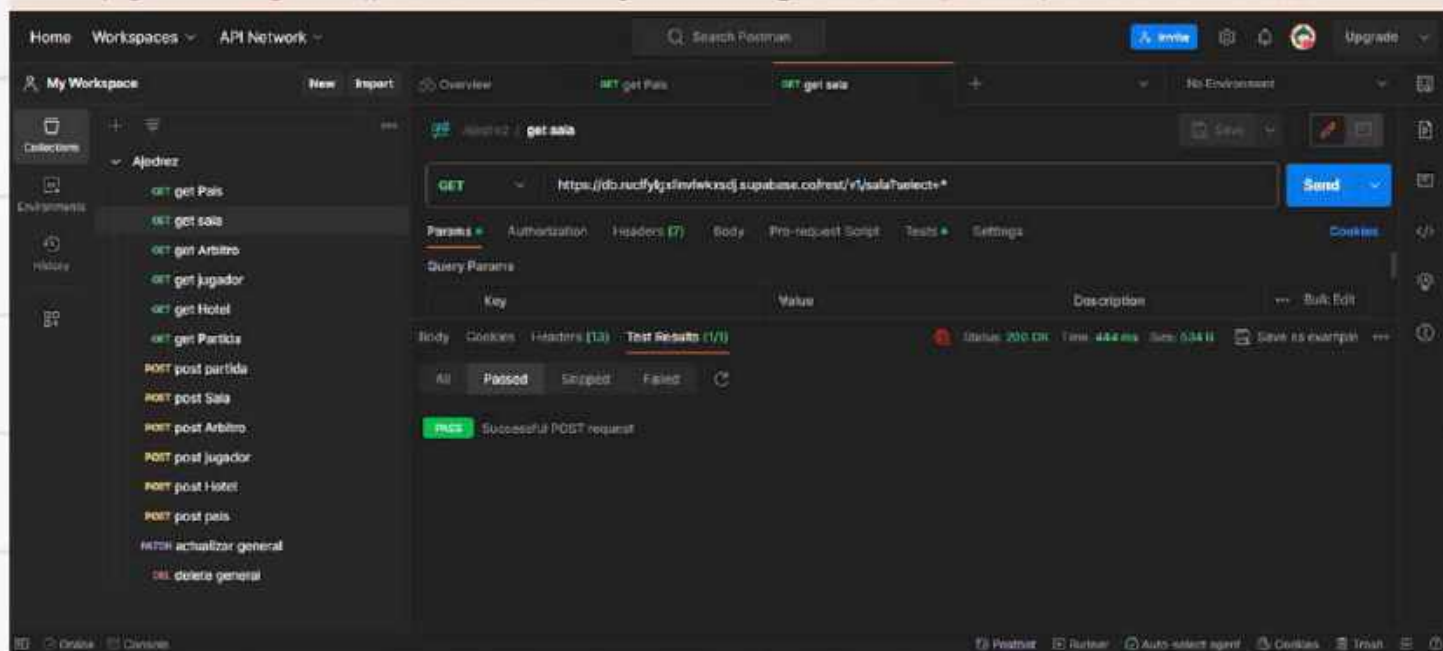
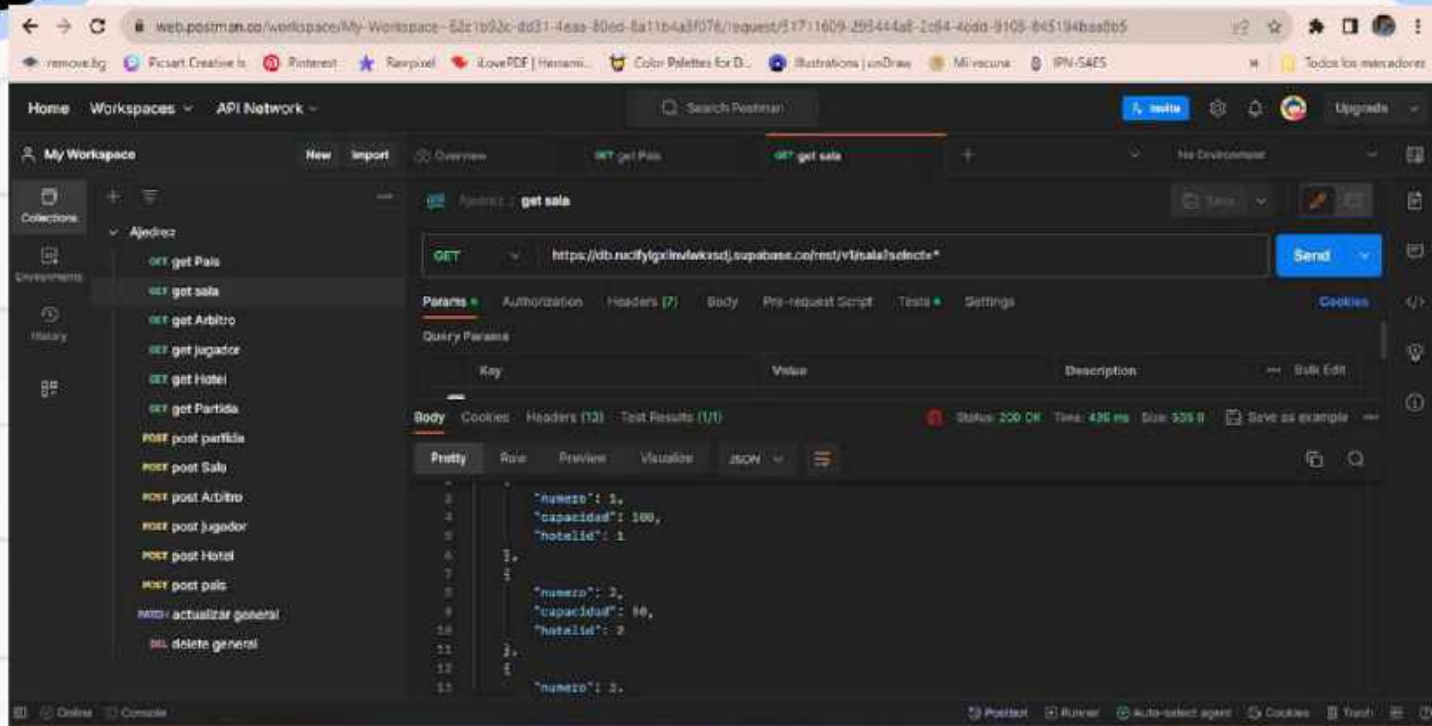


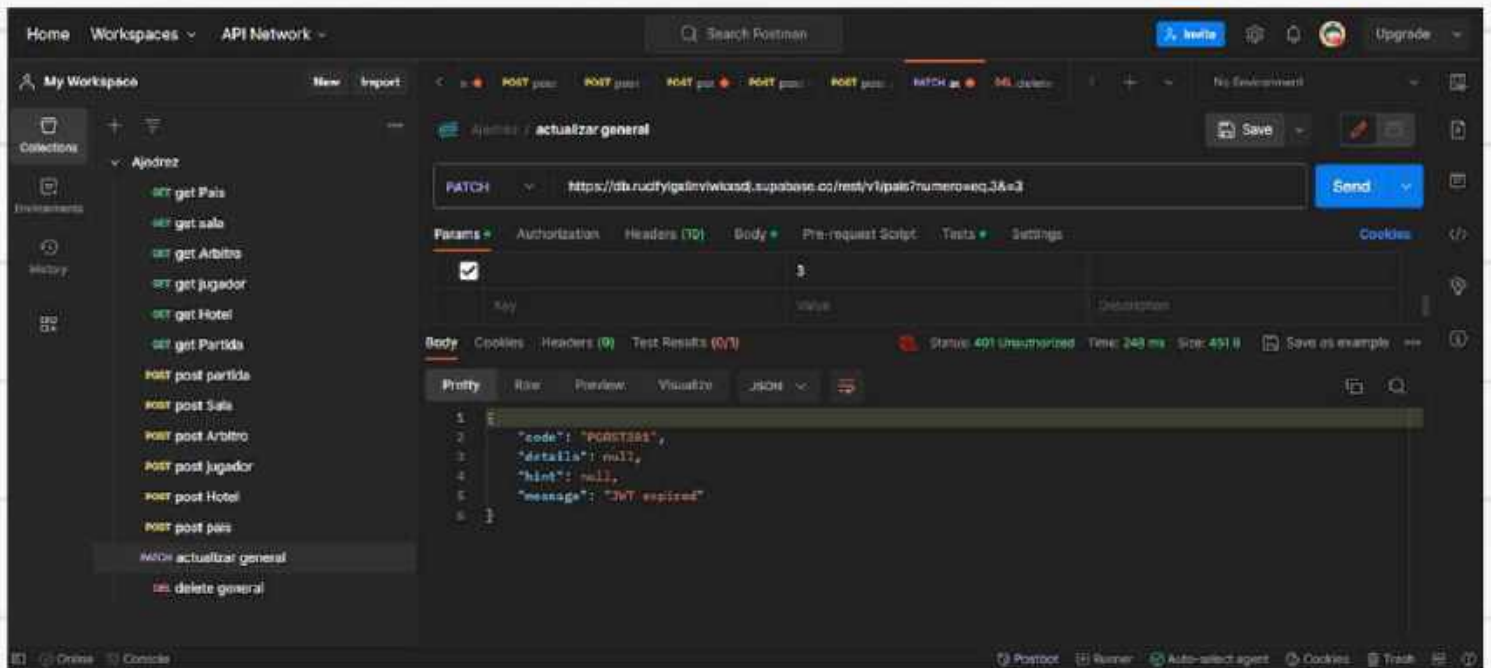
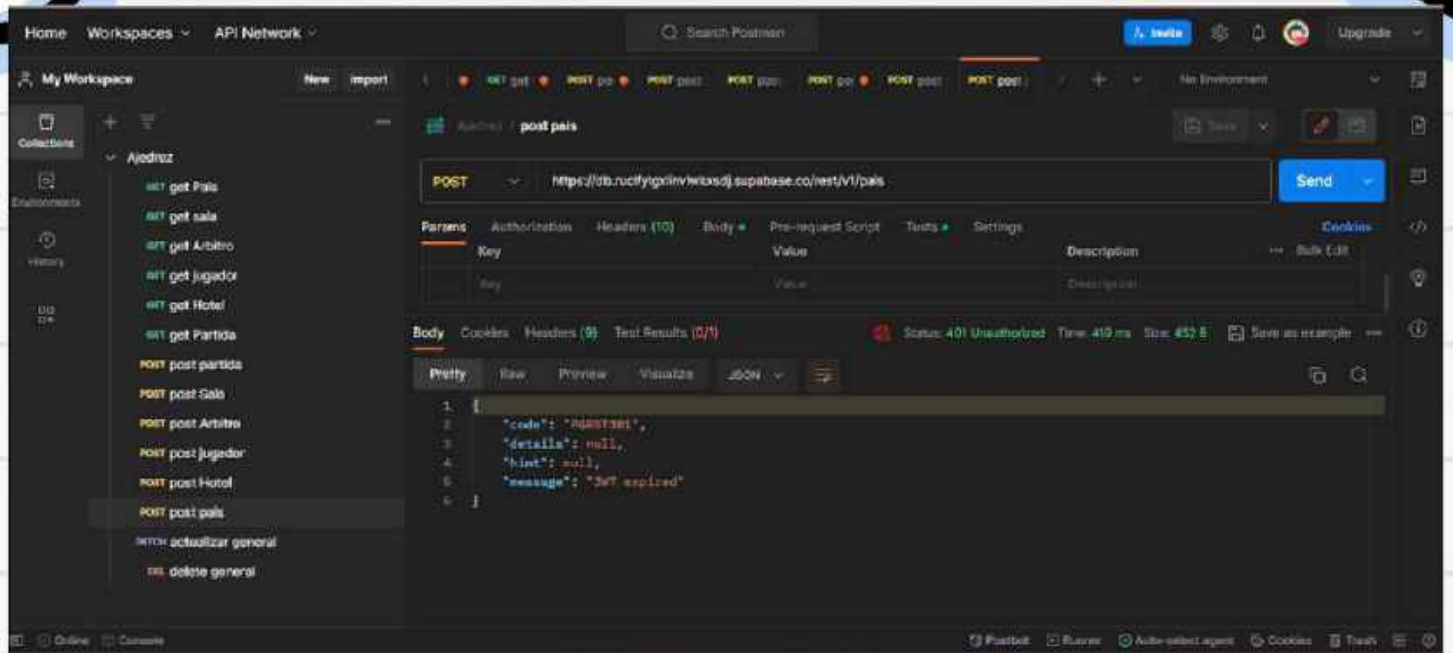
```
Arbitro.java
Ajedrez > src > main > java > com > Ajedrez > Ajedrez > Entidades > Arbitro.java
1 package com.Ajedrez.Ajedrez.Entidades;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import com.fasterxml.jackson.annotation.JsonProperty;
7
8 import jakarta.persistence.Column;
9 import jakarta.persistence.Entity;
10 import jakarta.persistence.FetchType;
11 import jakarta.persistence.GeneratedValue;
12 import jakarta.persistence.GenerationType;
13 import jakarta.persistence.Id;
14 import jakarta.persistence.JoinColumn;
15 import jakarta.persistence.ManyToOne;
16 import jakarta.persistence.OneToMany;
17 import jakarta.persistence.Table;
18
19 @Entity(name = "Arbitro")
20 @Table(name = "Arbitro")
21 public class Arbitro {
22
23     @Id
24     @GeneratedValue(strategy = GenerationType.AUTO)
25     private int numero;
26
27     @Column(name = "nombre")
28     private String nombre;
29
30     @Column(name = "telefono")
31     private String telefono;
32 }
```

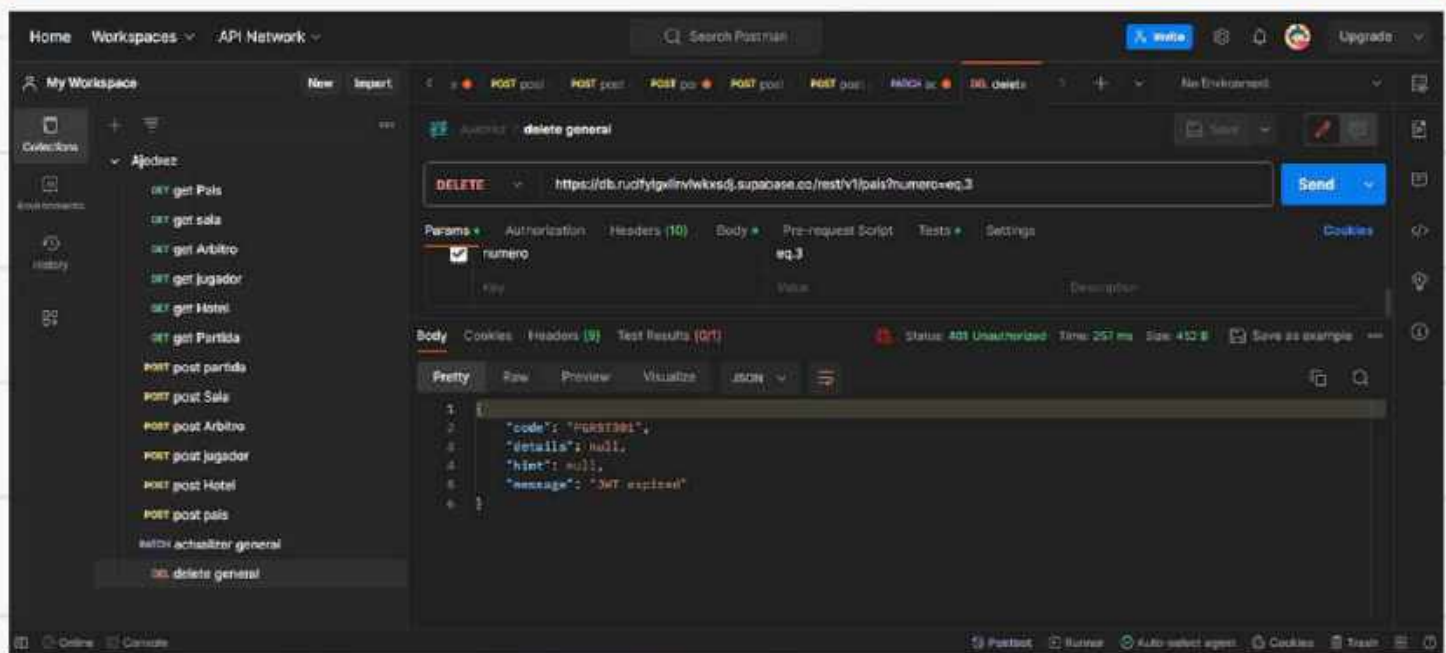


```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <parent>
6     <groupId>org.springframework.boot</groupId>
7     <artifactId>spring-boot-starter-parent</artifactId>
8     <version>3.2.0</version>
9     <relativePath><!-- lookup parent from repository -->
10   </parent>
11   <groupId>com.Ajedrez</groupId>
12   <artifactId>Ajedrez</artifactId>
13   <version>0.0.1-SNAPSHOT</version>
14   <name>Ajedrez</name>
15   <description>Proyecto de TW</description>
16   <properties>
17     <jdk.version>17</jdk.version>
18   </properties>
19   <dependencies>
20     <dependency>
21       <groupId>org.springframework.boot</groupId>
22       <artifactId>spring-boot-starter-data-jpa</artifactId>
23     </dependency>
24     <dependency>
25       <groupId>org.springframework.boot</groupId>
26       <artifactId>spring-boot-starter-web</artifactId>
27     </dependency>
28     <dependency>
29       <groupId>org.springframework.boot</groupId>
30       <artifactId>spring-boot-devtools</artifactId>
31       <scope>runtime</scope>
32   </dependencies>
```

```
1 package com.Ajedrez.Ajedrez.Controladores;
2
3 import com.Ajedrez.Ajedrez.Entidades.Pais;
4 import com.Ajedrez.Ajedrez.Repositorios.PaisRepository;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.web.bind.annotation.*;
7
8 import java.util.List;
9 import java.util.Optional;
10
11 @RestController
12 @RequestMapping("/países")
13 public class PaisController {
14
15     @Autowired
16     private PaisRepository paisRepository;
17
18     @GetMapping
19     public List<Pais> obtenerTodosLosPaíses() {
20         return paisRepository.findAll();
21     }
22
23     @GetMapping("/{numero}")
24     public Optional<Pais> obtenerPaisPorNumero(@PathVariable int numero) {
25         return paisRepository.findById(numero);
26     }
27
28     @PostMapping
29     public Pais crearPais(@RequestBody Pais pais) {
30         return paisRepository.save(pais);
31     }
32 }
```







Postman y supabase crea un entorno completo y eficaz para el desarrollo de aplicaiones, estas nos permiten simplificar el diseño y prueba de APIs, en este caso Supabase nos proporciona una base de postgresql sin necesidad de un servidor, lo que agiliza el proceso de construccion de aplicaciones. Estas herramientas nos brindan ventajas, como el poder realizar pruebas, documentacion clara y otro tipo de cosas.

Al final podemos visualizar que nuestra API funciona de forma correcta, lo que se ejecuta por medio de Postman que nos deja realizar pruebas detalladas de las tablas que implementamos.