

## **Parcial**

David Santiago Noguera Perez - 827823

Corporación Universitaria Minuto de Dios

Ing. William Alexander Matallana Porras

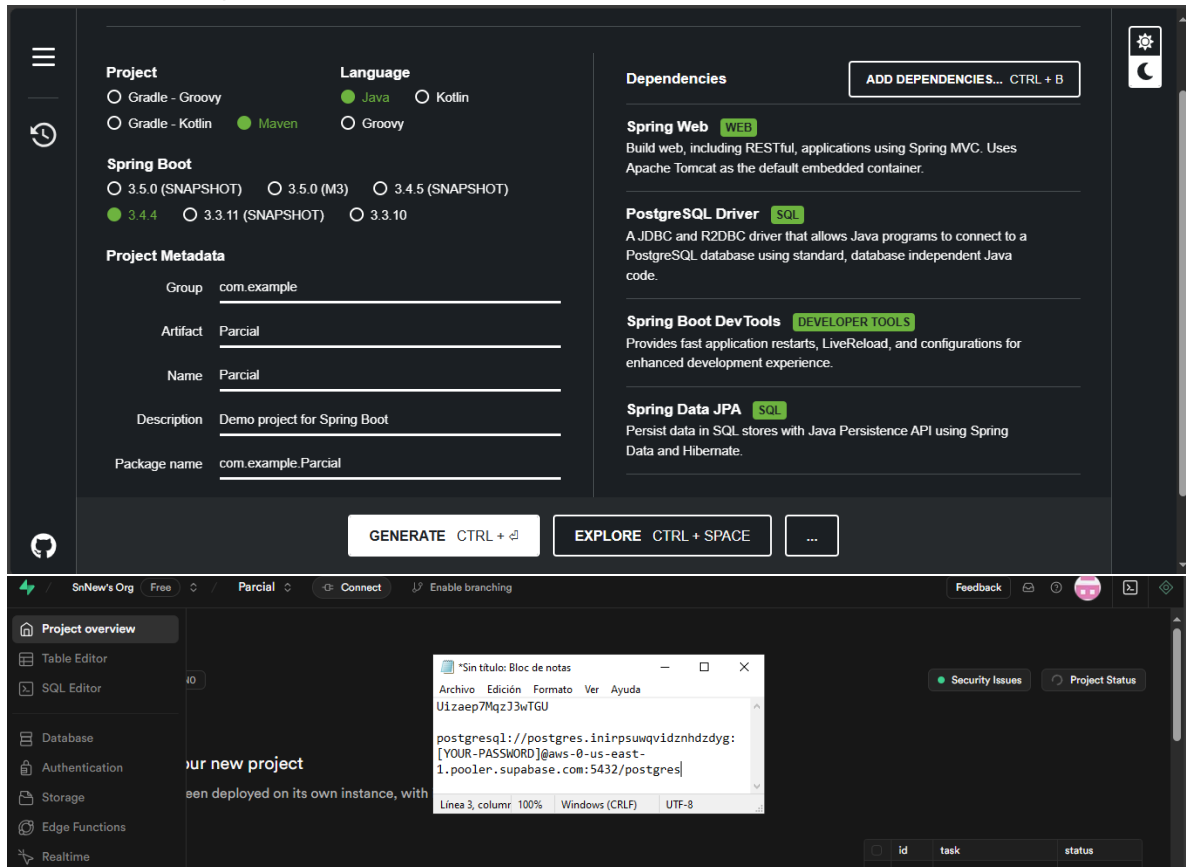
Arquitectura de software

2025

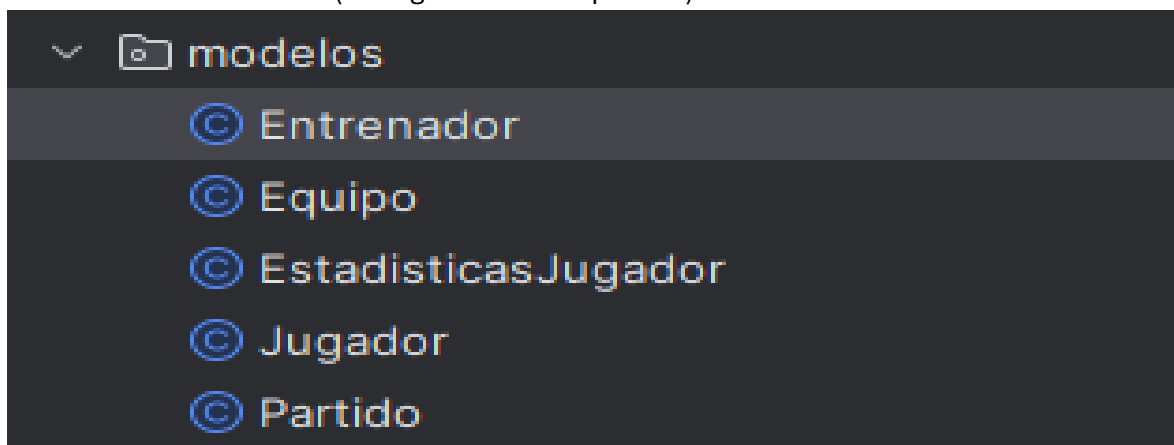
Desarrollar una API REST en Spring Boot que permita gestionar la información de un equipo de fútbol: jugadores, entrenadores, partidos y estadísticas. Se documentará con Swagger y usará Supabase como base de datos PostgreSQL.

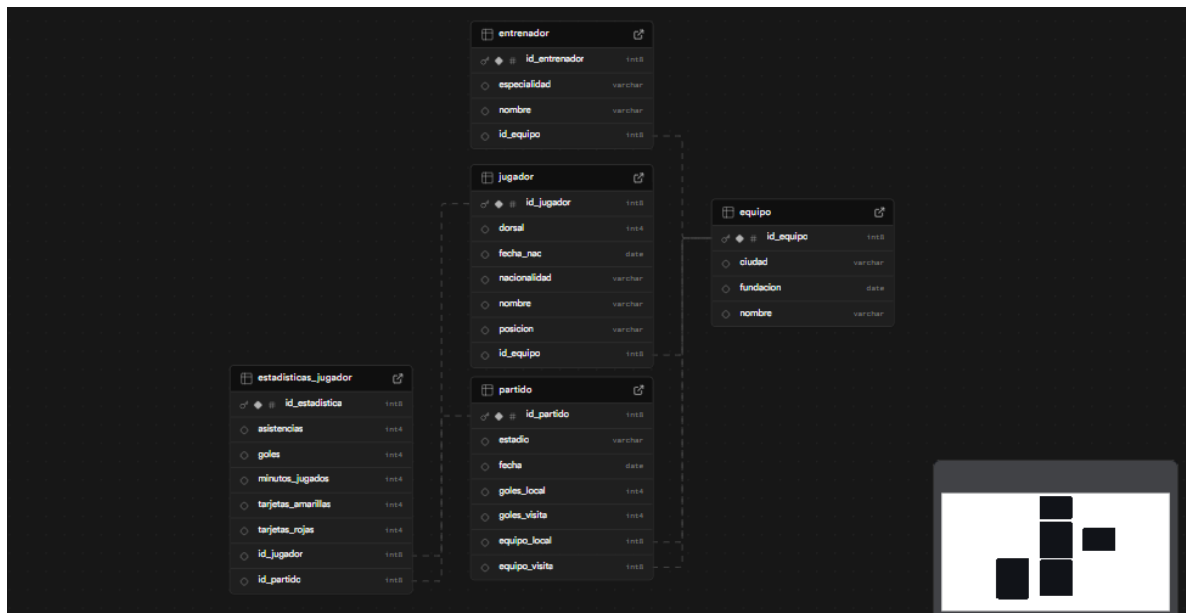
## Acciones y Etapas del Proyecto

### 1. Conexión con Supabase



### 2. Crear el Modelo de Datos (Entregar modelo Supabase)





```

1  INSERT INTO equipo (id_equipo, nombre, ciudad, fundacion)
2  SELECT
3    gs AS id_equipo,
4    'Equipo ' || gs,
5    'Ciudad ' || gs,
6    DATE '1980-01-01' + (random() * interval '40 years')
7  FROM generate_series(1, 100) gs;
8
9  INSERT INTO entrenador (id_entrenador, nombre, especialidad, id_equipo)
10 SELECT
11   gs AS id_entrenador,
12   'Entrenador ' || gs,
13   CASE WHEN gs % 3 = 0 THEN 'Ofensivo' WHEN gs % 3 = 1 THEN 'Defensivo' ELSE 'Físico' END,
14   (random() * 99 + 1)::int
15 FROM generate_series(1, 100) gs;
16
17 INSERT INTO jugador (id_jugador, nombre, posicion, dorsal, fecha_nac, nacionalidad, id_equipo)
18 SELECT
19   gs AS id_jugador,
20   'Jugador ' || gs,
21   CASE WHEN gs % 3 = 0 THEN 'Delantero' WHEN gs % 3 = 1 THEN 'Centrocampista' ELSE 'Defensa' END,
22   (random() * 30 + 1)::int,
23   DATE '1985-01-01' + (random() * interval '20 years'),
24   'Nacionalidad ' || gs,
  
```

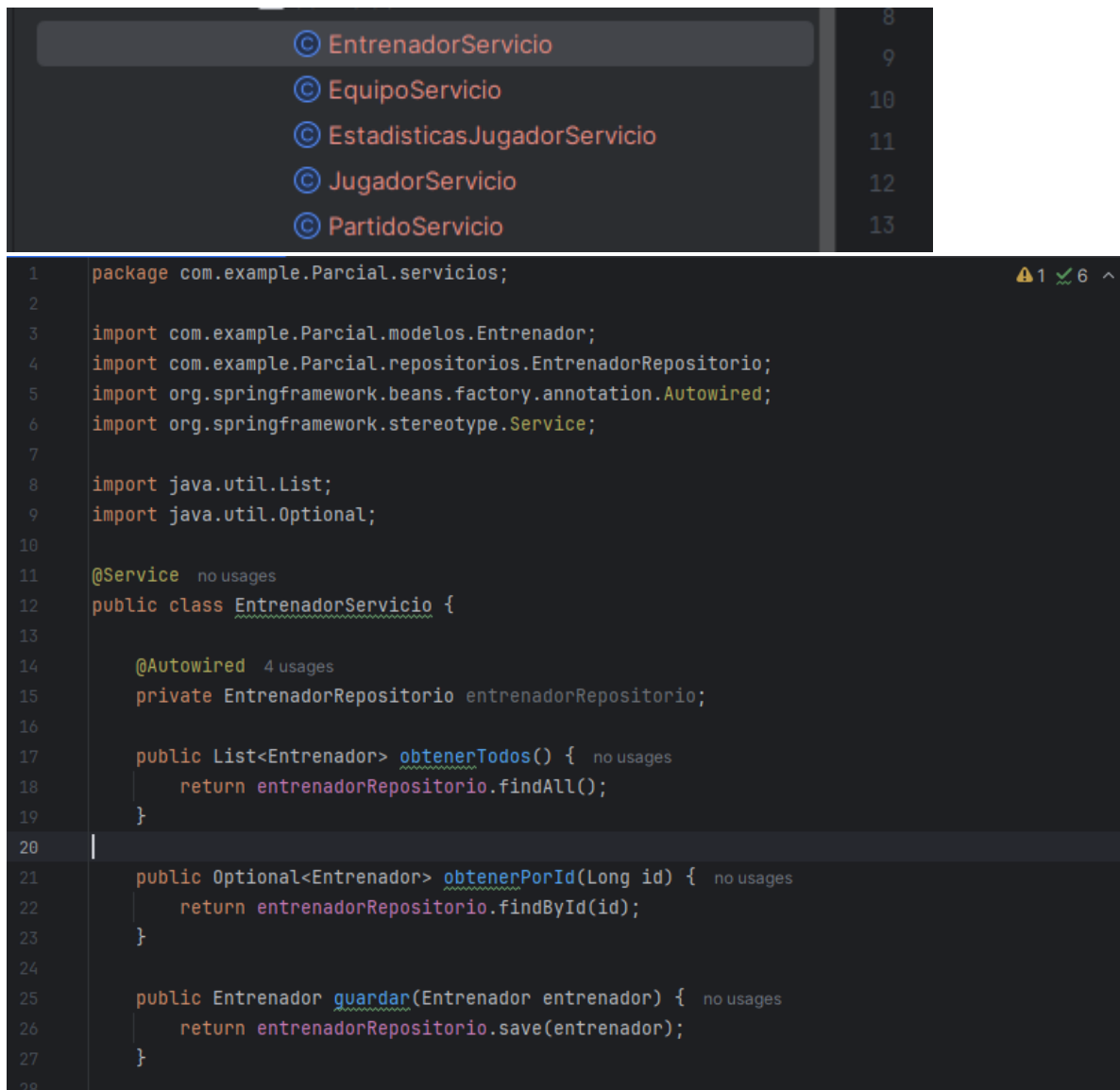
### 3. Crear los Repositorios

```

v repositórios
  EntrenadorRepositorio
  EquipoRepositorio
  EstadísticasJugadorRepositorio
  JugadorRepositorio
  PartidoRepositorio

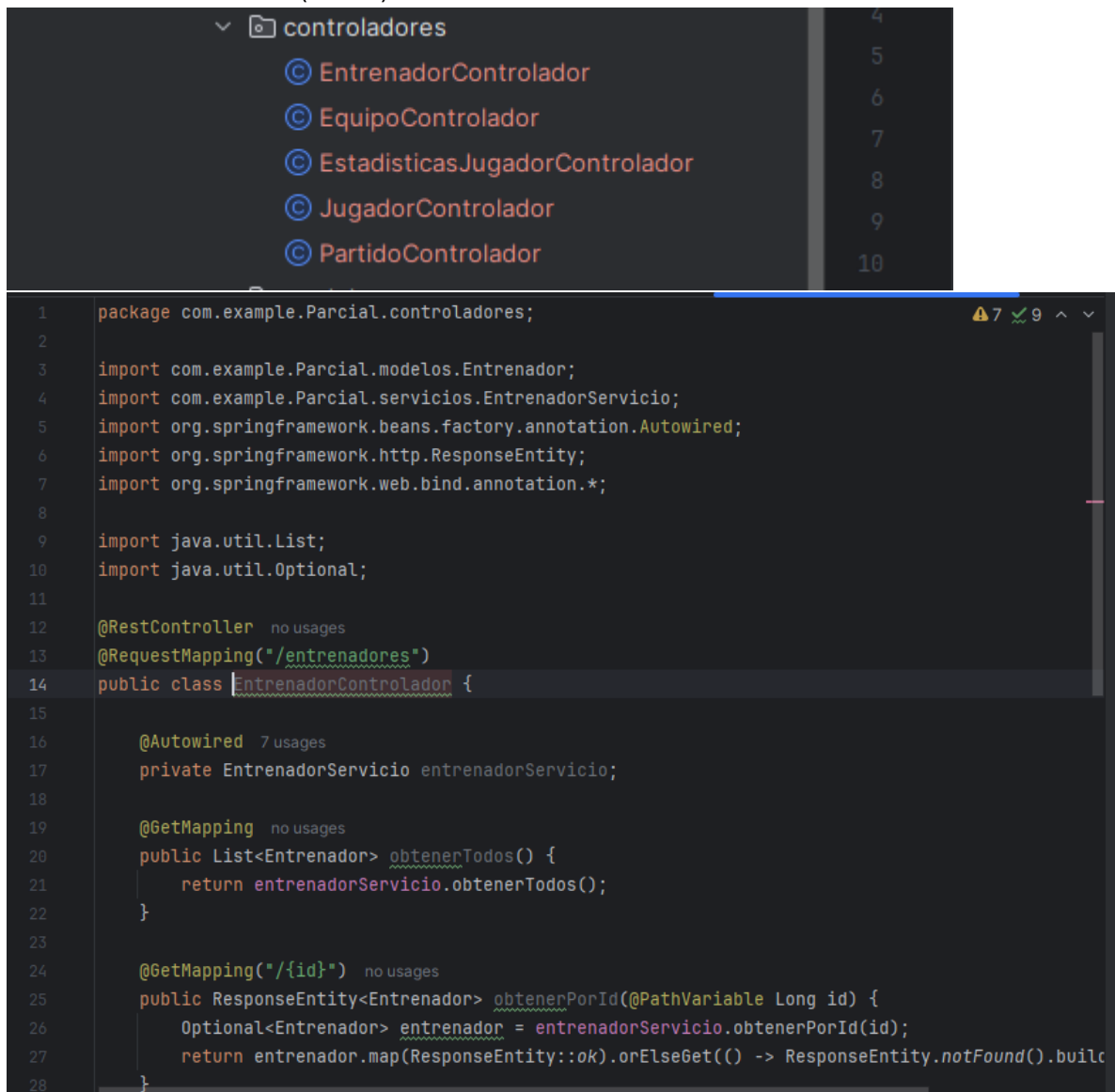
1 package com.example.Parcial.repositorios;
2
3 import com.example.Parcial.modelos.EstadísticasJugador;
4 import org.springframework.data.jpa.repository.JpaRepository;
5
6 public interface EstadísticasJugadorRepositorio extends JpaRepository<EstadísticasJugador, Integer> {
7 }
8
  
```

#### 4. Crear los Servicios



```
1 package com.example.Parcial.servicios;
2
3 import com.example.Parcial.modelos.Entrenador;
4 import com.example.Parcial.repositorios.EntrenadorRepositorio;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.stereotype.Service;
7
8 import java.util.List;
9 import java.util.Optional;
10
11 @Service no usages
12 public class EntrenadorServicio {
13
14     @Autowired 4 usages
15     private EntrenadorRepositorio entrenadorRepositorio;
16
17     public List<Entrenador> obtenerTodos() { no usages
18         return entrenadorRepositorio.findAll();
19     }
20
21     public Optional<Entrenador> obtenerPorId(Long id) { no usages
22         return entrenadorRepositorio.findById(id);
23     }
24
25     public Entrenador guardar(Entrenador entrenador) { no usages
26         return entrenadorRepositorio.save(entrenador);
27     }
28 }
```

## 5. Crear los Controladores (CRUD)

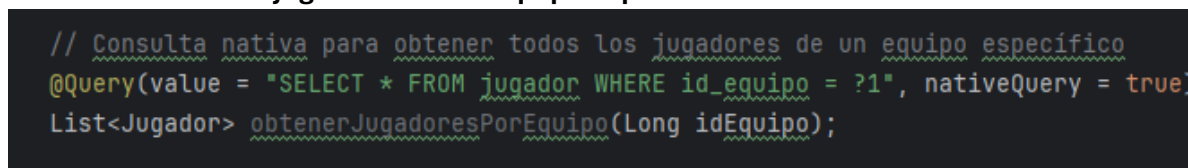


```
1 package com.example.Parcial.controladores;
2
3 import com.example.Parcial.modelos.Entrenador;
4 import com.example.Parcial.servicios.EntrenadorServicio;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.http.ResponseEntity;
7 import org.springframework.web.bind.annotation.*;
8
9 import java.util.List;
10 import java.util.Optional;
11
12 @RestController
13 @RequestMapping("/entrenadores")
14 public class EntrenadorControlador {
15
16     @Autowired
17     private EntrenadorServicio entrenadorServicio;
18
19     @GetMapping
20     public List<Entrenador> obtenerTodos() {
21         return entrenadorServicio.obtenerTodos();
22     }
23
24     @GetMapping("/{id}")
25     public ResponseEntity<Entrenador> obtenerPorId(@PathVariable Long id) {
26         Optional<Entrenador> entrenador = entrenadorServicio.obtenerPorId(id);
27         return entrenador.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.notFound().build());
28     }
29 }
```

## 6. Documentar la API con Swagger

## 7. Consultas nativas

### • Obtener todos los jugadores de un equipo específico



```
// Consulta nativa para obtener todos los jugadores de un equipo específico
@Query(value = "SELECT * FROM jugador WHERE id_equipo = ?1", nativeQuery = true)
List<Jugador> obtenerJugadoresPorEquipo(Long idEquipo);
```

Busca **todos los jugadores** que pertenezcan a un equipo específico. El `id_equipo` se refiere al equipo al que está asociado cada jugador. Es un **parámetro posicional**. `?1` representa el primer argumento que le pasas al método del repositorio. Si pasas por ejemplo 5, busca los jugadores cuyo `id_equipo` sea 5.

- **Obtener los jugadores que han marcado más de X goles**

```
15 // Consulta nativa para obtener los jugadores que han marcado más de X goles
16 @Query(value = "SELECT * FROM jugador WHERE goles > ?1", nativeQuery = true) no usages
17 List<Jugador> obtenerJugadoresConMasGoles(int goles);
18 }
```

Busca a todos los jugadores que han marcado más goles que un valor que tú determines (por ejemplo, todos los que han marcado más de 10 goles). Compara la columna goles de la tabla jugador con el número que le pasas como argumento. Si el jugador tiene más goles que ese número, se incluye en el resultado.

- **Obtener el número total de goles marcados por un equipo en todos sus**

**partidos**

```
1 // Consulta nativa para obtener el número total de goles marcados por un equipo
2 @Query(value = "SELECT SUM(p.goles_equipo_local + p.goles_equipo_visitante) FROM partido p WHERE p.id_equipo_local = ?1 OR p.id_equipo_visitante = ?1", nativeQuery = true)
3 int obtenerTotalGolesEquipo(Long idEquipo);
4 }
```

Suma todos los goles marcados en partidos donde el equipo participó, ya sea como local o visitante.

- **Obtener los resultados de todos los partidos indicando los nombres de**

**los equipos**

```
11 // Consulta nativa para obtener los resultados de todos los partidos indicando los nombres de los equipos
12 @Query(value = "SELECT p.fecha, e1.nombre AS equipo_local, e2.nombre AS equipo_visitante, p.goles_equipo_local, p.goles_equipo_visitante " +
13 "FROM partido p " +
14 "JOIN equipo e1 ON p.id_equipo_local = e1.id_equipo " +
15 "JOIN equipo e2 ON p.id_equipo_visitante = e2.id_equipo", nativeQuery = true)
16 List<Object[]> obtenerResultadosPartidos();
17 }
18 }
```

Muestra:

- la fecha del partido,
- el nombre del equipo local,
- el nombre del equipo visitante,
- los goles que anotó cada uno.

**¿Cómo funciona?**

- JOIN equipo e1 ...: busca el nombre del equipo local.
- JOIN equipo e2 ...: busca el nombre del equipo visitante.
- Se traen los goles y la fecha desde la tabla partido.

## Requisitos para la Entrega

### 1. Implementar cada consulta nativa en un método de repositorio

- Usar @Query con nativeQuery = true en el Repository.

### 2. Exponer las consultas a través del controlador (@RestController)

- Pueden usar métodos GET o POST según el tipo de consulta.

### 3. Probar cada consulta en Postman

- Usar la URL correcta (http://localhost:8080/...).
- Enviar parámetros (si aplica) por path o query.
- Mostrar la respuesta que entrega el backend.

### 4. Entregar capturas de pantalla

- Mostrar la solicitud con su método, parámetros y resultado.
- Si es posible, exportar la colección en formato .json