

Corporación Universitaria Minuto De Dios

Parcial 2

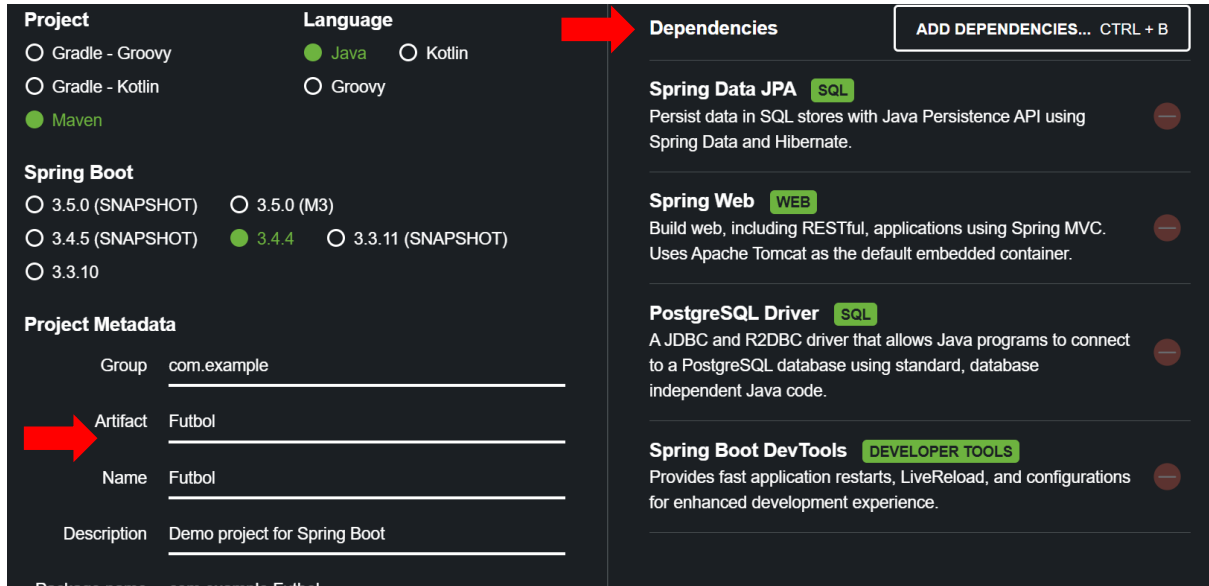
Juan David Bustos Castro

Arquitectura De Software

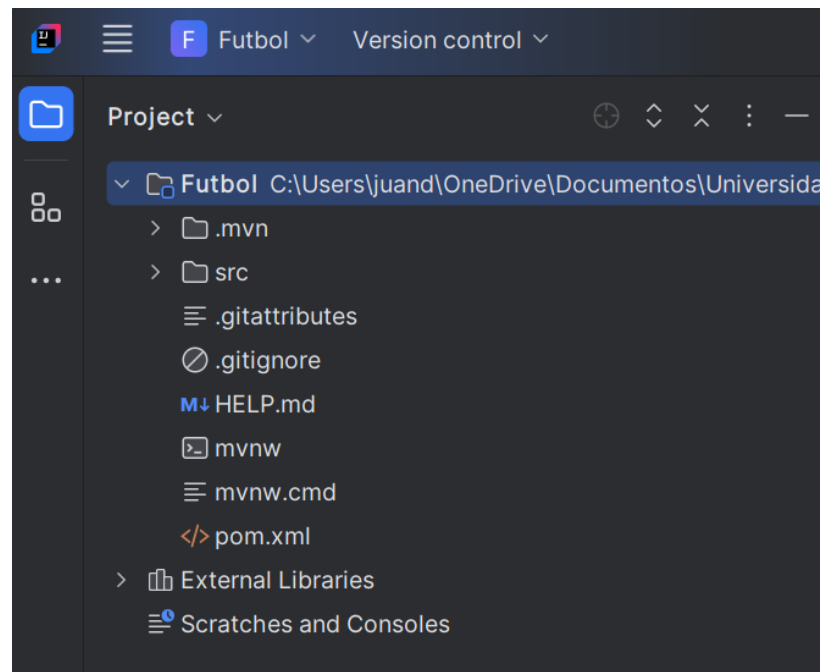
2025

Documentacion

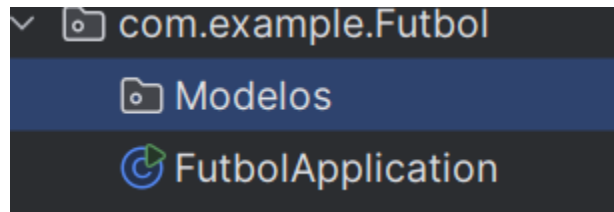
1. Lo primero que haremos será dirigirnos a Spring inicializr y le daremos el nombre del proyecto y agregaremos las dependencias que necesitaremos



2. Ahora lo abrimos desde intelliij



3. Ahora creamos la carpeta Modelos para comenzar con la estructuración del proyecto



4. Crearemos el primer Modelo que será Equipo y le agregamos las relaciones que nos piden en el proyecto

EQUIPO:

- Tiene datos básicos: nombre, ciudad y fecha de fundación
- Un equipo es papá de muchos jugadores, entrenadores, y de los partidos donde juega

```
@Entity no usages
@Table(name = "equipo")
public class Equipo {

    @Id 4 usages
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer idEquipo;

    private String nombre; 4 usages

    private String ciudad; 4 usages

    private LocalDate fundacion; 4 usages

    @OneToMany(mappedBy = "equipo", cascade = CascadeType.ALL) 4 usages
    private List<Jugador> jugadores;

    @OneToMany(mappedBy = "equipo", cascade = CascadeType.ALL) 4 usages
    private List<Entrenador> entrenadores;

    @OneToMany(mappedBy = "equipoLocal", cascade = CascadeType.ALL) 4 usages
    private List<Partido> partidosLocal;

    @OneToMany(mappedBy = "equipoVisita", cascade = CascadeType.ALL) 4 usages
    private List<Partido> partidosVisita;
```

5. Ahora vamos con el modelo jugador y también generamos las relaciones

Jugador:

- Guarda: nombre, posición, dorsal, fecha de nacimiento y nacionalidad
- Está amarrado a un equipo (cada jugador juega en un equipo)
- Y además tiene varias estadísticas (cuántos goles hizo, minutos jugados, etc.) en diferentes partidos
- El jugador es hijo de un equipo y tiene su propio historial de partidos

```
@ManyToOne no usages
@JoinColumn(name = "id_equipo", nullable = false)
private Equipo equipo;
```

6. Seguimos con el modelo Entrenador y generamos la relación que nos piden

Entrenador:

- Guarda: nombre y especialidad
- Cada entrenador trabaja para un equipo
- El entrenador es como otro hijo más del equipo, solo que no juega, dirige

7. Continuamos con el modelo Partido y generamos las relaciones

Partido:

- Guarda: fecha, estadio, goles local y visita
- Tiene relación con dos equipos:
 - equipoLocal (el que juega en casa)
 - equipoVisita (el que viene de visita)
- También guarda las estadísticas de cada jugador que jugó ese partido
- Cada partido sabe quién jugó, dónde y cómo quedaron

```

@ManyToOne no usages
@JoinColumn(name = "equipo_local", nullable = false)
private Equipo equipoLocal;

@ManyToOne no usages
@JoinColumn(name = "equipo_visita", nullable = false)
private Equipo equipoVisita;

private Integer golesLocal; no usages

private Integer golesVisita; no usages

@OneToMany(mappedBy = "partido", cascade = CascadeType.ALL) no usages
private List<EstadisticasJugador> estadisticas;

```

8. Por último generamos la clase EstadisticasJugador y hacemos las relaciones

EstadisticasJugador:

- Guarda: minutos jugados, goles, asistencias, tarjetas.
- Se conecta a:
 - Un jugador (quién jugó)
 - Un partido (dónde lo jugó).
- Aquí se guardan las estadísticas de cada jugador por partido

```

@ManyToOne no usages
@JoinColumn(name = "id_jugador", nullable = false)
private Jugador jugador;

@ManyToOne no usages
@JoinColumn(name = "id_partido", nullable = false)
private Partido partido;

private Integer minutosJugados; no usages

private Integer goles; no usages

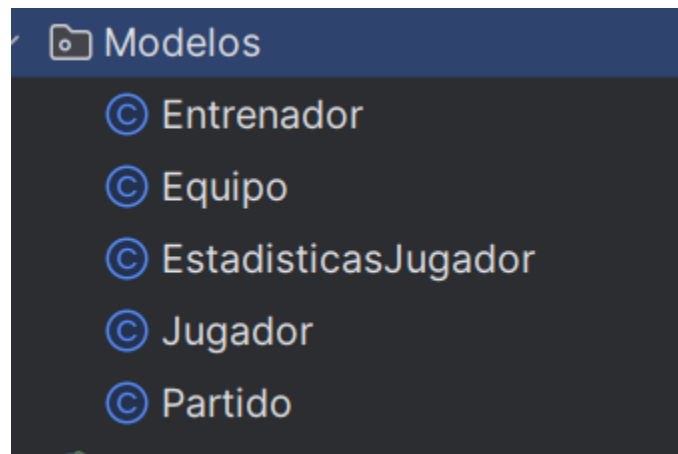
private Integer asistencias; no usages

private Integer tarjetasAmarillas; no usages

private Integer tarjetasRojas; no usages

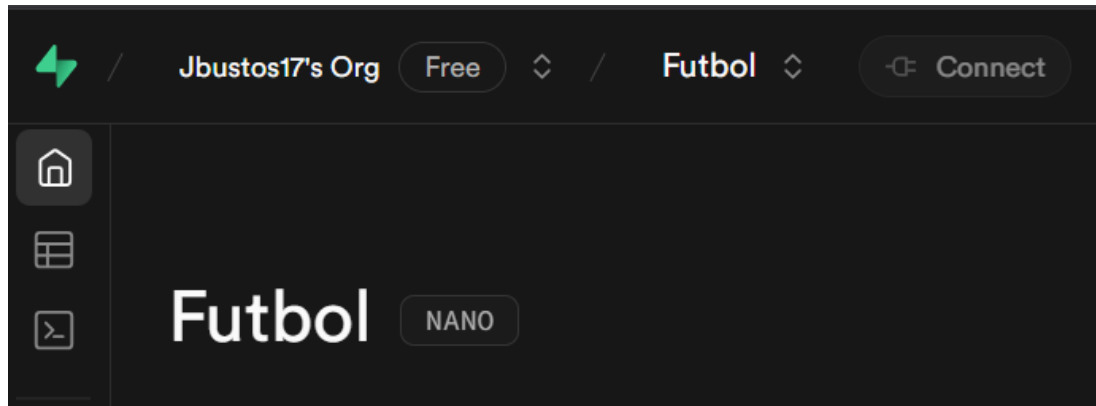
```

9. Después de crear todos los modelos con sus atributos, relaciones, constructores, getter y setters y el toString iremos a la parte de conectarlo con Supabase

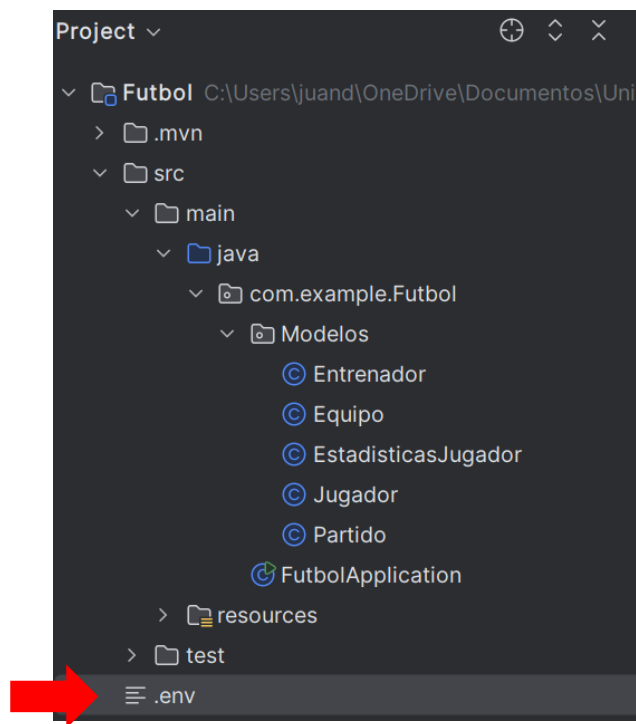


10. Así que nos dirigimos a supabase y creamos un nuevo proyecto llamado Futbol

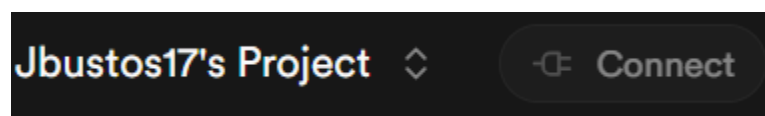
Conexión con Supabase



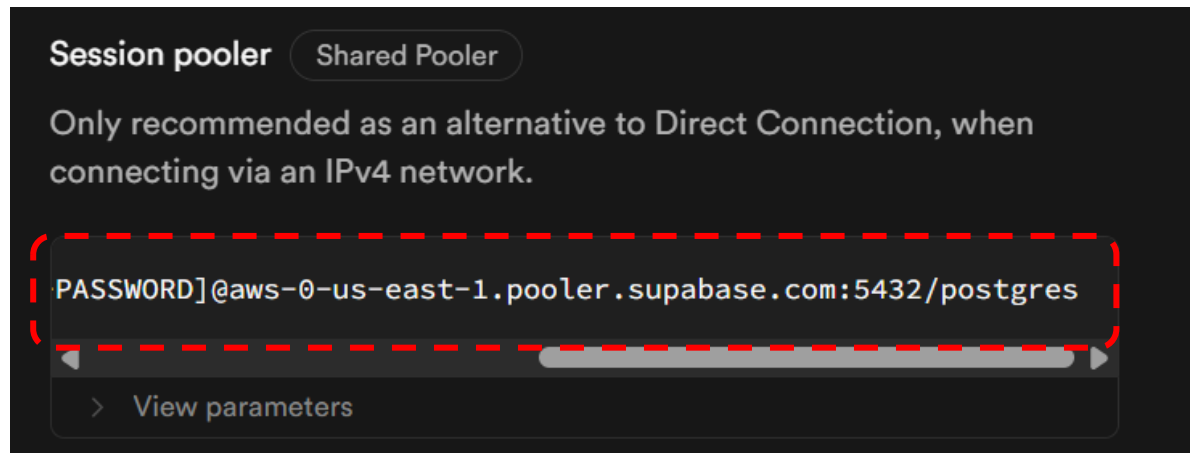
11. Ahora creamos un archivo file en nuestro proyecto



12. Volvemos a Supabase y le damos en connect



13. Copiamos el siguiente enlace



14. Ahora lo colocaremos de la siguiente forma en el .env

```
postgresql://postgres.udisjldmfcilexplbxfk:[YOUR-PASSWORD]@aws-0-us-east-1.pooler.supabase.com:5432/postgres
```

15. La parte encerrada en rojo será nuestro DB_URL, esto lo colocaremos de la siguiente forma:

DB_URL=jdbc:postgresql://aws-0-us-east-1.pooler.supabase.com:5432/postgres

16. Ahora haremos el de DB_USERNAME

```
postgresql://postgres.udisjldmfcilexplbxfk [YOUR-PASSWORD]@aws-0-us-east-1.pooler.supabase.com:5432/postgres
```

17. En el .env quedara de la siguiente forma:

DB_USERNAME=postgres.udisjldmfcilexplbxfk

18. Y por último pondremos la contraseña que pusimos al crear el proyecto, al final el .env se nos debe ver de la siguiente manera

```
DB_URL=jdbc:postgresql://aws-0-us-east-1.pooler.supabase.com:5432/postgres
DB_USERNAME=postgres.udisjldmfcilexplbxfk
DB_PASSWORD=AIBnka2Rm3z7ajhN
```


19. Luego haremos el `application.properties`, donde colocaremos el `username`, la `password` y la `url` en privado, para eso realizamos el `.env` para que estos datos no sean publicos

```
spring.application.name=Futbol
spring.datasource.url=${DB_URL}
spring.datasource.username=${DB_USERNAME}
spring.datasource.password=${DB_PASSWORD}
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

20. Para terminar, modificamos el `Application`

```
package com.example.QuizPapeleria;

import ...

@SpringBootApplication
public class QuizPapeleriaApplication {

    public static void main(String[] args) {

        loadEnv();
        SpringApplication.run(QuizPapeleriaApplication.class, args);
    }

    private static void loadEnv(){

        Dotenv dotenv = Dotenv.load();
        System.setProperty("DB_URL", dotenv.get("DB_URL"));
        System.setProperty("DB_USERNAME", dotenv.get("DB_USERNAME"));
        System.setProperty("DB_PASSWORD", dotenv.get("DB_PASSWORD"));
    }
}
```

21. Esto lo hacemos para:

- Se trae los datos secretos del `.env`.

- Los inyecta en Java como propiedades del sistema.
- Así el proyecto sabe cómo conectarse a Supabase sin que se ponga la URL y las contraseñas directo en el código

22. Ahora corremos el proyecto y esto nos genera el modelo relacional

Modelo Supabase



Crear los Repositorios

23. Ahora volvemos a IntelliJ y seguimos con el proyecto. Empezaremos con el de jugador:

JugadorRepository

- Todo lo que tenga que ver con jugadores

- Método para buscar todos los jugadores de un equipo (findByEquipo).
- Método para buscar jugadores que han hecho más de X goles (findJugadoresConMasDeXGoles).
- Este repositorio sabe qué jugadores son de un equipo y cuáles jugadores han hecho muchos goles.

```
@Repository no usages new *
public interface JugadorRepository extends JpaRepository<Jugador, Integer> {

    // 1. Obtener todos los jugadores de un equipo específico
    @Query(value = "SELECT * FROM jugador WHERE id_equipo = :idEquipo", nativeQuery = true)
    List<Jugador> findByEquipo(@Param("idEquipo") Integer idEquipo);

    // 2. Obtener jugadores que han marcado más de X goles
    @Query(value = "SELECT j.* FROM jugador j " + no usages new *
            "JOIN estadisticas_jugador e ON j.id_jugador = e.id_jugador " +
            "GROUP BY j.id_jugador " +
            "HAVING SUM(e.goles) > :goles", nativeQuery = true)
    List<Jugador> findJugadoresConMasDeXGoles(@Param("goles") Integer goles);
}
```

24. Ahora vamos con Equipo:

EquipoRepository

- Todo lo que tenga que ver con equipos.
- Método para calcular el total de goles que ha hecho un equipo sumando local + visitante (getTotalGolesPorEquipo).
- Este repositorio sabe cuántos goles ha hecho un equipo en todos los partidos.

```

@Repository no usages new *
public interface EquipoRepository extends JpaRepository<Equipo, Integer> {

    // 3. Obtener el número total de goles marcados por un equipo en todos sus partidos
    @Query(value = "SELECT SUM(p.goles_local) FROM partido p WHERE p.equipo_local = :idEquipo " + no usages new *
        "UNION ALL " +
        "SELECT SUM(p.goles_visita) FROM partido p WHERE p.equipo_visita = :idEquipo", nativeQuery = true)
    List<Integer> getTotalGolesPorEquipo(@Param("idEquipo") Integer idEquipo);
}

```

25. El siguiente es Partido:

PartidoRepository

- Todo lo relacionado con partidos
- Método para obtener una lista de resultados de partidos con nombres de los equipos (obtenerResultadosDePartidos)
- Este repositorio sabe mostrar los partidos, quién jugó contra quién y cuántos goles hubo

```

@Repository no usages new *
public interface PartidoRepository extends JpaRepository<Partido, Integer> {

    // 4. Obtener los resultados de todos los partidos indicando los nombres de los equipos
    @Query(value = "SELECT p.id_partido, p.fecha, el.nombre AS equipo_local, ev.nombre AS equipo_visita, " + no usages new *
        "p.goles_local, p.goles_visita " +
        "FROM partido p " +
        "JOIN equipo el ON p.equipo_local = el.id_equipo " +
        "JOIN equipo ev ON p.equipo_visita = ev.id_equipo", nativeQuery = true)
    List<Object[]> obtenerResultadosDePartidos();
}

```

26. Los repositorios de entrenador y estadísticas_jugador como no tienen consultas nativas quedan de la siguiente manera:

EntrenadorRepository

- Todo sobre entrenadores.
- solo CRUD básico (crear, buscar, actualizar, eliminar entrenadores).
- Este repositorio sabe guardar o buscar entrenadores

```
@Repository no usages new *
public interface EntrenadorRepository extends JpaRepository<Entrenador, Integer> {
    ⚡
}
```

EstadisticasJugadorRepository

- Todo sobre estadísticas de jugadores.
- Solo el CRUD básico (guardar estadísticas, buscar estadísticas)
- Este repositorio guarda qué hizo un jugador en cada partido (goles, minutos, tarjetas)

```
@Repository no usages new *
public interface EstadisticasJugadorRepository extends JpaRepository<EstadisticasJugador, Integer> {
}
```

Crear los Servicios

27. Ahora Seguimos con los Services, el primero que creamos es Jugador:

JugadorService

```
public interface JugadorService { 2 usages 1 implementation new *
    List<Jugador> listarJugadores(); no usages 1 implementation new *
    Jugador buscarJugadorPorId(Integer id); no usages 1 implementation new *
    Jugador guardarJugador(Jugador jugador); no usages 1 implementation new *
    void eliminarJugador(Integer id); no usages 1 implementation new *
}
```

Define lo que vamos a poder hacer con los jugadores:

- Listarlos todos
- Buscar uno por ID
- Guardar uno nuevo
- Eliminar uno

JugadorServiceImpl

```
@Override no usages new *
public List<Jugador> listarJugadores() {
    return jugadorRepository.findAll();
}

@Override no usages new *
public Jugador buscarJugadorPorId(Integer id) {
    return jugadorRepository.findById(id).orElse( other: null);
}

@Override no usages new *
public Jugador guardarJugador(Jugador jugador) {
    return jugadorRepository.save(jugador);
}

@Override no usages new *
public void eliminarJugador(Integer id) {
    jugadorRepository.deleteById(id);
}
```

Aquí es donde se hace el trabajo real:

- Llama al JugadorRepository para sacar todos los jugadores con findAll().
- Usa save() para guardar uno.
- Usa findById() para buscar por ID.
- Usa deleteById() para eliminar.

28. Seguimos con Entrenador

EntrenadorService

```
public interface EntrenadorService { 2 usages 1 implementation new *
    List<Entrenador> listarEntrenadores(); no usages 1 implementation new *
    Entrenador buscarEntrenadorPorId(Integer id); no usages 1 implementation new *
    Entrenador guardarEntrenador(Entrenador entrenador); no usages 1 implementation new *
    void eliminarEntrenador(Integer id); no usages 1 implementation new *
```

Solo define los métodos:

- Listar entrenadores
- Buscar uno
- Guardar
- Eliminar

EntrenadorServiceImpl

```
@Override no usages new *
public List<Entrenador> listarEntrenadores() {
    return entrenadorRepository.findAll();
}

@Override no usages new *
public Entrenador buscarEntrenadorPorId(Integer id) {
    return entrenadorRepository.findById(id).orElse( other: null);
}

@Override no usages new *
public Entrenador guardarEntrenador(Entrenador entrenador) {
    return entrenadorRepository.save(entrenador);
}

@Override no usages new *
public void eliminarEntrenador(Integer id) {
    entrenadorRepository.deleteById(id);
}
```

Usa el EntrenadorRepository para hacer las operaciones CRUD igual que con jugadores.

29. Seguimos con Partido

PartidoService

```
public interface PartidoService { 2 usages 1 implementation new *
    List<Partido> listarPartidos(); no usages 1 implementation new *
    Partido buscarPartidoPorId(Integer id); no usages 1 implementation new *
    Partido guardarPartido(Partido partido); no usages 1 implementation new *
    void eliminarPartido(Integer id); no usages 1 implementation new *
}
```

Solo define los métodos:

- Listar partidos
- Buscar partido
- Guardar partido
- Eliminar partido

PartidoServiceImpl

```
@Override no usages new *
public List<Partido> listarPartidos() {
    return partidoRepository.findAll();
}

@Override no usages new *
public Partido buscarPartidoPorId(Integer id) {
    return partidoRepository.findById(id).orElse( other: null);
}

@Override no usages new *
public Partido guardarPartido(Partido partido) {
    return partidoRepository.save(partido);
}

@Override no usages new *
public void eliminarPartido(Integer id) {
    partidoRepository.deleteById(id);
}
```

- Se conecta al PartidoRepository y hace esas acciones

30. Ahora vamos con equipo

EquipoService


```
public interface EquipoService { 2 usages 1 implementation new *
    List<Equipo> listarEquipos(); no usages 1 implementation new *
    Equipo buscarEquipoPorId(Integer id); no usages 1 implementation new *
    Equipo guardarEquipo(Equipo equipo); no usages 1 implementation new *
    void eliminarEquipo(Integer id); no usages 1 implementation new *
}
```

- Define lo básico: listar, buscar, guardar y eliminar equipos.

EquipoServiceImpl

```
@Override no usages new *
public List<Equipo> listarEquipos() {
    return equipoRepository.findAll();
}

@Override no usages new *
public Equipo buscarEquipoPorId(Integer id) {
    return equipoRepository.findById(id).orElse(other: null);
}

@Override no usages new *
public Equipo guardarEquipo(Equipo equipo) {
    return equipoRepository.save(equipo);
}

@Override no usages new *
public void eliminarEquipo(Integer id) {
    equipoRepository.deleteById(id);
}
```

- Usa el EquipoRepository para hacer esas tareas.

31. Por ultimo EstadisticasJugador

EstadisticasJugadorService

```
public interface EstadisticasJugadorService { 2 usages 1 implementation new *
    List<EstadisticasJugador> listarEstadisticas(); no usages 1 implementation new *
    EstadisticasJugador buscarEstadisticaPorId(Integer id); no usages 1 implementation new *
    EstadisticasJugador guardarEstadistica(EstadisticasJugador estadistica); no usages 1
    void eliminarEstadistica(Integer id); no usages 1 implementation new *
}
```

Interfaz con métodos para:

- Listar estadísticas
- Buscar una
- Guardar
- Eliminar

EstadisticasJugadorServiceImpl

```
@Autowired 4 usages
private EstadisticasJugadorRepository estadisticasJugadorRepository;

@Override no usages new *
public List<EstadisticasJugador> listarEstadisticas() {
    return estadisticasJugadorRepository.findAll();
}

@Override no usages new *
public EstadisticasJugador buscarEstadisticaPorId(Integer id) {
    return estadisticasJugadorRepository.findById(id).orElse(other: null);
}

@Override no usages new *
public EstadisticasJugador guardarEstadistica(EstadisticasJugador estadistica) {
    return estadisticasJugadorRepository.save(estadistica);
}

@Override no usages new *
public void eliminarEstadistica(Integer id) {
    estadisticasJugadorRepository.deleteById(id);
}
```

- Usa EstadisticasJugadorRepository para trabajar directamente con el historial de cada jugador en los partidos.

32. Por último, hacemos los controladores que son los que reciben las peticiones del cliente (Postman) y usa los métodos del Service para hacer lo que le piden., empezamos con jugadores

JugadorController

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

    @Autowired 4 usages
    private JugadorService jugadorService;

    @GetMapping no usages new *
    public List<Jugador> listarJugadores() {
        return jugadorService.listarJugadores();
    }

    @GetMapping("/{id}") no usages new *
    public Jugador buscarJugador(@PathVariable Integer id) {
        return jugadorService.buscarJugadorPorId(id);
    }

    @PostMapping no usages new *
    public Jugador guardarJugador(@RequestBody Jugador jugador) {
        return jugadorService.guardarJugador(jugador);
    }

    @DeleteMapping("/{id}") no usages new *
    public void eliminarJugador(@PathVariable Integer id) {
        jugadorService.eliminarJugador(id);
    }
}
```

Qué hace

- Lista todos los jugadores
- Trae un jugador por ID
- Guarda un nuevo jugador (envías JSON en el body)
- Borra un jugador por ID

33. Vamos con entrenador

EntrenadorController

```
@RestController no usages new *
@RequestMapping("/entrenadores")
public class EntrenadorController {

    @Autowired 4 usages
    private EntrenadorService entrenadorService;

    @GetMapping no usages new *
    public List<Entrenador> listarEntrenadores() {
        return entrenadorService.listarEntrenadores();
    }

    @GetMapping("/{id}") no usages new *
    public Entrenador buscarEntrenador(@PathVariable Integer id) {
        return entrenadorService.buscarEntrenadorPorId(id);
    }

    @PostMapping no usages new *
    public Entrenador guardarEntrenador(@RequestBody Entrenador entrenador) {
        return entrenadorService.guardarEntrenador(entrenador);
    }

    @DeleteMapping("/{id}") no usages new *
    public void eliminarEntrenador(@PathVariable Integer id) {
        entrenadorService.eliminarEntrenador(id);
    }
}
```

Qué hace

- Lista todos los entrenadores
- Trae uno por ID
- Guarda uno nuevo
- Elimina por ID

34. Seguimos con Partidos

PartidoController

```
@RestController no usages new *
@RequestMapping("/partidos")
public class PartidoController {

    @Autowired 4 usages
    private PartidoService partidoService;

    @GetMapping no usages new *
    public List<Partido> listarPartidos() {
        return partidoService.listarPartidos();
    }

    @GetMapping("/{id}") no usages new *
    public Partido buscarPartido(@PathVariable Integer id) {
        return partidoService.buscarPartidoPorId(id);
    }

    @PostMapping no usages new *
    public Partido guardarPartido(@RequestBody Partido partido) {
        return partidoService.guardarPartido(partido);
    }

    @DeleteMapping("/{id}") no usages new *
    public void eliminarPartido(@PathVariable Integer id) {
        partidoService.eliminarPartido(id);
    }
}
```

Que Hace

- Lista todos los partidos
- Trae un partido por ID
- Guarda un partido nuevo
- Borra un partido por ID

35. Seguimos con Equipo

EquipoController

```
@RequestMapping("/equipos")
public class EquipoController {

    @Autowired 4 usages
    private EquipoService equipoService;

    @GetMapping no usages new *
    public List<Equipo> listarEquipos() {
        return equipoService.listarEquipos();
    }

    @GetMapping("/{id}") no usages new *
    public Equipo buscarEquipo(@PathVariable Integer id) {
        return equipoService.buscarEquipoPorId(id);
    }

    @PostMapping no usages new *
    public Equipo guardarEquipo(@RequestBody Equipo equipo) {
        return equipoService.guardarEquipo(equipo);
    }

    @DeleteMapping("/{id}") no usages new *
    public void eliminarEquipo(@PathVariable Integer id) {
        equipoService.eliminarEquipo(id);
    }
}
```

Que Hace

- Lista todos los equipos
- Trae un equipo por ID
- Guarda uno nuevo
- Elimina uno por ID

36. Por último seguimos con EstadisticasJugador

EstadisticasJugadorController

```
@RequestMapping("/estadisticas")
public class EstadisticasJugadorController {

    @Autowired 4 usages
    private EstadisticasJugadorService estadisticasJugadorService;

    @GetMapping no usages new *
    public List<EstadisticasJugador> listarEstadisticas() {
        return estadisticasJugadorService.listarEstadisticas();
    }

    @GetMapping("/{id}") no usages new *
    public EstadisticasJugador buscarEstadistica(@PathVariable Integer id) {
        return estadisticasJugadorService.buscarEstadisticaPorId(id);
    }











    @PostMapping no usages new *
    public EstadisticasJugador guardarEstadistica(@RequestBody EstadisticasJugador estadistica) {
        return estadisticasJugadorService.guardarEstadistica(estadistica);
    }

    @DeleteMapping("/{id}") no usages new *
    public void eliminarEstadistica(@PathVariable Integer id) {
        estadisticasJugadorService.eliminarEstadistica(id);
    }
}
```

Que Hace

- Lista todas las estadísticas
- Trae una por ID
- Guarda uno nuevo
- Elimina uno por ID

37. Primero ingresaremos 100 datos a cada modelo

Name	Description	Rows (Estimated)	Size (Estimated)	Realtime Enabled	
 entrenador	No description	100	32 kB	×	4 columns 
 equipo	No description	101	32 kB	×	4 columns 
 estadisticas_jugador	No description	100	24 kB	×	8 columns 
 jugador	No description	100	32 kB	×	7 columns 
 partido	No description	102	24 kB	×	7 columns 

38. Seguimos con el Crud de postman, Para esto haremos el crud para cada uno de los modelos, y luego pondremos las consultas nativas

Crud Jugador