

Parcial Segundo Corte

Parcial

Juan Esteban Capurro Buitrago

Guía resolución del parcial

Ing. William Alexander Matallana Porras

Juan Esteban Capurro Buitrago

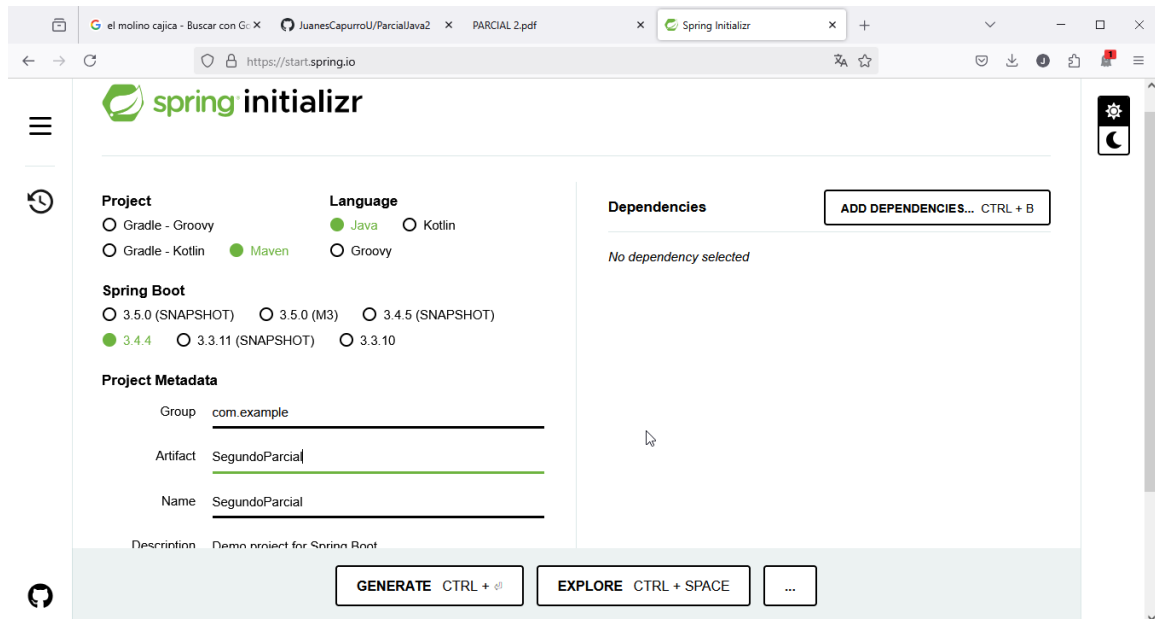
Corporacion Universitaria Minuto de Dios

Ingenieria de Sistemas

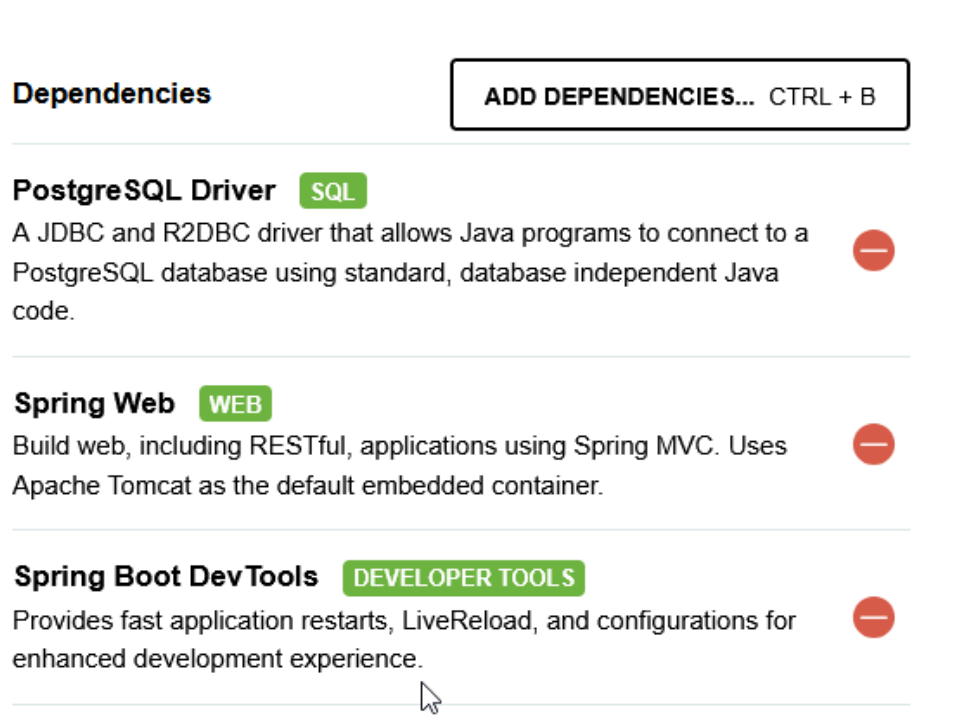
2025

Parcial Segundo Corte

Primero vamos a generar el archivo en la pagina de Spring Initializr.



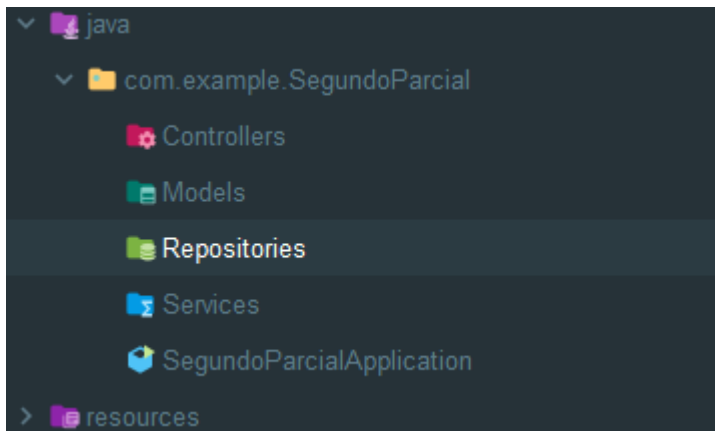
Ahora vamos a añadir las dependencias..



Y ahora vamos a descargar el archivo y abrirlo con el IntelliJ

Parcial Segundo Corte

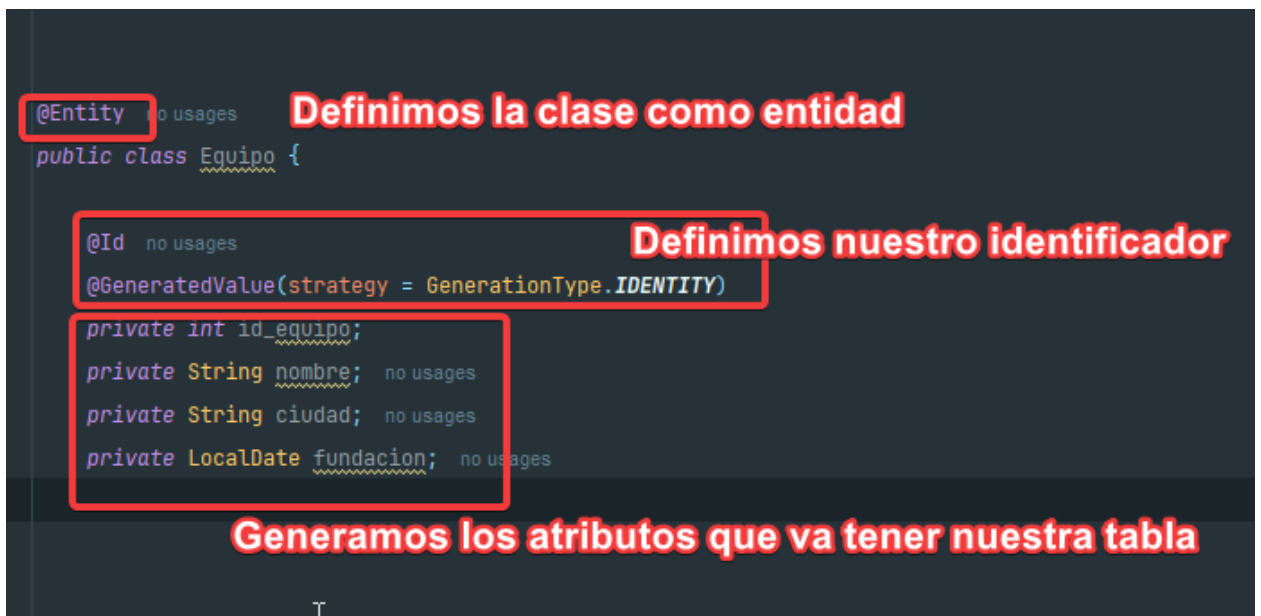
Una vez abierto nuestro proyecto, vamos a crear en la ruta del Main los siguientes paquetes: Controllers, Models, Services, Repositories.



Una vez creados las carpetas vamos a iniciar creando los modelos..

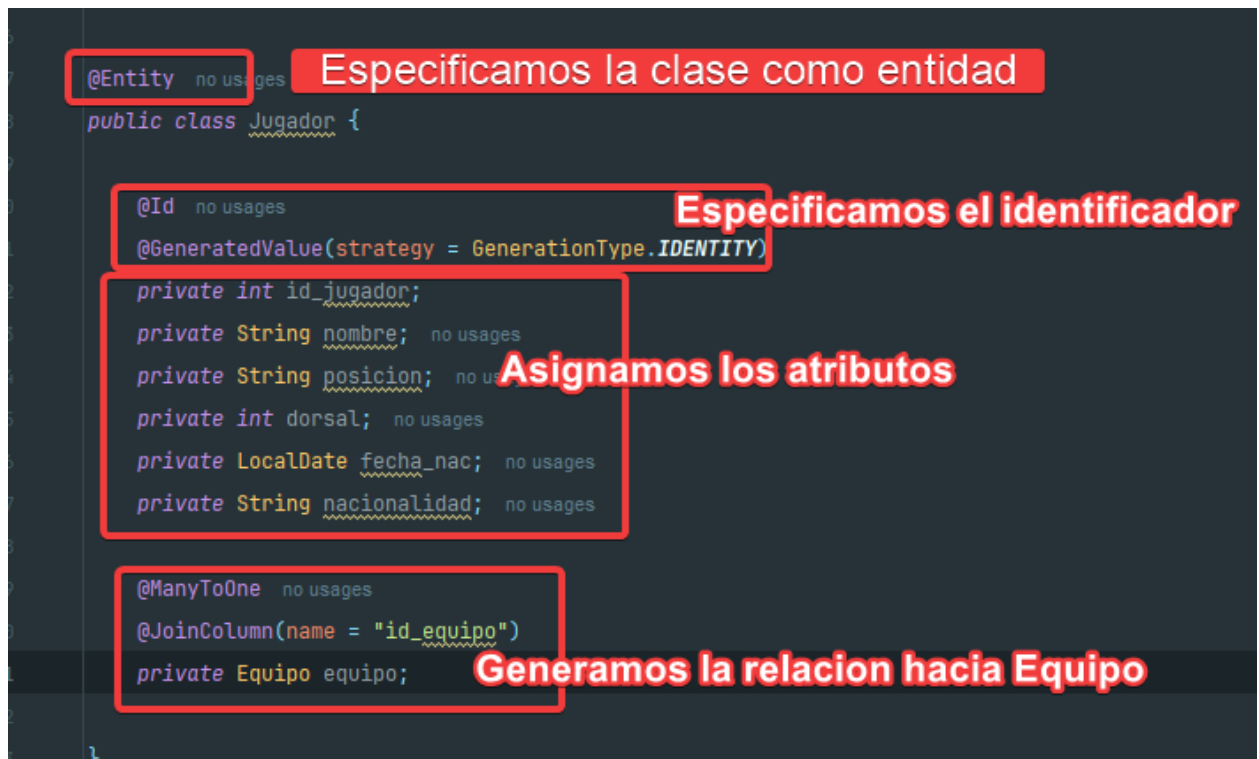
MODELS:

Equipo:

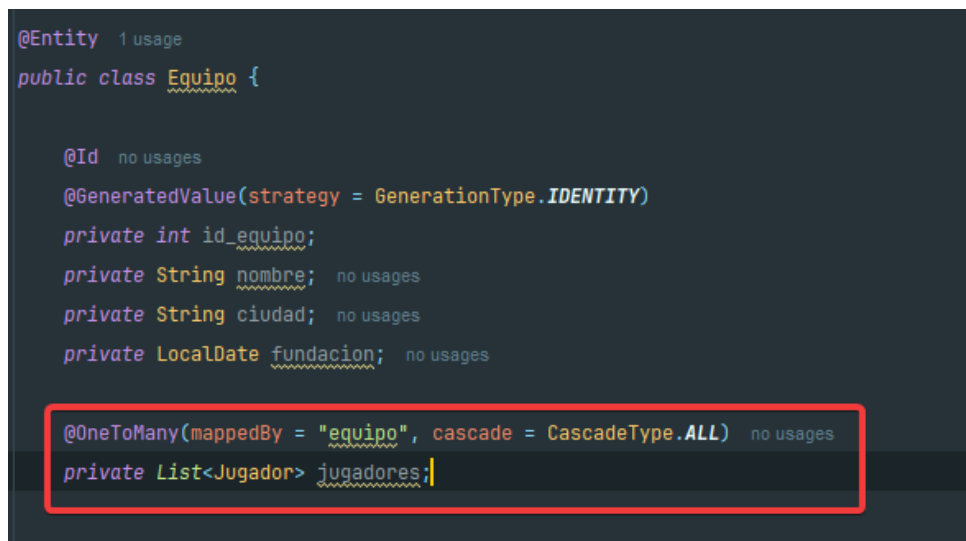


Después vamos a generar el Constructor, Getter and Setter y el toString.

Jugador:



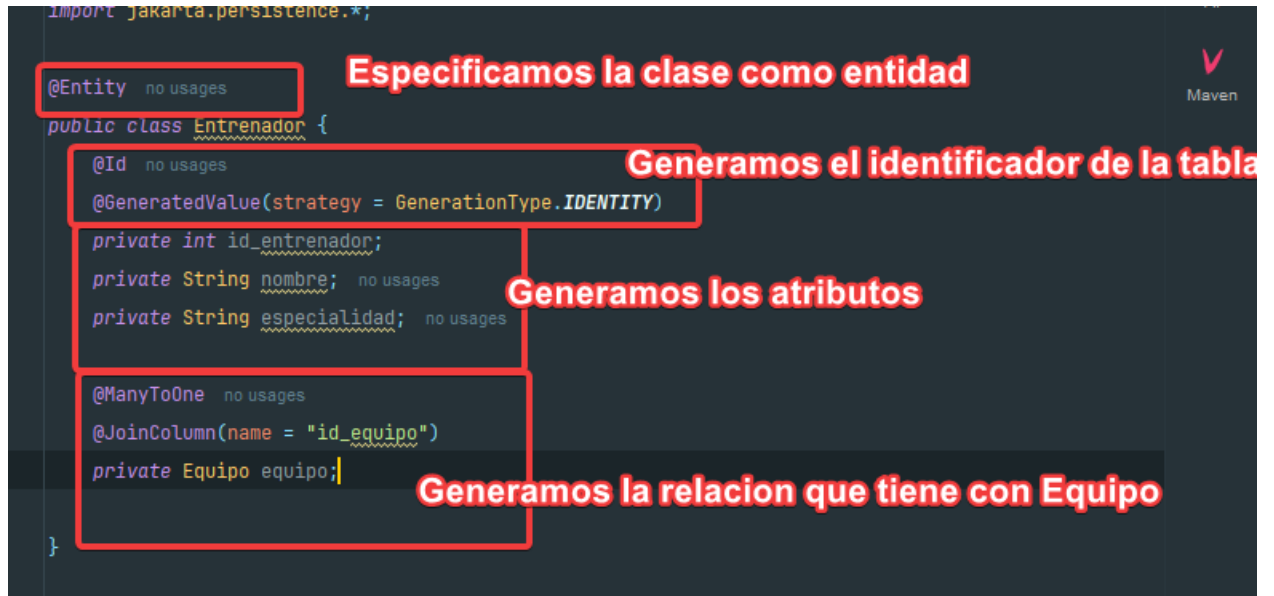
Aquí como generamos la relación hacia equipo, tambien debemos hacerla en la clase de Equipo con la relación hacia Jugador..



Como se evidencia, ahí generamos la relación de Jugador con equipo.

Parcial Segundo Corte

Entrenador:



Como generamos la relación con equipo, tambien debemos generarla en la clase del Equipo con Entrenador



Partido:

```
@Entity no us
public class Partido {

    @Id no usages
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id_partido;

    private LocalDate fecha; no usages
    private String estadio; no usages

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

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

    private int golesLocal; no usages
    private int golesVisita; no usages
}
```

asignacion de entidad a la clase

asignacion del identificador

atributos

relacion con equipo visitante y local

atributos

EstadisticaJugador:

```
@Entity no usages JuanesCapurroU *
public class EstadisticaJugador {

    @Id no usages
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id_estadistica;
    private int minutosJugados; no usages
    private int goles; no usages
    private int asistencias; no usages
    private int tarjetasAmarillas; no usages
    private int tarjetasRojas; no usages

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

    @ManyToOne no usages
    @JoinColumn(name = "id_partido")
    private Partido partido;
}
```

definimos la entidad

definimos atributos

definimos las relaciones

Parcial Segundo Corte

Añadimos la relación en jugador:

```
@Id no usages
@GeneratedValue(strategy = GenerationType.IDENTITY)
private int id_jugador;
private String nombre; no usages
private String posicion; no usages
private int dorsal; no usages
private LocalDate fecha_nac; no usages
private String nacionalidad; no usages

@ManyToOne no usages
@JoinColumn(name = "id_equipo")
private Equipo equipo;

@OneToMany(mappedBy = "jugador", cascade = CascadeType.ALL) no usages
private List<EstadisticaJugador> estadisticas;
}
```

Añadimos relación en partidos:

```
13 private int id_partido;
14 private LocalDate fecha; no usages
15 private String estadio; no usages
16
17 @ManyToOne no usages
18 @JoinColumn(name = "equipo_local")
19 private Equipo equipoLocal;
20
21 @ManyToOne no usages
22 @JoinColumn(name = "equipo_visita")
23 private Equipo equipoVisita;
24
25 private int golesLocal; no usages
26 private int golesVisita; no usages
27
28 @OneToMany(mappedBy = "partido", cascade = CascadeType.ALL) no usages
29 private List<EstadisticaJugador> estadisticas;
30 }
31
```

REPOSITORIES:

En esta sección vamos a iniciar a crear los repositories, para esto tenemos que crear las interfaces y asignamos un nombre..

Parcial Segundo Corte



```
1 package com.example.SegundoParcial.Repositories;
2
3 import com.example.SegundoParcial.Models.Entrenador;
4 import org.springframework.data.jpa.repository.JpaRepository;
5
6 public interface EntrenadorRepository extends JpaRepository<Entrenador, Integer> { no usages new *
7 }
```

Asignamos el JpaRepository para generar las consultas por defecto, definiendo nuestro Modelo que representara

Y así hacemos con todos...

```
package com.example.SegundoParcial.Repositories;

import com.example.SegundoParcial.Models.Equipo;
import org.springframework.data.jpa.repository.JpaRepository;

public interface EquipoRepository extends JpaRepository<Equipo, Integer> { no usages new *
}
```

```
package com.example.SegundoParcial.Repositories;

import com.example.SegundoParcial.Models.EstadisticaJugador;
import org.springframework.data.jpa.repository.JpaRepository;

public interface EstadisticaJugadorRepository extends JpaRepository<EstadisticaJugador, Integer> { no usages new *
}
```

```
package com.example.SegundoParcial.Repositories;

import com.example.SegundoParcial.Models.Partido;
import org.springframework.data.jpa.repository.JpaRepository;

public interface PartidoRepository extends JpaRepository<Partido, Integer> { no usages new *
}
```

```
package com.example.SegundoParcial.Repositories;

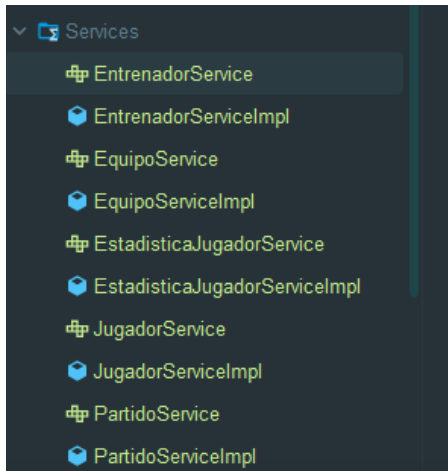
import com.example.SegundoParcial.Models.Jugador;
import org.springframework.data.jpa.repository.JpaRepository;

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


Parcial Segundo Corte

SERVICES:

Para la creación de los servicios debemos tambien crear las implementaciones.. De tal manera que nos queda asi:



Para los servicios son las mismas consultas nativas para todos:

```
public interface EntrenadorService { 2 usages 1 implementation new *
    List<Entrenador> obtenerTodos(); no usages 1 implementation new *
    Entrenador obtenerPorId(int id_entrenador); no usages 1 implementation new *
    Entrenador guardar(Entrenador entrenador); no usages 1 implementation new *
    public void eliminar(int id_entrenador); no usages 1 implementation new *
}
```

```
public interface EquipoService { 2 usages 1 implementation new *
    List<Equipo> obtenerTodos(); no usages 1 implementation new *
    Equipo obtenerPorId(int id_equipo); no usages 1 implementation new *
    Equipo guardar(Equipo equipo); no usages 1 implementation new *
    public void eliminar(int id_equipo); no usages 1 implementation new *
```

```
public interface EstadisticaJugadorService { 2 usages 1 implementation new *
    List<EstadisticaJugador> obtenerTodos(); no usages 1 implementation new *
    EstadisticaJugador obtenerPorId(int id_estadistica); no usages 1 implementation new *
    EstadisticaJugador guardar(EstadisticaJugador estadistica); no usages 1 implementation new *
    public void eliminar(int id_estadistica); no usages 1 implementation new *
```

Parcial Segundo Corte

```
public interface JugadorService { 2 usages 1 implementation new *  
    List<Jugador> obtenerTodos(); no usages 1 implementation new *  
    Jugador obtenerPorId(int id_jugador); no usages 1 implementation new *  
    ⚡ Jugador guardar(Jugador jugador); no usages 1 implementation new *  
    public void eliminar(int id_jugador); no usages 1 implementation new *
```

```
public interface PartidoService { 2 usages 1 implementation new *  
    List<Partido> obtenerTodos(); no usages 1 implementation new *  
    Partido obtenerPorId(int id_partido); no usages 1 implementation new *  
    ⚡ Partido guardar(Partido partido); no usages 1 implementation new *  
    public void eliminar(int id_partido); no usages 1 implementation new *
```

Se define los métodos para obtener, eliminar y guardar por la Id del modelo que se solicite..

Para las Implementaciones se generan los métodos para las consultas básicas, son las mismas para los 5 modelos

```
12 public class EntrenadorServiceImpl implements EntrenadorService {  
14     @Autowired 4 usages  
15     private EntrenadorRepository entrenadorRepository;  
16  
17     @Override no usages new *  
18     ⚡ public List<Entrenador> obtenerTodos() {  
19         return entrenadorRepository.findAll();  
20     }  
21  
22     @Override no usages new *  
23     ⚡ public Entrenador obtenerPorId(int id_entrenador) {  
24         return entrenadorRepository.findById(id_entrenador).orElse( other: null);  
25     }  
26  
27     @Override no usages new *  
28     ⚡ public Entrenador guardar(Entrenador entrenador) {  
29         return entrenadorRepository.save(entrenador);  
30     }  
31  
32     @Override no usages new *  
33     ⚡ public void eliminar(int id_entrenador) {  
34         entrenadorRepository.deleteById(id_entrenador);
```

Parcial Segundo Corte

```
public class EquipoServiceImpl implements EquipoService {  
  
    private EquipoRepository equipoRepository;  
  
    @Override no usages new *  
    public List<Equipo> obtenerTodos() {  
        return equipoRepository.findAll();  
    }  
  
    @Override no usages new *  
    public Equipo obtenerPorId(int id_equipo) {  
        return equipoRepository.findById(id_equipo).orElse( other: null);  
    }  
  
    @Override no usages new *  
    public Equipo guardar(Equipo equipo) {  
        return equipoRepository.save(equipo);  
    }  
  
    @Override no usages new *  
    public void eliminar(int id_equipo) {  
        equipoRepository.deleteById(id_equipo);  
    }  
}
```

```
2 public class JugadorServiceImpl implements JugadorService {  
3  
4     private JugadorRepository jugadorRepository;  
5  
6     @Override no usages new *  
7     public List<Jugador> obtenerTodos() {  
8         return jugadorRepository.findAll();  
9     }  
10  
11     @Override no usages new *  
12     public Jugador obtenerPorId(int id_jugador) {  
13         return jugadorRepository.findById(id_jugador).orElse( other: null);  
14     }  
15  
16     @Override no usages new *  
17     public Jugador guardar(Jugador jugador) {  
18         return jugadorRepository.save(jugador);  
19     }  
20  
21     @Override no usages new *  
22     public void eliminar(int id_jugador) {  
23         jugadorRepository.deleteById(id_jugador);  
24     }  
25 }
```

Parcial Segundo Corte

```
@Service no usages new *
public class EstadisticaJugadorServiceImpl implements EstadisticaJugadorService {

    @Autowired 4 usages
    private EstadisticaJugadorRepository estadisticaJugadorRepository;

    @Override no usages new *
    public List<EstadisticaJugador> obtenerTodos() {
        return estadisticaJugadorRepository.findAll();
    }

    @Override no usages new *
    public EstadisticaJugador obtenerPorId(int id_estadistica) {
        return estadisticaJugadorRepository.findById(id_estadistica).orElse( other: null);
    }

    @Override no usages new *
    public EstadisticaJugador guardar(EstadisticaJugador estadistica) {
        return estadisticaJugadorRepository.save(estadistica);
    }

    @Override no usages new *
```

```
public class PartidoServiceImpl implements PartidoService {

    private PartidoRepository partidoRepository;

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

    @Override no usages new *
    public Partido obtenerPorId(int id_partido) {
        return partidoRepository.findById(id_partido).orElse( other: null);
    }

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

    @Override no usages new *
    public void eliminar(int id_partido) {
        partidoRepository.deleteById(id_partido);
    }
}
```

Parcial Segundo Corte

Ahora hacemos los controllers:

Controllers:

Vamos a hacer los controladores para nuestro crud:

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

    12 public class EntrenadorController {
    13
    14     @Autowired 6 usages
    15     private EntrenadorService entrenadorService;
    16
    17     @GetMapping no usages new *
    18     public List<Entrenador> listarEntrenadores() {
    19         return entrenadorService.obtenerTodos();
    20     }
    21
    22     @GetMapping("/{id_entrenador}") no usages new *
    23     public Entrenador obtenerEntrenador(@PathVariable int id_entrenador) {
    24         return entrenadorService.obtenerPorId(id_entrenador);
    25     }
    26
    27     @PostMapping no usages new *
    28     public Entrenador crearEntrenador(@RequestBody Entrenador entrenador) {
    29         return entrenadorService.guardar(entrenador);
    30     }
    31
    32     @DeleteMapping("/{id_entrenador}") no usages new *
    33     public void eliminarEntrenador(@PathVariable int id_entrenador) {
    34         entrenadorService.eliminar(id_entrenador);
    35     }
    36 }
```

Para ello definimos la clase como RestController para que sepa que esta clase será nuestro controlador y el RequestMapping para la ruta de este controlador, como se ve anteriormente son los mismos controladores para todos lo que cambia son los nombres de los modelos y los id a obtener..

Parcial Segundo Corte

```
EntrenadorController.java  EquipoController.java  EstadisticaJugadorController.java
10  @RestController no usages new *
11  @RequestMapping("/api/equipos")
12  public class EquipoController {
13
14      @Autowired 6 usages
15      private EquipoService equipoService;
16
17      @GetMapping no usages new *
18      public List<Equipo> listarEquipos() {
19          return equipoService.obtenerTodos();
20      }
21
22      @GetMapping("/{id_equipo}") no usages new *
23      public Equipo obtenerEquipo(@PathVariable int id_equipo) {
24          return equipoService.obtenerPorId(id_equipo);
25      }
26
27      @PostMapping no usages new *
28      public Equipo crearEquipo(@RequestBody Equipo equipo) {
29          return equipoService.guardar(equipo);
30      }
31  }
```

```
EntrenadorController.java  EquipoController.java  EstadisticaJugadorController.java  JugadorController.java
5  import org.springframework.beans.factory.annotation.Autowired;
6  import org.springframework.web.bind.annotation.*;
7
8  import java.util.List;
9
10 @RestController no usages new *
11 @RequestMapping("/api/estadisticas")
12 public class EstadisticaJugadorController {
13
14  @Autowired 6 usages
15  private EstadisticaJugadorService estadisticaJugadorService;
16
17  @GetMapping no usages new *
18  public List<EstadisticaJugador> listarEstadisticas() {
19      return estadisticaJugadorService.obtenerTodos();
20  }
21
22  @GetMapping("/{id_estadistica}") no usages new *
23  public EstadisticaJugador obtenerEstadistica(@PathVariable int id_estadistica) {
24      return estadisticaJugadorService.obtenerPorId(id_estadistica);
25  }
26  }
```

Parcial Segundo Corte

```
0  @RestController no usages new *
1  @RequestMapping("/api/jugadores")
2  public class JugadorController {
3
4      @Autowired 6 usages
5      private JugadorService jugadorService;
6
7      @GetMapping no usages new *
8      public List<Jugador> listarJugadores() {
9          return jugadorService.obtenerTodos();
10     }
11
12     @GetMapping("/{id_jugador}") no usages new *
13     public Jugador obtenerJugador(@PathVariable int id_jugador) {
14         return jugadorService.obtenerPorId(id_jugador);
15     }
16
17     @PostMapping no usages new *
18     public Jugador crearJugador(@RequestBody Jugador jugador) {
19         return jugadorService.guardar(jugador);
20     }
21 }
```

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

    @Autowired 6 usages
    private PartidoService partidoService;

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

    @GetMapping("/{id_partido}") no usages new *
    public Partido obtenerPartido(@PathVariable int id_partido) {
        return partidoService.obtenerPorId(id_partido);
    }

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

Parcial Segundo Corte

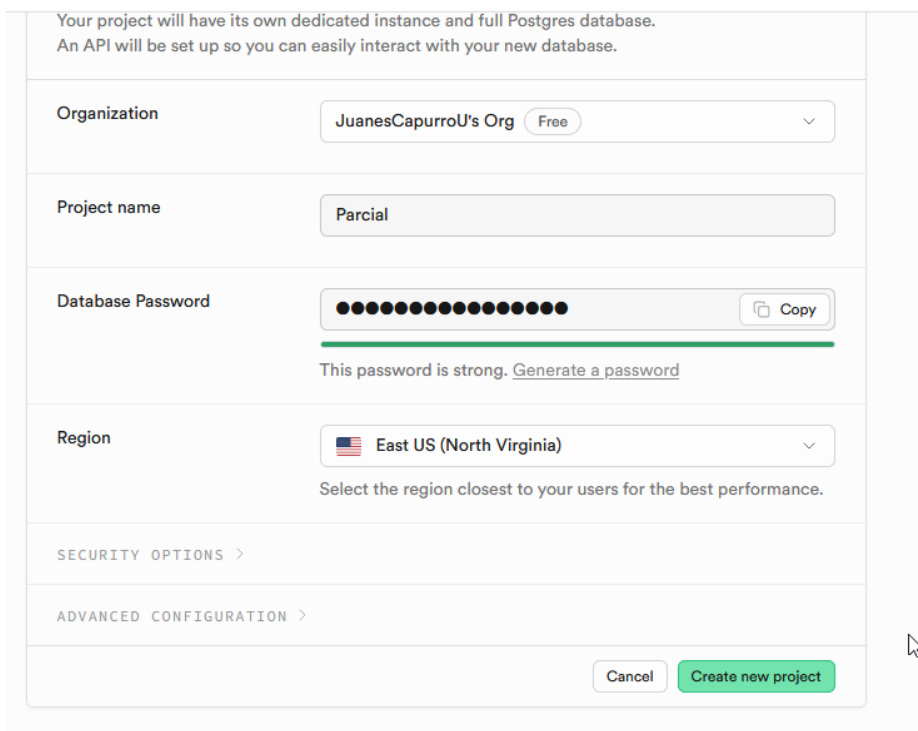
Configuración y conexión de la base de datos Supabase..

Para esto necesitamos configurar en el application properties

```
1  spring.application.name=SegundoParcial
2  spring.datasource.url=${DB_URL}
3  spring.datasource.username=${DB_USERNAME}
4  spring.datasource.password=${DB_PASSWORD}
5  spring.datasource.driver-class-name=org.postgresql.Driver
6
7  spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
8  spring.jpa.hibernate.ddl-auto=Create
9  spring.jpa.show-sql=true
10
```

Los datos que van en \${ } son ocultos para que no tengan facil acceso, ya que iran en un archivo .env vamos a hacer la creación de este archivo..

Ahora iremos al supabase y crearemos un nuevo proyecto..



Your project will have its own dedicated instance and full Postgres database.
An API will be set up so you can easily interact with your new database.

Organization: JuanesCapurroU's Org Free

Project name: Parcial

Database Password: [Masked Password] Copy

This password is strong. [Generate a password](#)

Region: East US (North Virginia)

Select the region closest to your users for the best performance.

SECURITY OPTIONS >

ADVANCED CONFIGURATION >

Cancel Create new project

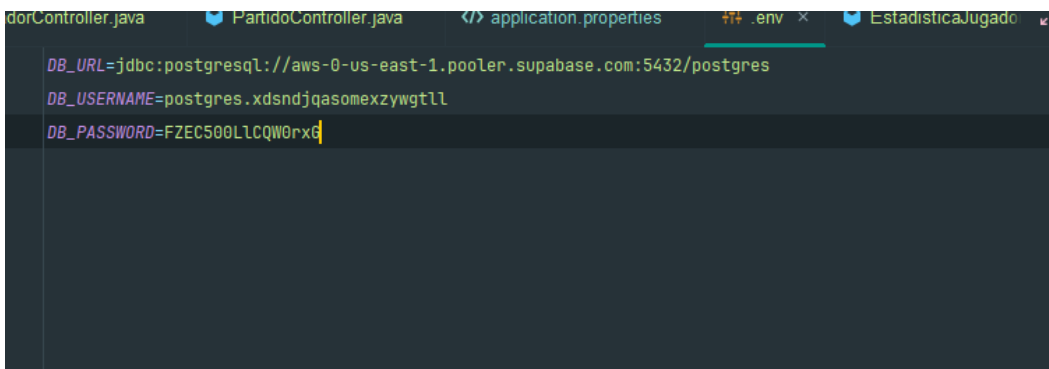
Recolectamos los datos de conexión:

Parcial Segundo Corte

FZEC500L1CQW0rxG

postgresql://postgres.xdsndjqasomexzywgt1l:[YOUR-PASSWORD]@aws-0-us-east-1.pooler.supabase.com:5432/postgres|

Y los pegamos en nuestro .env de tal manera:



```
DB_URL=jdbc:postgresql://aws-0-us-east-1.pooler.supabase.com:5432/postgres
DB_USERNAME=postgres.xdsndjqasomexzywgt1l
DB_PASSWORD=FZEC500L1CQW0rxG
```

Ahora para que esto funcione debemos cerar en nuestro main e importar el uso de este archivo env para que cargue al iniciar nuestro proyecto y se puedan leer sus datos..



```
public static void main(String[] args) { // JuanesCapurroU *
    loadEnv();
    SpringApplication.run(SegundoParcialApplication.class, args);
}

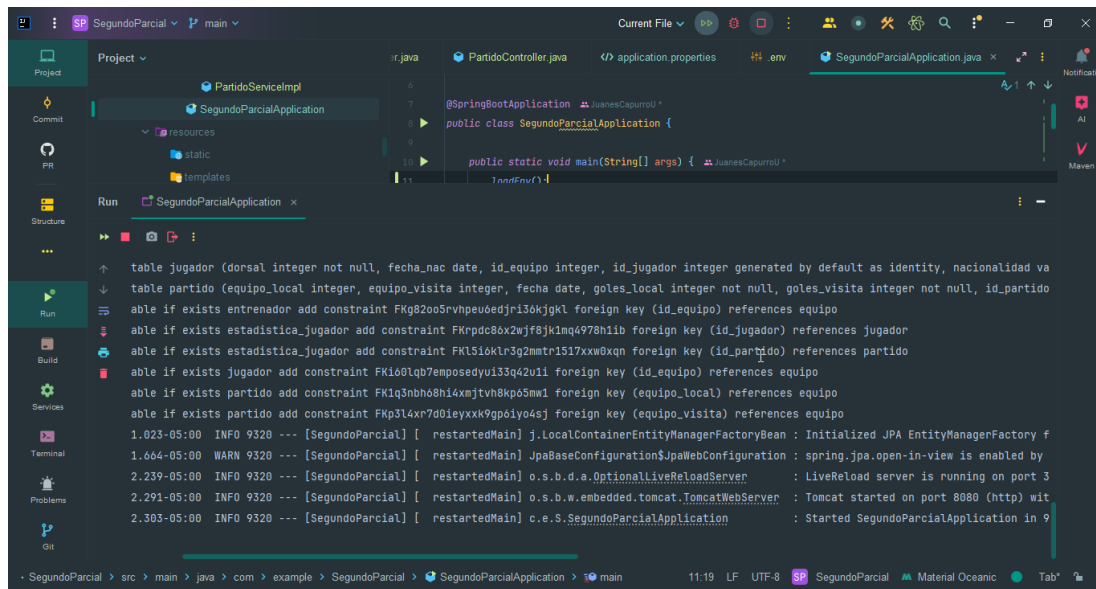
private static void loadEnv() { // usage: new *
    Dotenv dotenv = Dotenv.load();
    System.setProperty("DB_URL", dotenv.get("DB_URL"));
    System.setProperty("DB_USERNAME", dotenv.get("DB_USERNAME"));
    System.setProperty("DB_PASSWORD", dotenv.get("DB_PASSWORD"));

    System.out.println("DB_URL: " + dotenv.get("DB_URL"));
    System.out.println("DB_USERNAME: " + dotenv.get("DB_USERNAME"));
    System.out.println("DB_PASSWORD: " + dotenv.get("DB_PASSWORD"));
}
```

Parcial Segundo Corte

Ahí lo que hacemos es asignarle las propiedades que llevara en env en nuestro archivo...

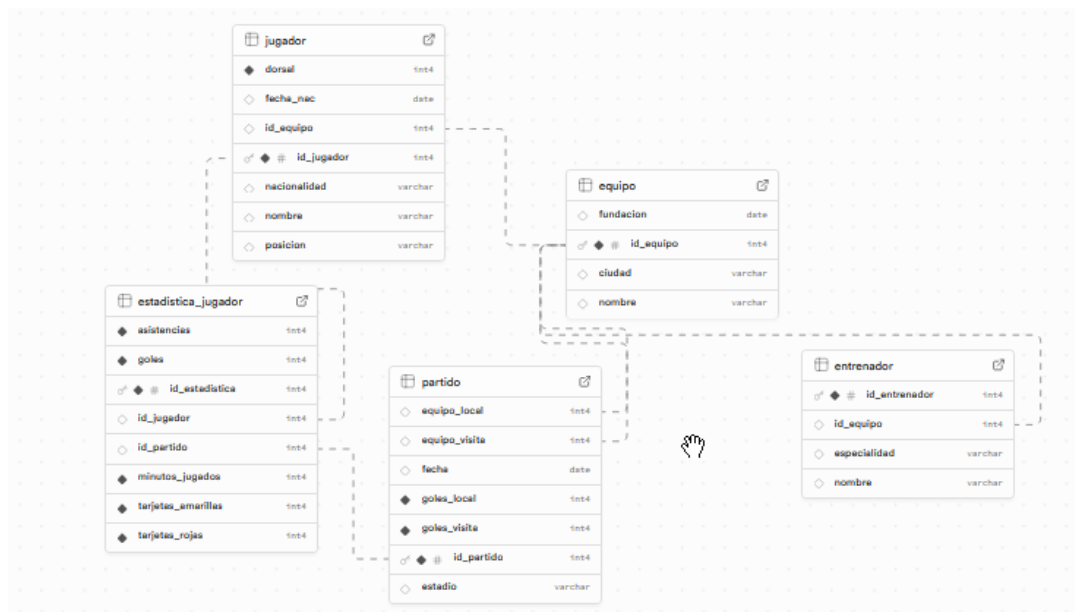
Ahora vamos a iniciar nuestro proyecto y verificar que todo este correctamente..



The screenshot shows an IDE with a project named 'SegundoParcial'. The project structure includes a 'resources' folder with 'static' and 'templates' subfolders. The 'Run' tab is active, showing the execution of 'SegundoParcialApplication'. The terminal window displays the following logs:

```
1.023-05:00 INFO 9320 --- [SegundoParcial] [ restartedMain] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory f
1.664-05:00 WARN 9320 --- [SegundoParcial] [ restartedMain] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by
2.239-05:00 INFO 9320 --- [SegundoParcial] [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 3
2.291-05:00 INFO 9320 --- [SegundoParcial] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) wit
2.303-05:00 INFO 9320 --- [SegundoParcial] [ restartedMain] c.e.S.SegundoParcialApplication : Started SegundoParcialApplication in 9
```

Como vemos se ejecuto bien, ahora iremos al supabase a verificar las tablas..



REFERENCIAS

JPA entity relationships. (2015, enero 10). Translate.Goog; Tutorialspoint. https://www-tutorialspoint-com.translate.goog/es/jpa/jpa_entity_relationships.htm?_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=tc