# Entrega 5

**Fecha de entrega: 23 de septiembre**

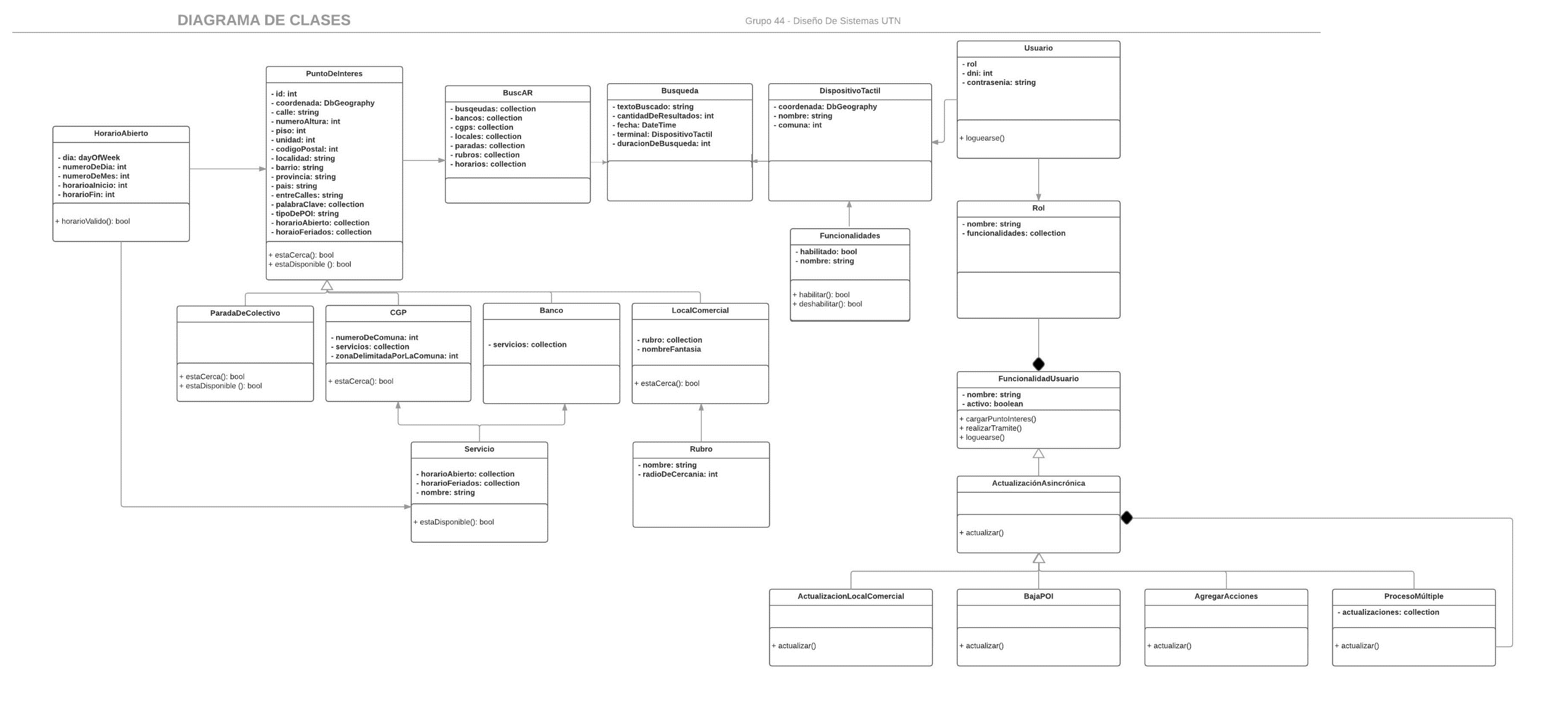
Esta entrega incluye:

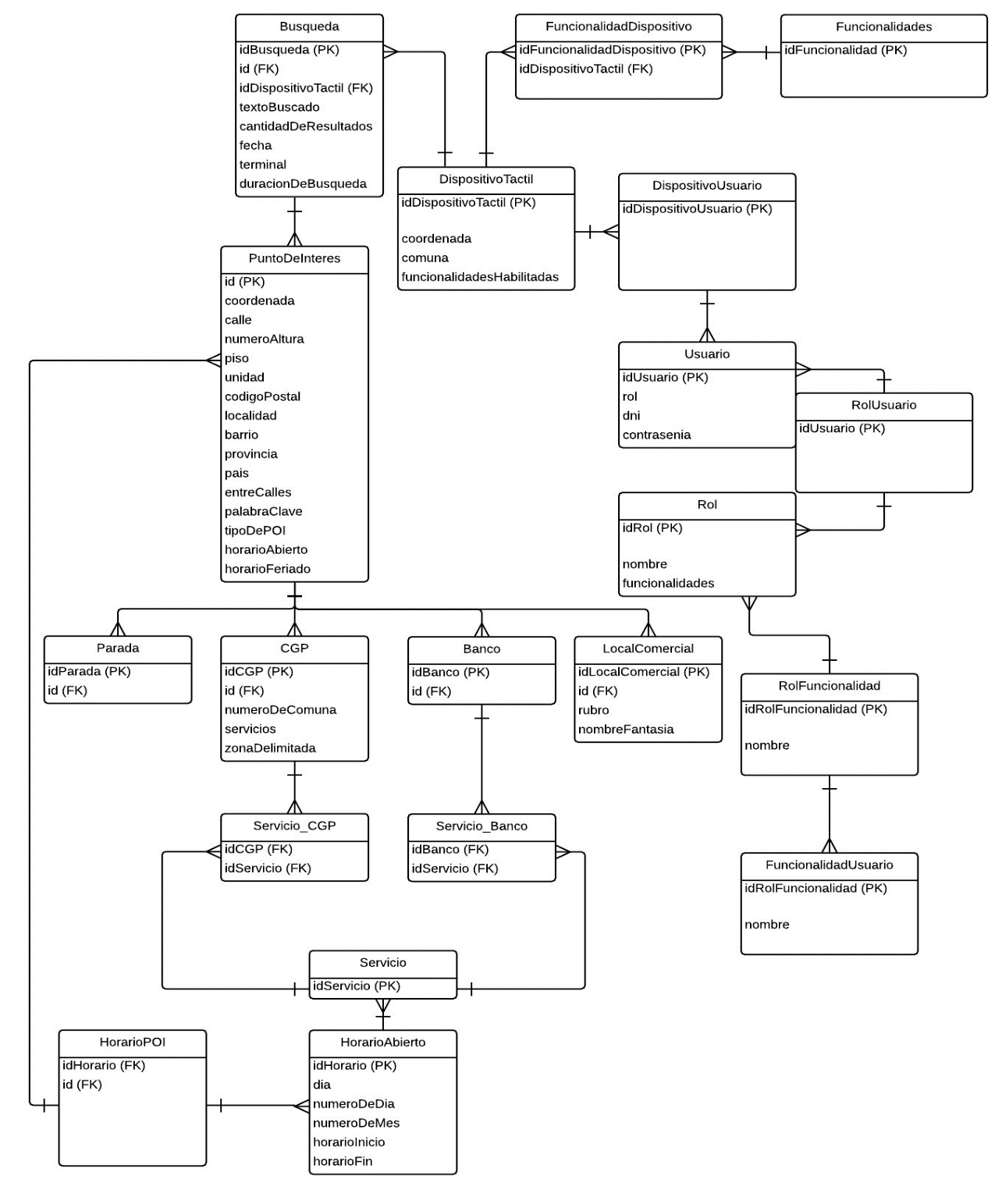
* Modelado de datos
* Construcción de componente para servicios REST
* Diseño e implementación de interfaz de usuario.

### 5.1. Modelado de datos

Realizar el modelado de datos mediante un Diagrama de Entidad-Relación (DER) justificando las decisiones sobre el mismo. Para su construcción utilizar la técnica de mapeo objetos-relacional (ORM) justificando cada decisión tomada.

No es obligatorio realizar la persistencia sobre un motor de bases de datos. Esto será requerido en entregas siguientes.



Diagrama Entidad Relación (DER)

Se utiliza la técnica de mapeo objetos-relacional (ORM)

Detalle de cómo se realiza el mapeo

|  |  |
| --- | --- |
| @MappedSuperclass  abstract class PersistenEntity   @Id   @GeneratedValue   long id  @Entity  @Table('Regiones')  class Region extends PersistentEntity   @ManyToMany   Collection<Lugar> lugares   @ManyToOne   Casa casaPrincipal  ... las demás entidades    heredan de PersistentEntity también ...  @Entity  @Table('Casas')  class Casa   @ManyToOne   Casa vasallaDe   @ManyToOne   Lugar origen   @OneToMany   @JoinColumn('casa\_id')   Collection<FuerzaMilitar> vasallaDe | @Entity  @Table('Lugares')  @Inheritance(JOINED)  abstract class Lugar  @Entity  @Table('Castillos')  class Castillo extends Lugar  ...mapeo análogo en Ciudad...  @Entity  @Table('Fuerzas\_Militares')  @Inheritance(SINGLE\_TABLE)  @DiscriminatorColumn('tipo')  abstract class FuerzaMilitar  @Entity  @DiscriminatorValue('A')  class Aerea extends FuerzaMilitar    Int cantidadDragones  ...mapeo análogo en los otros tipos de fuerza.... |

Justificación de las decisiones tomadas sobre los siguientes aspectos:

* Cambios realizados al modelo de objetos

|  |
| --- |
| Se agregaron ids en cada una de las clases como claves primarias (PK).  Por otro lado, se agregan ids para las claves foráneas (FK) en donde se presenta la relación “Muchos a Muchos” lo que requiere también en cada uno de esos casos una tabla asociada como es en el caso de las clases “CGP” con la clase “Servicio” la tabla asociada “Servicio\_CGP”, la clase “Banco” con la clase “Servicio” la tabla asociada “Servicio\_Banco”, con la clase “HorarioAbierto” y “PuntoDeInteres” la clase asociada “HorarioPOI”, también para la clase “DispositivoTactil” con la clase “Funcionalidades” la tabla asociada “FuncionalidadDispositivo”, la clase “Usuario” y la clase “Rol” con la tabla asociada “RolUsuario” y por último la clase “Rol” y la clase “FuncionalidadUsuario” con la tabla asociada “RolFuncionalidad”.  , se debió convertir la fuerza militar en clase abstracta |

* Si se utilizó alguna estrategia de mapeo de herencia, o alguna @MappedSuperclass

|  |
| --- |
| Se podría utilizar una MappedSuperclass para tener la lógica de mapeo y comportamiento común a todas las entidades (PersistentEntity). Se podría considerar para extraer la lógica de mapeo común entre por ejemplo la clase padre “PuntoDeInteres” con sus clases hijas “Parada”, “CGP”, “Banco” y “LocalComercial”. |

* Si se embebió (@Embeddeded) o enumeró alguna entidad (@Enumerated)

|  |
| --- |
| @Enumerated es útil cuando tenemos strategies stateless. Pero las fuerzas militares son strategies stateful, por tanto, no podemos convertirlos en una enumeración de códigos, y necesitaremos una tabla para almacenar el estado extra.  Por otro lado, podríamos **suponer** que las fuerzas militar son value objects (es decir, que no importa su identidad, motivo por el cual el dominio no nos lleva a compartirlas entre casas), y pensar por tanto que podrían ser embebibles, pero:   * es una relación one to many, no one to one * hay polimorfismo, y JPA no se lleva bien con esto |

* Si se tuvo alguna consideración particular para mantener el orden o duplicados en colecciones

|  |
| --- |
| Ninguna de las colecciones de este problema necesita soportar orden (ninguna es una List), por lo que no utilizamos @OrderColumn's  No se pueden tener datos duplicados en la base de datos, como por ejemplo no se puede tener dos POIs iguales, en cambio el orden en que vengan los datos es indistinto. |

* Si se empleó algún objeto de base de datos como constraints, triggers, vistas o procedimientos

|  |
| --- |
| En la solución propuesta no usamos triggers, vistas ni procedimientos. Sí se podrían aplicar constraints not null sobre por ejemplo las siguientes columnas:  LUGARES.nombre  CIUDADES.nombre  CASTILLOS.cantidad\_torres,  etc  Y constraints check sobre FUERZAS\_MILITARES.tipo:  check FUERZAS\_MILITARES.tipo in ('A', 'T', 'N')  No se podrían haber aplicado constraints not null sobre los atributos no comunes de las fuerzas militares. |