**Anotaciones**

**Paso 1 de 7: Programamos todas las clases de negocio**

1- modelo > contiene las clases y los dtos

Entidades: private Set<Cursada> cursadas; *Puede tener muchas\* cursadas, la colección set es porque después utilizamos Hibernate, el cual utiliza los sets para no repetir los elementos que están dentro de la colección.*

Dto: Nos permiten transportar la información de una capa a la otra, un dao por cada entidad (clase). Mostramos los datos que necesitamos, normalmente por esto es más “liviano”

**Paso 2 de 7: Programamos todas las clases de acceso a datos**

Como usamos ORM de hibernete haremos:

* 1) Realizar todas las anotaciones en las clases del modelo, para marcar las relaciones entre las clases y los mapeos con los campos y tablas de la BD.
* 2) Realizar por cada entidad una interfaz que herede de JPARepository

Lo que se hacia con DAO se hace con el ORM (es más simple)

Utilizamos como dependencia H2 DataBase

Mapeos => import javax.persistence.\*;

@Entity   
@Table (name = “comosellamalatabla”)

@Id  
@GeneratedValue // Si no le pasamos el id que lo autogenere

// Establecemos la relación

@OneToMany(mappedBy = “comosellamalapropiedad que relaciona las tablas”) ej: estudiante  
@JsonIgnore // siempre que se trate de relación uno a muchos, para que no entre en un loop  
private Set<Cursada> cursadas;

@JoinColumn(name=”forengkey, nullable= false)

2- Repository (EX DAO).

Por cada una de las entidades, un repositoy (Interface) ej : IEstudianteRepository **extends** JpaRepository <Estudiante, (tipo de id) Long>

@Repository

**Paso 3 de 7: Programamos todas las clases de servicio**

Es una buena practica que estas clases implementen una interface (es opcional). Creamos interface y service ej: IEstudianteService (Interface) EstudianteService (clase)

3- Service (implementa IEstudianteService)

Interface IEstudianteService, en esta clase realizamos el CRUD (Crear, Leer, Modificar, Eliminar), en este caso + un método para devolver un listado con todos los estudiantes:

Set<EstudianteDTO> getTodos();

El service necesita acceder a la base de datos => private IEstudianteRepository est; con @Autowired

@Autowired  
ObjetMapper (se requiere dependencia modelmapper (Jackson))

**Paso 4 de 7: Programamos todos los casos de testeos unitarios**

Import org.springframework.boot.test.context.SpringBootTest

@ SpringBootTest

En el ejemplo desarrolla un test unitario a la clase EstudianteService

4- Test

Completa configuraciónes en Aplication.properties

**Paso 5 de 7: Programamos todas las APIs**

Por cada entidad: Un controller

5- Controllers (EstudianteController)

@RestController  
@RequestMapping(“/estudiantes”)

Los controller se comunican con la capa de servicio, por lo cual se instancia una referencia del mismo ej.:

@Autowired  
IEstudianteService estudianteService;

Definimos los métodos.

Post - Get - Put - Delete

**Paso 6 de 7: Programamos el manejo de errores de manera global**

Capturamos todos los errores y los guardamos en un log

Utilizamos GlobalExceptionHandler

1) Creamos el properties de log4j con las configuraciones necesarias

2) Creamos el archivo en donde se guardarán los errores

3) En la clase donde esta el main, pasamos el properties

4)Creamos una clase en donde esta el main ; Como “GlobalExceptionHandler”

@ControllerAdvice

@ExceptionHandler(Exception.class)

**Paso 7 de 7: Programamos el front end, la vista**

Utilizamos JS para consumir apis