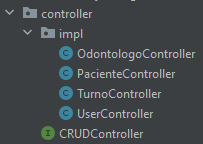
Proyecto Integrador, ficha técnica.

El proyecto integrador “Clinica odontológica” correspondiente a BackEnd contienene las siguientes carpetas:

**Configuracion**



**Controller**:

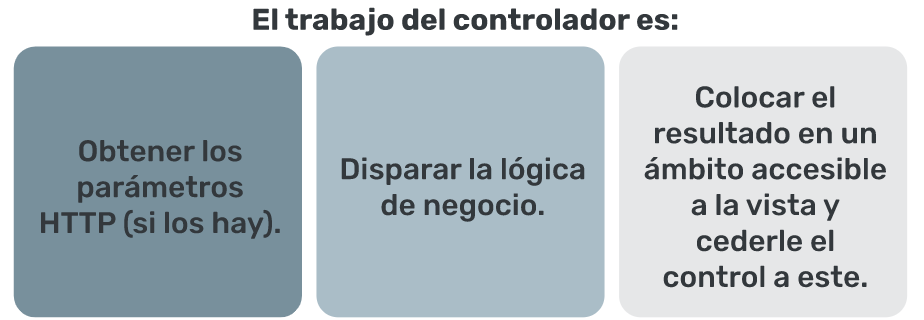


Conforma la capa intermedia entre las **vistas y los modelos (Patrón ModeloVistaControlador).** Sus responsabilidades son **procesar** los datos que recibe de los modelos y elegir la vista correspondiente en función de aquellos datos. **Tiene relación directa con las vistas y con los modelos y es un componente fundamental dentro del flujo del patrón.**

**MVC:**

1. La **vista** se conecta con el **controlador** para solicitarle esos datos
2. El **controlador** recibe la petición, valida que los datos sean correctos y le solicita al **modelo** el detalle
3. El **modelo** busca la información solicitada y se la envía al **controlador**
4. El **controlador** recibe la información y le envía los datos a la **vista**

El patrón MVC es un patrón de diseño que nos permite ***organizar el código y los archivos en función de su responsabilidad,*** en otras palabras, en función de la tarea que debe realizar.



Definimos : @Interface CRUDCONTROLLER; la cual tiene el “esqueleto” que utilizaran las demás clasesController, el cual es ResponseEntyti<?> ; Ej :

ResponseEntity<?> crear(@RequestBody T t) throws ServiceException, BadRequestException, FindByIdException, UnauthorizedAccessException;

Este método contiene excepciones, las cuales desarrollaremos en la carpeta **EXCEPTIONS.**

**Es importante aclarar que todas los “subControllers” usan:**

implements CRUDController<OdontologoDto> . Es decir interface y tipo de objeto dinamico.

**Cuestiones a tener en cuenta en clase controller.**

@RestController = Indica que esta clase cumple la función de controlador  
@RequestMapping("/odontologos") Asigna /odontólogos post localhost8080

Anotaciones:

@Autowired : permite inyectar unas dependencias con otras dentro de Spring

@Autowired  
UserServiceImpl appUserServiceImpl;  
@Autowired  
private OdontologoServiceImpl odontologoServiceImpl;

@GetMapping

@PostMapping

@PutMapping

@DeleteMapping

Cada anotación tiene su desarrollo, lo mejor aplicarlos mediante reutilización de código.

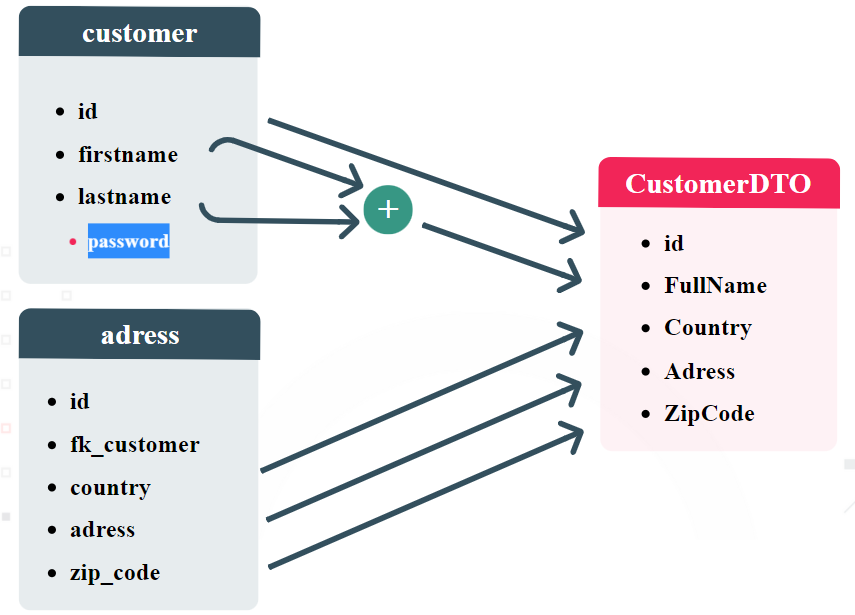
**Data Transfer Object (DTO)**

**El patrón DTO tiene como finalidad crear un objeto plano (POJO) con una serie de  
atributos que puedan ser enviados o recuperados del servidor en una sola invoca-  
ción, de tal forma que un DTO puede contener información de múltiples fuentes o  
tablas y agruparla en una única clase simple.**

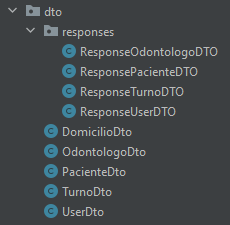
**DTO tiene que cumplir 2 reglas:**

A- Solo lectura: dado que el objetivo de un DTO es utiilizarlo como un objeto de transferencia entre cliente y servidor, es importante evitar tener operaciones de negocio o métodos que realicen cálculos sobre los datos. Por eso necesitaremos get y set de los atributos

B- Serializable: Si los objetos tienen que viajar por la red, deberán ser serializables, tanto en la clase en sí como todos los atributos que contenga el DTO.

****

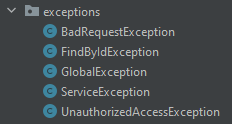
**Proyecto:**



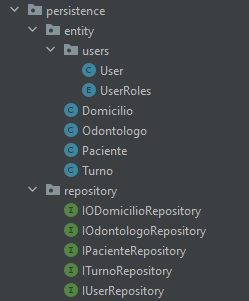
Un DTO copia a las clases o entidades seleccionadas en su desarrollo. Las entidades son clases que **fueron diseñadas para mapear contra la base de datos**, no para ser una vista para una pantalla o servicio determinado, lo que provoca que muchos de los campos no puedan ser serializables o no contengan todos los campos necesarios.

Para pasar los datos de una entidad a un DTO podemos utilizar los Getters y Setters, o utilizar Jackson, una librería. En el caso “proyecto integrador” esta con Get&Set

**Exceptions:**



**Persistence:**

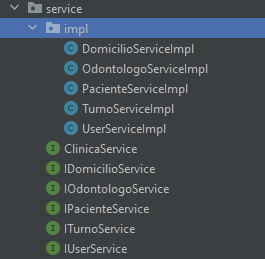


En esta carpeta encontraremos 2 ramas principales

A: Dentro de la carpeta “entity” Las clases/entidades base del proyecto y una carpeta **Users**, con el desarrollo y permisos para los usuarios con los roles.

B: La carpeta resository la cual contiene interfaces que extienden de “JpaRepository”. La mismas tienen una sola funcionalidad desarollada.

**Service:**



Service contiene 2 ramas:

A: La carpeta IMPL la cual contiene el ServiceImpl. De cada clase, en el cual se desarrolla CRUD en base a los DTO. EJ:

@Override  
public DomicilioDto actualizar(DomicilioDto domicilio) throws FindByIdException {

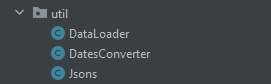
B: Por otro lado están las interfaces con el esqueleto del crud:

public interface IDomicilioService extends ClinicaService<DomicilioDto,DomicilioDto> {  
  
 DomicilioDto registrar(DomicilioDto domicilioDto);  
  
 DomicilioDto buscar(Integer id) throws FindByIdException;  
  
 List<DomicilioDto> buscarTodos();  
  
 String eliminar(Integer id) throws FindByIdException;  
  
 DomicilioDto actualizar(DomicilioDto domicilioDto) throws FindByIdException;

Las cuales extienden de ClinicaService

@Service  
public interface ClinicaService<T,E> {  
  
 T registrar(E e) throws BadRequestException, FindByIdException;  
  
 T buscar(Integer id) throws FindByIdException;  
  
 List<T> buscarTodos() throws FindByIdException;  
  
 String eliminar(Integer id) throws FindByIdException;  
  
 T actualizar(E e) throws FindByIdException;  
}

Por último, tenemos la carpeta Util con archivos “útiles”.



Esta es la rama “principal” del proyecto. Por otro lado, se desarrolla todo lo que son los test.

