**REPOSITORIOS**

**Lombok**

***@Data*** - Crear getter y setter automático

***@NoArgsConstructor*** y ***@AllArgsConstructor*** - Crear constructores con o sin argumentos

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Employee implements Serializable{

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long id;

private String name;

private String email;

}

***@RequiredArgsConstructor*** - Se deben crear los atributos con el final delante

@Data proporciona esta etiqueta por lo que no es necesario usarla dos veces

**MapStruct** -<https://mapstruct.org/>

Se creará el código Java necesario para convertir los DTO en entidades u otro DTOS, cuando compilemos.

public class User {

private String username;

private String password;

private String name;

private String lastname;

//constructor, getters, setters etc.

}

public class UserOutputDTO {

private String username;

private String name;

private String lastname;

//constructor, getters, setters etc.

}

Aquí @Getter y @Setter sirven para poner los get y los set propios de una clase para poder acceder a ellos desde fuera. Estas también son dos etiquetas que se pueden colocar justo encima de, por ejemplo, public class Usuario.

**Clásico** - Recurrimos a lo que denominamos GetSetGetSet (Setear los getters del DTO)

public UserOutputDTO setPersonDTO(User user){

UserOutputDTO userOutput = new UserOutputDTO();

userOutput.setUsername(user.getUsername());

userOutput.name(user.getName());

userOutput.lastname(user.getLastName());

return userOutput;

}

**Con MapStruct** -

**@Mapper**

public interface CarMapper {

UserMapper INSTANCE = Mappers.getMapper( UserMapper.class );

**@Mapping**(source = "username", target = "username")

UserOutputDTO userToUserOutputDTO(User user);

}

1. Utilizamos la anotación @Mapper que indica que esta interfaz es de MapStruct

2. El método se encarga de obtener el objeto de origen y devilver el objeto que se quiere pasar

* Si los nombres que vamos a utilizar para las anotaciones son diferentes podemos configurar la anotación
* Si se puede y es necesario, incluso se encarga de realziar conversiones de tipos del origen al objeto
* Hay que tener en cuenta que puede tener diferentes métodos para todas las implementaciones que genere MapStruct

3. La instancia de la interfaz puede recuperarse a través de la clase **Mapper.**

* Por convención, declara un miembro "INSTANCE" que se utiliza para proveer el acceso a los usuarios de la implementación de Mapper.

Ya tenemos la conversión de clases, simplificando mucho el trabajo haciéndolo más óptimo.

**ModelMapper**  - <http://modelmapper.org/>

No difiere mucho de MapStruct. Nos ofrece simplicidad.

* Mayor diferencia: No genera el código Java en tiempo de codificación, usando reflexión, busca los setters y getters necesarios para hacer la conversión de datos.
* Es más lento, cuando sea importante la velocidad, sería mejor usar otra libería.

public class User {

private String username;

private String password;

private String name;

private String lastname;

//constructor, getters, setters etc.

}

public class UserOutputDTO {

private String username;

private String name;

private String lastname;

//constructor, getters, setters etc.

}

Para trabajar sobre esto simplemente haremos lo siguiente:

ModelMapper modelMapper = new ModelMapper();

UserOutputDTO userOutput = modelMapper.map(User, UserOutputDTO.class);

**Dozer** - <https://dozer.sourceforge.net/>

Funciona como ModelMapper, en tiempo de ejecución, a través de reflexión escanea las clases y hace las conversiones adecuadas.