Refactorización para utilizar DTO

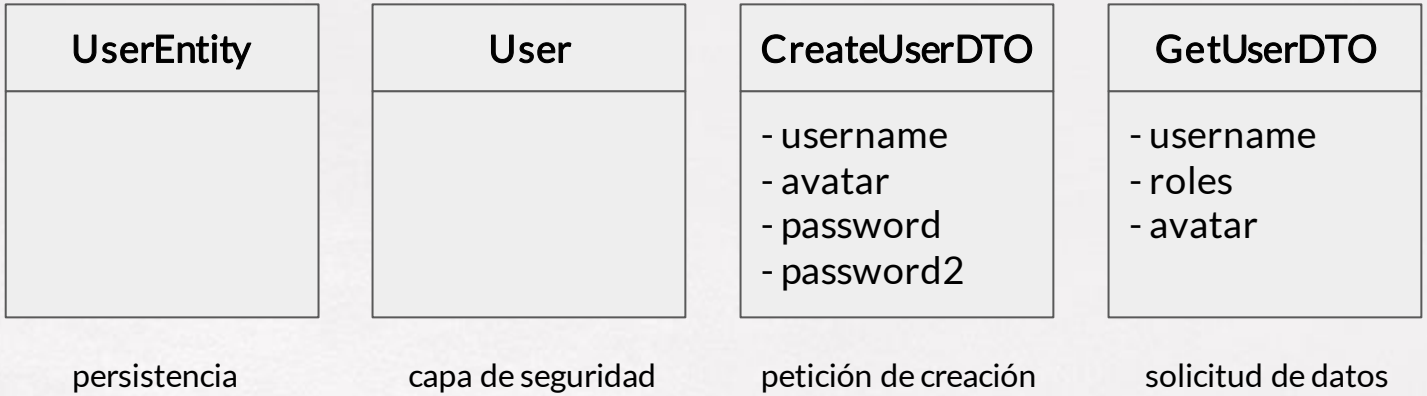
Implementa la seguridad de tu API Rest con Spring Boot

Clases modelo

- Nuestro modelo de usuario no es conveniente para crear un nuevo usuario.
  - Sería bueno recibir la contraseña por duplicado, para que podamos verificar si hay algún error o no.
  - Solo necesitamos algunos datos, no todos
- Tampoco es un buen candidato para la salida
  - Algunos datos aparecen más de una vez, dada la estructura del modelo (roles, authorities, ...)

Patrón DTO

- *Data Transfer Object*
- Sirve para transportar datos entre capas de un sistema



Gestión de errores

- Gestionamos en servidor la validación de la contraseña del usuario (que ambas contraseñas sean iguales)
- También gestionamos el intento de insertar un usuario duplicado (*@Column(unique = true)*)
- Tratamiento global y tratamiento local

Clases en la gestión de errores

- *NewUserWithDifferentPasswordsException*
  - Excepción para manejar la validación de la contraseña.
- *ApiError*, *ApiErrorAttributes*
  - Modelo de error para enviar una respuesta al cliente
- *GlobalControllerAdvice*
  - Tratamiento global de errores

Cambios en el código

- Nuevas clases CreateUserDto y GetUserDto
- Conversor de UserEntity a GetUserDto
- Cambios en el servicio y el controlador

Reto

- Es un buen momento para poder integrar el proyecto base (*00_ProyectoBase*) con el trabajo realizado hasta ahora.
- No servirá como nueva base para aprender los diferentes mecanismos de autenticación.
- El punto de unión entre ambos proyectos es modificar el modelo de pedido, para que el *cliente* sea un *UserEntity*.