

Go4Me

6. Implementación y Pruebas

Alejandro Megías Mata,
Alejandro Garau Madrigal,
Jesús Parejo Aliaga,
Raúl Morales Perujo,
Pedro Gallego Vela,
Manuel Veredas Galdeano.

V1.0

Implementación:	2
Pruebas:	3

Implementación:

Durante la implementación hemos trabajado con distintas tecnologías. El lenguaje de programación usado para el backend es *Java*, ya que consideramos que es un lenguaje que, para el problema planteado, trae mucha versatilidad y facilidades a la hora de implementar la base de datos con *JPA*, y la conexión de ésta con el servidor de *SpringBoot*. Además de eso, es un lenguaje que todo el equipo conoce, y por tanto, no supone ninguna pérdida de tiempo el aprendizaje de otras tecnologías.

Por otro lado, el lenguaje utilizado para el frontend es *HTML5*, junto con el motor de plantillas *Thymeleaf*, *Bootstrap 4.0* y *CSS3*. Hemos escogido esta combinación puesto que *Thymeleaf* funciona con *HTML*, además de su fácil aprendizaje y su facilidad para añadir estilos con *Bootstrap* y *CSS*. Creemos que *Bootstrap* es una herramienta fundamental hoy en día para la programación de aplicaciones web que permite tener un estilo profesional de manera sencilla sin necesidad de tener conocimientos avanzados de *CSS*. Para ejecutar el servidor de *Spring* y trabajar sobre el mismo utilizamos software de *JetBrains*; del mismo modo usamos *Atom* para que todos los miembros del equipo pudiesen programar de forma conjunta con su complemento *Teletype*.

Respecto a la aplicación, se decidió implementar un prototipo con el funcionamiento básico de la aplicación que permite dar una idea general de cómo funcionará la aplicación web una vez terminada, y de cómo será el estilo de la misma. Además, durante el transcurso de la implementación nos hemos encontrado con bastantes obstáculos, y, por falta de tiempo, se han decidido dejar sin implementar varias características del prototipo para ajustarnos lo mejor posible a la planificación. Las funcionalidades implementadas son el login, el registro, el perfil de un usuario y poder valorar su servicio, poder buscar usuarios, todo esto respecto al usuario. En relación con los pedidos, las funcionalidades son la creación y borrado, la información del pedido, la asignación y la búsqueda.

Las funcionalidades no implementadas serían el chat y los mensajes, el modificar un pedido, así como la funcionalidad de compartir en las redes sociales (principalmente Twitter)

Pruebas:

Las pruebas las realizaremos en *JUnit* y *Mockito*, tal y como indica la programación de la asignatura.

testPasswordCypher(): comprueba que cuando se guarda la contraseña en la base de datos se cifra. Para ello, vamos a comprobar que cuando obtienes la contraseña del objeto *Usuario* es distinto del resultado de cifrar la contraseña usando el *userRepository*.

testGetUserByUserName(): comprueba que cuando buscas a un usuario por su nombre de usuario se llama una única vez al método de la base de datos y el elemento devuelto es el nombre de usuario. Para ello, vamos a *mockear* la base de datos usando *userRepository* y estableciendo el comportamiento que vamos testear. En este caso, la base de datos devolverá el *User* cuando reciba cualquier cadena de caracteres. Luego, comprobamos que solo se llame una única vez al método *findByUserName*. Por último, se comprueba que el usuario recibido como resultado es igual al usuario dado.

testGetByEmail(): comprueba que cuando buscas a un usuario por su email se llama una única vez al método de la base de datos y que únicamente existe un usuario para ese email. Para ello, vamos a *mockear* la base de datos usando *userRepository* y estableciendo el comportamiento que vamos testear. En este caso, la base de datos devolverá el *email* cuando reciba cualquier cadena de caracteres. Luego, comprobamos que solo se llame una única vez al método *findByEmail*. Por último, se comprueba que el email recibido como resultado es igual al email dado.

testGetByID(): comprueba que cuando buscas a un usuario por su ID se llama una única vez al método de la base de datos y que únicamente existe un usuario para ese ID. Para ello, vamos a *mockear* la base de datos usando *userRepository* y estableciendo el comportamiento que vamos testear. En este caso, la base de datos devolverá el *ID* cuando reciba cualquier cadena de caracteres. Luego, comprobamos que solo se llame una única vez al método *findById*. Por último, se comprueba que el *ID* recibido como resultado es igual al *ID* dado.

testGetAllUsers(): devolver de la base de datos todos usuarios y comprueba que solo se llama una única vez al método de la base de datos y que el resultado es correcto. Para ello, vamos a *mockear* la base de datos usando *userRepository* y estableciendo el comportamiento que vamos testear. En este caso, la base de datos devolverá una lista vacía cuando se llame al método *findAll*. Por último, se comprueba que la lista recibida como resultado es una lista vacía.

testDeleteUser(): comprueba que se ha llamado una única vez al método *deleteById*, de la base de datos.