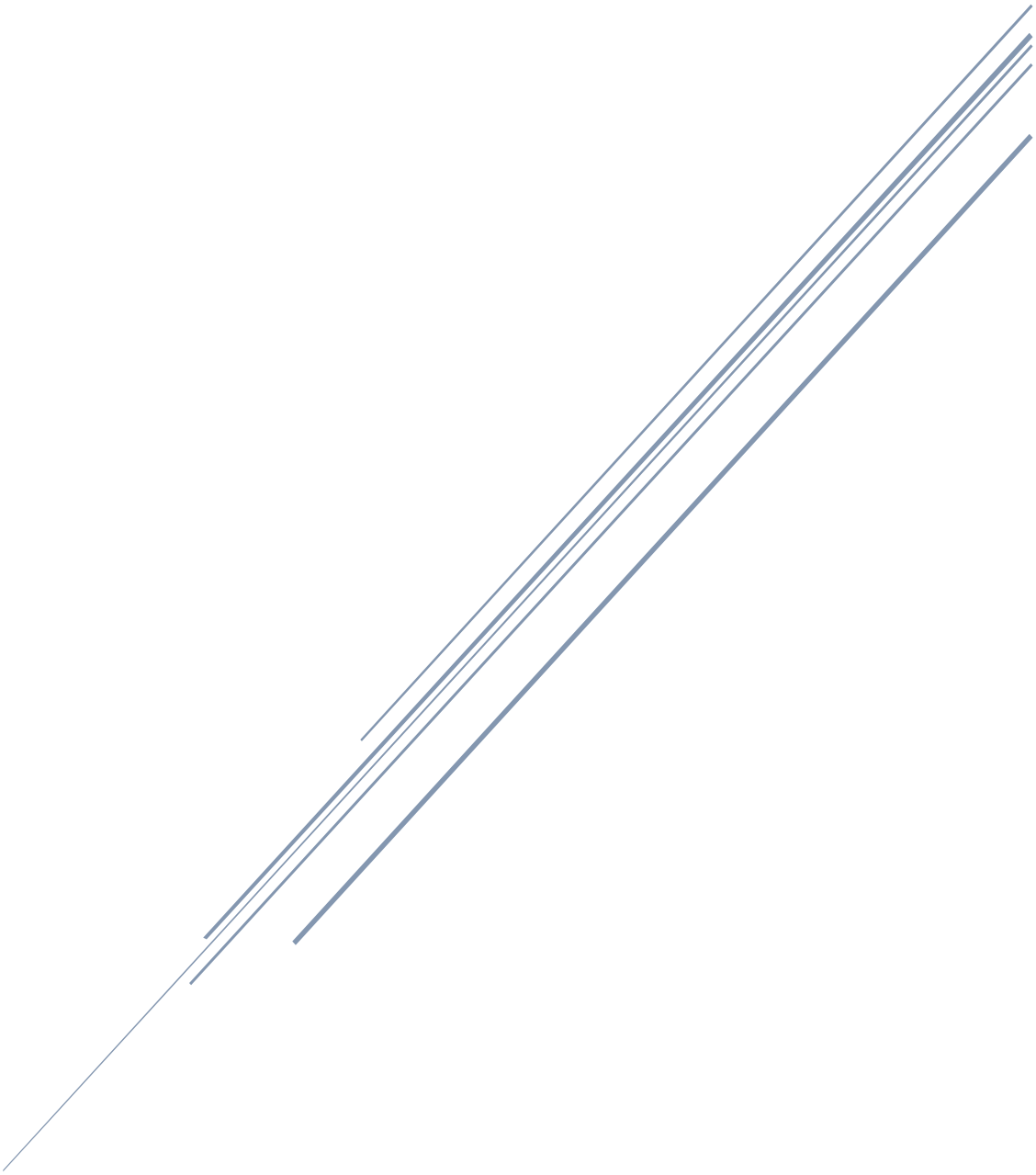
TESTING SHOP

Gonzalo Gómez Nogales



ETSII - URJC
Grado en Ingeniería Informática - AIS

Índice

Pruebas Unitarias con Dobles

La clase en la que se definen los tests unitarios se llama *DoublesUnitTests*, cuenta con 3 métodos para subdividir los casos de uso requeridos en diferentes contextos y 1 método con la anotación *BeforeEach* para crear los dobles de las clases que intervienen en el método *createPurchase* de la clase *PurchaseService*.

Antes de realizar cada test se crean todos los dobles mediante la función *mock* y se instancia un objeto de la clase *PurchaseService* para llamar al método de creación de compra.

El funcionamiento de estas pruebas es similar entre sí, salvo en algunos puntos concretos que se tratarán a continuación.

En primer lugar todos los tests crean un objeto de la clase auxiliar *Request*, esta clase se ha creado para encapsular los datos de las pruebas, pero podrían haberse diseñado de la misma forma sin necesidad de crearla. La decisión de incluir la clase *Request* es para aumentar la encapsulación y la reutilización de los datos de prueba.

Construido el objeto *request* se asigna un valor de retorno al producto en cuestión, es decir una vez se intente obtener el precio de un producto en concreto se devolverá el precio correspondiente.

Después, se establece la excepción a lanzar en el caso de que deba fallar el test, como en las pruebas en las que no hay stock y en la que el cliente no tiene suficiente dinero para pagar el producto que desea.

Se crea una compra invocando al método *createPurchase* asegurando que lanzan la excepción en los casos que sea necesario pasándole la aserción con una función lambda.

Finalmente, los tests verifican las veces que se ha ejecutado cada método en la llamada a *createPurchase,* con esto se consigue comprobar hasta donde ha llegado el método, si la compra se ha realizado correctamente y si se ha notificado el mensaje, o si por el contrario se ha quedado a medias y ha lanzado una excepción.

Pruebas de Sistema Web UI

En la clase *WebUISystemTest* se encuentran las pruebas de sistema con Selenium para probar la interfaz de usuario y los componentes visibles de la aplicación.

En primer lugar se define una lista de *WebDrivers* para crear diferentes sesiones del navegador en los casos que se necesitan crear varias simulaciones para una misma prueba, se ha elegido el navegador Firefox al ser el más común en equipos Linux como en Windows.

Estos tests inician la aplicación y la cierran de forma autónoma para facilitar su ejecución en servidores de integración contínua o a nivel de usuario para desarrolladores. Se consigue gracias a unos métodos anotados con *BeforeAll* y *AfterAll* que se ejecutan al comenzar y al finalizar los tests, además el diseño cuenta con otros 2 métodos que se ejecutan antes y después de cada prueba para configurar los navegadores iniciándolos y cerrándolos en el momento oportuno.

Al comienzo de cada prueba se instancia un objeto *request* con los datos necesarios para las pruebas a realizar (id del producto, id del cliente y mensaje de confirmación de la compra o error), se navega en el driver a la url de la aplicación y mediante selectores con Selenium se escoge el producto a comprar y se escribe la id del cliente que va a realizar la operación.

Por último simplemente se compara el mensaje devuelto al finalizar la compra en la pantalla para determinar el funcionamiento correcto de la aplicación, con esto se detectan los casos en los que devuelve mensaje de error y en cuales sale el mensaje de *Succesful Purchase*, confirmando la compra. Para asegurar que el mensaje se ha devuelto por pantalla durante el test se utiliza el método *until* de la clase *WebDriverWait* y de esta forma siempre se comprueba el mensaje cuando éste ya existe.

Pruebas de Sistema API REST

En la clase *RestfulAPISystemTest* se encuentran las pruebas a la API, no se utiliza la clase resumen Request.

En primer lugar se inicia la aplicación de forma similar a las pruebas anteriores con dos métodos anotados (uno para iniciarla y otro para cerrarla), y mediante el esquema **given-when-then** de la librería *RestAssured* se configura la petición de tipo post correspondiente al comportamiento de la aplicación que se pretende probar.

Para mandar la petición se utiliza el método *post*, previamente se configura el *body* y el *contentType* de la petición con la id del cliente que realiza la compra y con la id del producto que desea comprar.

Finalmente se comprueba el estado que devuelve la petición, 200 si se ha comprado el producto correctamente y 400 si la compra ha fallado con alguna de las dos causas posibles en las pruebas (no había stock o el cliente no tenía suficiente crédito), y el cuerpo devuelto tras la petición (se revisa el atributo id en el caso correcto y el mensaje en los que lanzan una excepción).

Configuración de Jenkins

Para configurar el *job* del servidor de integración continua (*Jenkins*) conectado con el repositorio de *GitHub* en el que se encuentra el código fuente de la aplicación, se ha utilizado un archivo *Jenkinsfile* ubicado en la raíz del proyecto y un archivo *log.txt* que recoge la salida de la ejecución de las pruebas en el servidor.

El archivo se ha configurado en su formato declarativo comenzando éste con la palabra *pipeline*, luego establece la herramienta necesaria al ser un proyecto *Maven* y se definen las etapas de ejecución.

La primera etapa (*Preparation)* clona el repositorio de *GitHub* usando las credenciales guardadas en el servidor con el identificador *devloper*.

Finalmente ejecuta las pruebas en la etapa *Test* según el sistema operativo con el comando correspondiente y en el apartado *post* se configura el trato que se le da a la salida de la ejecución de los mismos.