Actividad integradora 3EV
Desarrollo de una aplicación Web con Spring Boot
9 – Compra sin redirección con servicio REST

CONTENIDO

1.- Objetivos

- Objetivos de desarrollo web
  - Practicar el desarrollo de métodos de servicios web REST
- Objetivos complementarios
  - Practicar con JS, CSS, y/o frameworks de maquetación, para integrar llamadas a API en aplicaciones Spring Boot Web MVC.

2.- Requisitos

Haber completado todas las partes anteriores de la actividad, y, más específicamente, disponer de un carro de la compra funcional, que permita añadir productos al carro, y visualizar el contenido.

3.- Observaciones previas sobre la ausencia de usuarios y sesiones en la aplicación

De momento seguimos sin usuarios en el sistema. Así que se mantienen las mismas observaciones sobre la compra de productos:

- Los productos del carro de la compra no se asociarán a usuarios o sesiones de la web. Por así decirlo, de momento, habrá un único carro de compra "global".
- Este carro de compra "global" se compartirá entre todos los usuarios. Si dos usuarios accedieran a la vez a la vez a la aplicación (ejemplo: abrir dos navegadores), y añadieran productos al carro, los dos verían añadidos estos productos.
- Se mantienen las restricciones en el carro de compra sobre duplicidad de productos, que habrá que modificar en un futuro, cuando se incorporen usuarios al sistema.

4.- Tareas a realizar

4.1.- Crear un nuevo paquete para controladores REST, y controlador para los servicios del carro de compra

Como ya se dispone del paquete "controllers" donde se encuentran los controladores para genera páginas con Spring Web MVC, para mantener cierto orden, crear un nuevo paquete "restcontrollers".

Dentro de este paquete, crear un nuevo controlador para los servicios REST relacionados con el carro de compra. Dar un nombre adecuado al controlador. Por, ejemplo, si ya se dispone de un controlador "CartController", podría ser "CartRestController".

Observación: puede que este controlador desaparezca en un futuro, si se decide "fusionar" toda la funcionalidad del carro en un único controlador. Pero de momento, se mantendrá separado para poder trabajar más cómodamente.

Anotar el controlador con @RestController, para activar la devolución de datos por defecto.

4.2.- Crear un DTO para recibir los datos del producto a añadir

Crear un DTO (POJO que sólo se usa para transferir datos entre cliente y el servicio) para recibir los datos del producto a añadir. El DTO tendrá:

- El id del producto que se quiere añadir al carro.
- El número de unidades que se quieren añadir al carro.

Este DTO se usará en las llamadas para añadir productos al carro.

4.3.- Crear un método para añadir un producto al carro

Crear un método en el nuevo controlador REST para añadir un producto al carro de compra.

Este método recibirá un objeto del tipo DTO definido en el apartado anterior, y utilizando métodos de servicios y/o repositorios:

- Comprobará que el producto existe. Si no existe devolverá un 404 (Not found) con un mensaje "No existe el producto con código <código>"
- Comprobará que el número de unidades en stock serían suficientes en el caso de que se completase el pedido. Si no hay suficientes, devolverá un error 409 (Conflict), con un mensaje "No hay suficientes unidades. Sólo hay <cuantas> en stock." Para esto, debe tenerse en cuenta si ya se habían añadido al carro unidades de ese producto. Se debe poder entregar el TOTAL de unidades.
- Si las dos precondiciones se cumplen, se devolverá un código 200 con el mensaje "Producto añadido".

A la hora de desarrollar esta nueva funcionalidad, tener en cuenta los siguientes requisitos.

- El grueso de la funcionalidad debe estar en el servicio. El controlador debe quedar lo más ligero posible. En este caso, el controlador se limitará a llamar a un método del servicio, y a capturar excepciones para lanzar los mensajes de error adecuados.
- El método de servicio:
    - Comprobará el id de producto, y si no existe, lanzará una excepción "EntityNotFoundException", que el controlador debe gestionar en un catch.
    - Comprobará el stock, y si no ha suficiente, lanzará una excepción personalizada (hay que crear una nueva clase), que el controlador debe controlar.
    - Si todo va bien, simplemente terminará sin incidencias.
- La lógica para añadir productos al carro será similar a la implementada. Si ya se había añadido el producto al carro, se incrementa el número de unidades, si no, se añaden las indicadas. En ambos casos solo si hay stock suficiente.

4.4.- Probar el método.

Usando el cliente HTTP de IntelliJ, realizar llamadas para probar todos los casos posibles. Por ejemplo:

- Añadir un nuevo producto al carro, y que funcione sin problemas.
- Incrementar unidades de un producto, y que funcione sin problemas.
- Añadir un producto que no existe
- Añadir un producto sin stock
- Etc.

Antes de proceder a realizar una integración con HTML y JS, es importante verificar que los métodos funcionan si se llaman correctamente.

4.5.- Integración con la aplicación web

Integrar la funcionalidad en la aplicación, haciendo que, al hacer clic en los enlaces / botones para añadir al carro, no se haga una petición HTTP que recargue página (con redirección en servidor), sino que:

- Se haga una llamada POST al servicio, con un objeto adecuado.
- Se recoja el resultado (código y mensaje)
- Si ha ido todo bien, se muestre un mensaje indicándolo.
- Si ha habido error se muestre otro con los detalles del error.

Para hacerlo, usar cualquier tecnología que se considere adecuada. JavaScript será necesario, pero para los diálogos / mensajes se puede usar utilidades integradas en ciertos frameworks. Por ejemplo:

- Con toasts:
  - Bootstrap toasts: https://getbootstrap.com/docs/5.3/components/toasts/
  - Toastr.js (necesita jQuery)
  - SweetAlert2
  - Otras alternativas a Bootrstrap Toast
- Con ventanas modales:
  - Bootstrap modal: https://getbootstrap.com/docs/5.3/components/modal/
  - SweetAlert2
  - Micromodal.js
  - Otras alternativas a Bootstrap modal.

En cualquier caso, no se puede usar alert de javascript. Es un método que los navegadores pueden bloquear si se producen muchos en un sitio.