Introducción

El objetivo de este Bootcamp es la construcción de una **API Rest** utilizando **Spring boot** como framework de desarrollo. Para ello, se provee un **repositorio en gitlab** con un proyecto que sirve como base para desarrollar los requerimientos funcionales. Este repositorio se encuentra restringido y, por ende, el desarrollador del bootcamp deberá crear un *fork* para trabajar.

Los progresos y consultas serán concentrados en una reunión diaria con el *coach asignado*. El horario de dichas reuniones será de mutuo acuerdo. **Se estima para la finalización del bootcamp entre 2 y 3 semanas.**

Requerimiento funcional

Se necesita un sistema de comercio electrónico. Un sistema de comercio electrónico, mejor conocido como E-commerce, es un sistema de compra y venta de productos o servicios que se realiza exclusivamente a través de Internet. Se refiere a las transacciones entre compradores y vendedores mediante una plataforma online que gestiona los cobros y los pagos de manera completamente electrónica. Estos tipos de sistemas presentan múltiples módulos que se enfocan en distintos subdominios del problema: finanzas, stock, etc. Como primer requerimiento se desea desarrollar el carrito de compras del sistema. El carrito de compras se responsabiliza de llevar control de los productos y cantidades que un usuario desea comprar. A su vez contiene la información del precio total a pagar por dicho carrito de compras.

Teniendo en cuenta estos requerimientos desde el lado del backend se necesita API que haga una ABM de este carrito de compra. La API debe contener servicios para crear un carrito de compras (con al menos un producto), agregar productos al carrito de compra, eliminar productos, calcular el precio total del carrito de compra y finalizar la compra de un carrito.

En una reunión de planeamiento para el nuevo requerimiento, el equipo de desarrollo acordó utilizar las siguientes tecnologías:

1. Spring Boot.
2. Postgresql
3. Docker

La idea con estas decisiones está en manejar tecnologías open-source que faciliten el desarrollo. A su vez, se establece como convención de arquitectura, la estructura de una arquitectura hexagonal que dirija todo el desarrollo.

Finalmente, teniendo en cuenta la metodología scrum en la cuál se espera el desarrollo de un producto iterativo e incremental se espera que la funcionalidad está divida en varias etapas la cuál necesitará el uso de mocking de datos para hacer entregas funcionales.

User Stories

Ver un carrito de compra

Se debe diseñar e implementar una API para devolver el listado de productos y cantidades que el carrito contiene actualmente.

Criterios de aceptación

- **En caso de no contener productos se puede responder una lista vacía.**

Creación de carrito de compra

Se debe diseñar e implementar una API que permita crear un carrito de compra para un usuario específico. Este carrito de compra debe contener una lista de códigos de productos con sus cantidades. El sistema debe validar que el carrito de compra contenga al menos un producto.

Criterios de aceptación

- **Documento de diseño del endpoint.**
- **El sistema debe validar que el carrito de compra contenga al menos un producto.**
- El sistema debe validar que el código de producto pertenezca a un producto existente. En caso de no existir el código se debe arrojar un error indicando el problema.
- El sistema debe validar que la cantidad de productos sea válida. En caso de que la cantidad supere al stock entonces debe arrojar un error indicando el problema.
- Las respuestas de error deben ser coherentes al problema existente.
- En caso de éxito se debe devolver el código del carrito creado.
- Test unitario del caso de uso

Agregar productos al carrito de compra

Se debe diseñar e implementar una API que permita agregar un producto con su cantidad a un carrito de compra de un usuario específico.

Criterios de aceptación

- Documento de diseño del endpoint.
- El sistema debe validar que el carrito de compra exista y pertenezca al usuario asociado. En caso de no existir el código se debe arrojar un error indicando el problema.
- El sistema debe validar nuevamente que el producto con su cantidad no supere el stock disponible.
- Test unitario del caso de uso

Eliminar productos del carrito de compra

Se debe diseñar e implementar una API que permita eliminar un producto con su cantidad a un carrito de compra de un usuario específico.

Criterios de aceptación

- Documento de diseño del endpoint.
- El sistema debe validar que el carrito de compra exista y pertenezca al usuario asociado. En caso de no existir el código se debe arrojar un error indicando el problema.
- Test unitario del caso de uso

Calcular costo total de un carrito de compra

Se debe diseñar e implementar una API que dado un código de carrito perteneciente a un usuario se calcule el costo total de la compra.

Criterios de aceptación

- Documento de diseño del endpoint.
- El sistema debe validar que el carrito de compra exista y pertenezca al usuario asociado. En caso de no existir el código se debe arrojar un error indicando el problema.
- Test unitario del caso de uso

Finalizar compra

Se debe diseñar e implementar una API que dado un código de carrito perteneciente a un usuario se finalice la compra. La finalización de la compra consiste en la actualización del stock de los productos del sistema. A su vez se debe marcar el carrito de compra como procesado. Este estado inhabilita nuevos cambios al mismo (eliminar, agregar productos)

Criterios de aceptación

- Documento de diseño del endpoint.
- Al momento de finalizar la compra se debe validar que el stock quede consistente (no queden valores negativos de producto).
- Test unitario del caso de uso

Realizar persistencia de los datos

Se deben actualizar los casos de uso previos para permitir el almacenamiento efectivo en una base de datos. De esta forma el sistema mantendrá estado cuando se reinicie.

Para este escenario se debe utilizar el container de postgresql que se encuentra en el repositorio.

Workflow

Los requerimientos son presentados en forma de **User Stories**. Estas deben ser analizadas y desglosadas en **tareas** concretas de desarrollo para así poder brindar *visibilidad* del trabajo en progreso y el trabajo pendiente. El ***proyecto* particular** dentro de **Taiga** donde se realice el seguimiento y avance del bootcamp **quedará a elección del *Scrum Master* del proyecto al que ingresa el nuevo integrante**.

Se espera que se tenga una mentalidad orientada a la integración y entrega continua de funcionalidades (*CI/CD*). Por ende, los commits en el repositorio se espera que sean pequeños y legibles para poder comprender una nueva versión estable del sistema que se está desarrollando. Cada uno de estos commits debe corresponder con una tarea en Taiga mediante un identificador como prefijo en el mensaje del commit.

Ej: **tg-[nro_tarea_taiga] [Descripción de la funcionalidad que incorpora]**

De esta manera se puede tener *trazabilidad* entre los requerimientos y los desarrollos incluidos en el repositorio.

El repositorio cuenta con *pipelines* de gitlab para correr los tests del proyecto a la vez que corren chequeos de estilos de código (*ESlint).* Cada commit que no pase estos chequeos será anunciado en los pipelines y no podrá ser mergeado en la rama principal **master.** Se recomienda revisar estos chequeos en el ambiente local antes de realizar un commit.