INFORME DE TEST UNITARIOS BACKEND – SPRINT III

MS-CART

Los tests unitarios en el backend se están realizando con base en el framework Junit. Para este sprint se inició con su implementación en el microservicio Cart, por lo que a continuación detallaré lo realizado.

- **CartServiceImplementsTest**
  En esta clase se realizó la validación de los métodos traer todos los productos, adicionar y remover producto al carrito. Es importante aclarar que se validaron casos positivos como negativos con el fin de dar una mayor cobertura al testing.

Nombre y descripción	Código del Test	Resultado

1. **mustGetAllProductsInCart()**: Valida el escenario en que se recuperen todos los productos que están en el carrito. Se utiliza la funcionalidad assertEquals para validar que la lista que se guardó en el carrito sea igual a lo que se está recuperando.	`@Test`
	`void mustGetAllProductsInCart(){`
	`  List<Product> product = FactoryProductDataTest.getProducts();`
	`  List<Cart> cart = FactoryCartDataTest.getCarts();`
	
	`  List<String> productIds = cart.stream()`
	`  .map(Cart::getProduct)`
	`  .collect(Collectors.toList());`
	`  String productIdsString = String.join(",", productIds);`
	
	`  Mockito.when(cartRepository.findByClient(clientId)).thenReturn(cart);`
	`  Mockito.when(productClient.findMultipleProducts(productIdsString)).thenReturn(product);`
	
	`  List<Product> productsList = cartServiceImplements.getAllProductsInCart(clientId);`
	`  assertEquals(productsList, product);`
	`}`
		✔Test passed

2. **mustNotAddProductToCart()**: Valida que si el producto ya está adicionado al carro no permite que nuevamente lo adicione.	`@Test`
	`void mustNotAddProductToCart(){`
	`  Cart cart = FactoryCartDataTest.createCart();`
	
	`  Mockito.when(cartRepository.findByClientAndProduct(clientId, productId)).thenReturn(Optional.of(cart));`
	`  Assertions.assertThrows(BadRequestException.class, () -> cartServiceImplements.addProductToCart(cart));`
	
	`}`
		✔Test passed

3. **mustAddProductToCart()**: Valida que si se puede agregar un producto al carrito si este no ha sido adicionado previamente.	`@Test`
	`void mustAddProductToCart() throws BadRequestException {`
	`  Cart cart = FactoryCartDataTest.createCart();`
	
	`  Mockito.when(cartRepository.findByClientAndProduct(clientId, productId)).thenReturn(Optional.empty());`
	`  Mockito.when(productClient.isProductAvailable(cart.getProduct(), cart.getQuantity(), false)).thenReturn(true);`
	
	`  cartServiceImplements.addProductToCart(cart);`
	`  Mockito.verify(cartRepository).save(Mockito.any(Cart.class));`
	`}`
		✔Test passed

4. **mustNotAddProduct ToCartIfThereIsNotStock()**: Valida que no se pueda agregar un producto al carrito si no hay stock.	`@Test`
	`void mustNotAddProductToCartIfThereIsNotStock() {`
	`  Cart cart = FactoryCartDataTest.createCart();`
	
	`  Mockito.when(cartRepository.findByClientAndProduct(clientId, productId)).thenReturn(Optional.empty());`
	`  Mockito.when(productClient.isProductAvailable(cart.getProduct(), cart.getQuantity(), false)).thenReturn(false);`
	
	`  Assertions.assertThrows(BadRequestException.class, () -> cartServiceImplements.addProductToCart(cart));`
	`}`
		✔Test passed

5. mustRemoveA ProductFromTheCart(): Valida que se pueda eliminar un producto del carro en caso de que exista.	
	`@Test`
	`void mustRemoveAProductFromTheCart() {`
	`Cart cart = FactoryCartDataTest.createCart();`
	
	`Mockito.when(cartRepository.findByClientAndProduct(cart.getClient(),cart.getProduct())).thenReturn(Optional.of(cart));`
	
	`cartServiceImplements.removeProductFromCart(cart.getClient(),cart.getProduct());`
	`Mockito.verify(cartRepository).deleteByClientAndProduct(cart.getClient(), cart.getProduct());`
	`}`
		✔Test passed