

Universidad ORT Uruguay

Facultad de Ingeniería

Obligatorio 1 - Diseño de Aplicaciones 2

Evidencia de la ejecución de las pruebas de la API con POSTMAN

 **GitHub** <https://github.com/IngSoft-DA2-2023-2/266628-255981-271568>

Federico Rodriguez - 255981

Santiago Salinas - 266628

Manuel Morandi - 271568

Tutores: Daniel Acevedo

Rafael Alonso

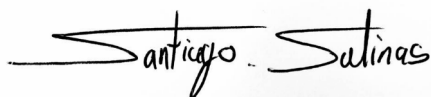
Franco Galeano

Septiembre 2023

Declaración de autoría

Federico Rodriguez, Santiago Salinas y Manuel Morandi declaramos que el trabajo que se presenta en esta obra es de nuestra propia mano. Podemos asegurar que:

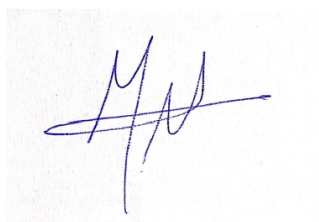
- La obra fue producida en su totalidad mientras construimos el primer obligatorio de Diseño de Aplicaciones 2
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad
- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra
- En la obra, hemos acusado recibo de las ayudas recibidas
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros y qué fue contribuido por nosotros
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes



Santiago Salinas 26/9/2023



Federico Rodriguez 26/9/2023



Manuel Morandi 26/9/2023

Abstract

Reporte que muestra evidencia del resultado de ejecutar los casos de prueba especificados para la API con Postman para determinadas funcionalidades de las definidas en la letra del obligatorio.

Esto incluye: valores inválidos, valores límites, ingreso de tipos de datos erróneos, datos vacíos, datos nulos, omisión de campos obligatorios, campos redundantes, Body vacío {}, formato de los mensajes inválidos, pruebas de las reglas del negocio como alta de elementos existentes, baja de elementos inexistentes, etc.

Además se encuentra un link a un video publicado en Youtube en donde se pueden ver las ejecuciones de las pruebas.

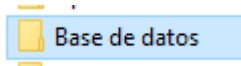
Abstract	2
Link Video Youtube Postman Tests	4
Cargar Datos de Prueba	4
Sobre el database incluido	5
Documentación de pruebas de funcionalidades prioritarias	5
Usuarios	6
Productos	8
Marca, Categoría y Color	8
Manejo de compras	10

Link Video Youtube Postman Tests

<https://www.youtube.com/watch?v=zDB7hVJ7758>

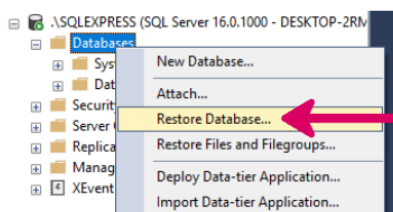
Cargar Datos de Prueba

Para todos: Cargar una base de datos .bak

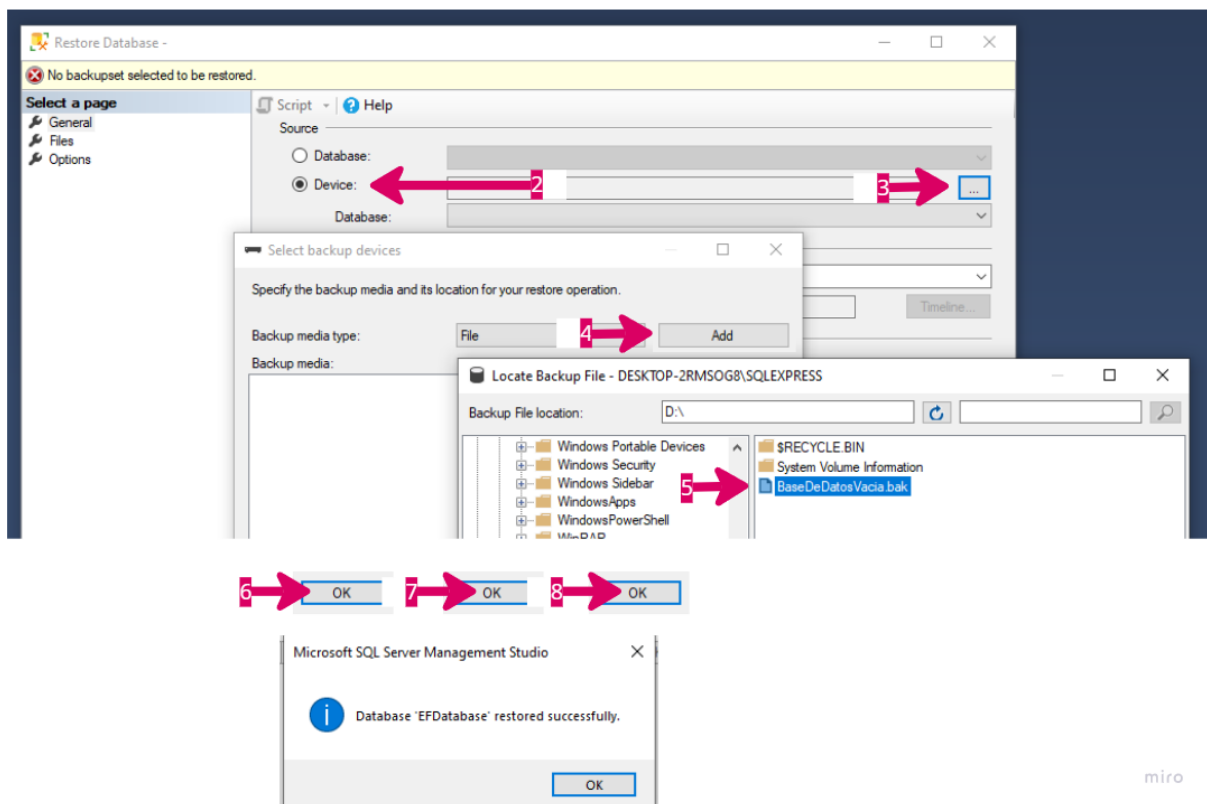


← En esta carpeta se encuentran los .bak y .sql

Se recomienda mover a disco C: o D: de alto nivel, ya que SQL solo permite acceder hasta tres niveles de carpetas en el buscador.



Para cargar una Base de Datos, click derecho sobre la carpeta databases.
Click en Restore Database...
Source seleccionado en Device
Add...
Y buscamos el archivo .bak proporcionado
OK OK



Sobre el database incluido

A continuación, se indican los tokens de los distintos usuarios incluidos en los datos de prueba y sus roles:

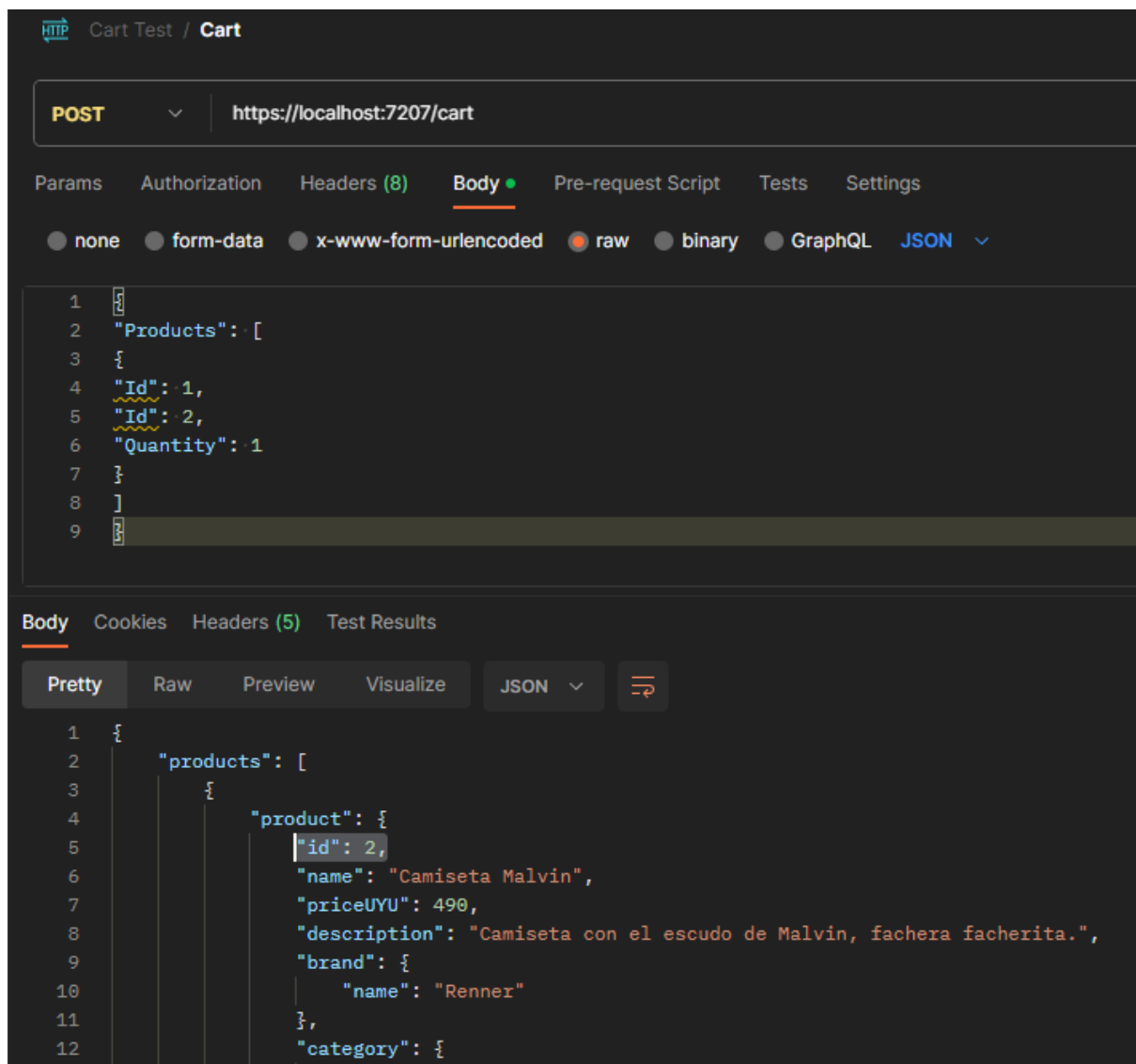
Token	Roles
tokenbwayne@gmail.comsecure	Admin y Customer
tokenclarkent@gmail.comsecure	Customer
tokendiana@gmail.comsecure	Customer
tokenollieq@outlook.comsecure	Admin
tokenharls@hotmail.comsecure	-

Documentación de pruebas de funcionalidades prioritarias

Pese a que las pruebas escritas en Postman ya son lo suficientemente claras como para entender los distintos casos de prueba posibles, hagamos hincapié en aquellos pertenecientes a las funcionalidades marcadas como prioritarias.

Antes de comenzar con cosas más generales, conviene hablar de los casos bordes, que se asumen que se comportan de igual manera para todos los tests. Por ejemplo, si en el JSON colocamos un campo redundante funcionará en cascada, tomando el último valor que se le asignó. Se adjunta una imagen para ejemplificar

esto.



Frente a la asignación de valores de tipos erróneos, se recibirá un error del tipo `BadRequest`. Finalmente, si un dato no se incluye (ej: `"id" = ,`; o directamente ni se pone el campo de `id`), lo que sucederá es que se le asignará el valor por defecto para su tipo (si es un valor numérico se trata del 0, si es un string es el string vacío). En estos casos, la request se realizará, pero si el sistema exige que no sea vacío, la propia lógica de la aplicación lanzará un error (puede ser `BadRequest` u otro).

Usuarios

Los usuarios pueden llevar a cabo 4 operaciones sobre su propia cuenta: crear cuenta, modificar y eliminar su usuario y obtener su información. En lo que

respecta al Get, lo único relevante es que el header de la request incluya el token del usuario, ya que será usado para obtener la información. Si no se pasase ningún header o el token fuera incorrecto, se mostrará un mensaje de error. Lo mismo sucede con el Delete.

Las otras dos operaciones (Post y Put) son más interesantes, ya que el request debe incluir un body. Este body debe incluir email, contraseña, y dirección.

En el caso de un Put para actualizar un usuario, se requiere de que en el header se envíe el token asociado al usuario en cuestión en la base de datos.

Cualquier valor vacío debería resultar en un error, así como un formato incorrecto (en el caso del email deberá tener el formato típico del mail electrónico, [nombreUsuario]@[dominio]). Para crear un usuario no es necesario estar registrado, por lo que no se enviará el token en el header. También debe tomarse en cuenta que intentar crear o actualizar un usuario con un email ya ocupado en la base de datos resultará en una response BadRequest.

Los usuarios también deben poder loguearse (el LogOut como tal consta de dejar de enviar el token en el header, entendemos que se trata de una funcionalidad a implementar en el frontend, o en el cliente). El LogIn se hace desde un endpoint del mismo nombre, a través de un Post. Se recibe en el body el email, password y token del usuario a loguearse. Si los datos son correctos se recibe un código 200, pero si las credenciales recibidas no coinciden con las almacenadas en la base de datos, se lanzará un error 400.

Operaciones similares a las de User se pueden dar desde el panel de Admin. Aquí, primero que nada, todas las operaciones reciben el token del usuario que está realizando la operación en el header, porque hay que verificar que realmente sea un administrador y está autorizado a realizar la transacción (si no lo está se lanzará un error). Un administrador puede consultar la información de un usuario específico, obtener la lista de todos los usuarios disponibles, así como también la lista de las compras realizadas por todos los usuarios. Si solicita información de un usuario inexistente, recibirá un error 404.

Encontramos diferencias a la hora de que un administrador registre o modifique un usuario, ya que, adicionalmente a los datos incluidos en el body

anteriormente expresado, deberá asignarle roles. Si se introduce una lista vacía, el usuario no tendrá roles, y todos los roles incluidos en la lista serán adjudicados al usuario. Debemos recordar que los únicos roles soportados son Admin y Customer, por lo que establecer cualquier otro rol resultará en un error. Finalmente, eliminar un usuario funciona igual, pero si el id del usuario es inválido se recibe 404.

Productos

La mayoría de operaciones relacionadas con el manejo de productos están limitadas a ser utilizadas únicamente por usuarios con el rol de Admin. Más específicamente, puede crear, modificar y borrar productos. Los procedimientos son similares a los anteriormente vistos. Si el id no es válido retorna 404, si la id del producto a modificar no coincide con la del parámetro, BadRequest. El body del producto cuenta con id, nombre, precio, descripción, marca, categoría y una lista de colores. Recordar que las marcas, categorías y colores deben estar almacenadas en la base de datos para el correcto funcionamiento. Naturalmente, establecimos que el precio asignado a un producto no puede ser igual o menor que cero, y el nombre de un producto debe ser único en la base de datos, por lo que una operación que contradiga estas condiciones resultará en un BadRequest.

Como siempre, si el token recibido en el header no es válido, o no es perteneciente a un Admin, nos toparemos con un error de autorización.

Las únicas operaciones que cualquier usuario puede realizar son las de obtener todo el listado de productos disponibles o un listado filtrado por cualquier combinación de nombre, categorías o marca. Estas dos operaciones no tienen errores específicamente asociados, por lo que se recibirá como respuesta siempre un código 200, ya sea una lista con productos o una lista vacía si no hay productos o ninguno cumple con los parámetros del filtrado.

Marca, Categoría y Color

Debido a la falta de operaciones de creación, modificación o eliminación, estos endpoints son los más simples de toda la api. Las únicas request que se

pueden hacer son Get, ya sean para conseguir todos los elementos de una tabla o uno en específico. Ni siquiera requieren de autorización o autenticación del usuario para realizar la operación, por lo que las únicas respuestas posibles son un código de estado 200 con un body conteniendo los elementos disponibles, o un error 500 debido a que una operación falle inesperadamente.

Manejo de compras

Para el manejo de compras tenemos varios endpoints: primero el endpoint `/cart` el cual permite manejar el carrito incluso si no se es un usuario en la base de datos, pues se pueden agregar productos al carrito incluso sin haber iniciado sesión.

Este endpoint permite mandar un carrito mediante un Post con productos al backend y recibir una response 200 que incluye el precio final y la promoción aplicada, o un 400 BadRequest en el caso de que el body CartDTO enviado incluya parámetros incorrectos como un producto que no existe en la base de datos o una cantidad de productos negativa.

Sin embargo, si se desea realizar la compra, se mandaría un Post del mismo CartDTO esta vez al endpoint `/cart/buy`, incluyendo en el header el token del usuario realizando la compra de manera que se pueda verificar que tiene el rol de Customer requerido para la operación. De esta request se pueden recibir varias responses: 200 si se completa correctamente, 403 si el usuario no tiene el permiso requerido, o incluso un 400 si los parámetros del carrito están incorrectos o no tienen ningún producto dentro.

Aunque esta operación completada correctamente resulta en una nueva entrada a la tabla de compras registrada en la base de datos, los carritos en si no se persisten, por lo que no existen más operaciones que se puedan realizar en los endpoint de `/cart`.

Adicionalmente a manejar un carrito, un usuario customer puede ver su propio historial de compras realizando un Get al endpoint `/purchase/history`, siempre y cuando agregue su token correctamente en el header. Esto se espera que resulte en un 200 con la lista de compras del usuario, ya sea que tenga compras o esté vacía.

Un administrador a su vez puede realizar un Get a `/purchase/{id}` con el id de la compra en cuestión para recibirla con un response 200, siempre y cuando el token agregado al header corresponda al de un administrador. En caso contrario, puede recibir un 403 si el usuario no tiene el rol requerido o un 404 si el id no corresponde a ninguna compra. Si desea ver todas las compras, un get a `/purchase`

retornaría un response 200 con todas las compras de la base de datos, mientras el que realice la operación sea un administrador.

Al llevar a cabo cualquier operación en el carrito, se aplicará sobre él la promo que mejor se ajuste. En otras palabras, se tomarán todos los productos y se probarán las promos una a una, para luego tomar la que maximiza el ahorro del cliente. Estas promos se pueden ver en las pruebas generando distintos carritos y operando sobre ellos.