

100 Ladrillos Challenge

Elaborado por: M.C. Luis Eduardo Villela Zavala

Introducción:

100 Ladrillos Challenge es un proyecto solicitado como parte de la evaluación del proceso de reclutamiento para poder incorporarme al equipo de trabajo. Por medio del presente proyecto se tiene la posibilidad de poner en práctica mis conocimientos previamente adquiridos y mi capacidad para diseñar soluciones a una problemática planteada y una serie de requerimientos por cumplir.

El presente documento se clasifica en distintas secciones que mostrarán explicaciones y capturas del funcionamiento del sistema desarrollado.

Cabe destacar que el objetivo es evaluar los conocimientos de backend, por lo que se hará especial énfasis en esta área; aunque también se desarrolló un pequeño index como parte del frontend.

Objetivo General:

Elaborar un sistema de carrito de compras por medio del cual se podrán adquirir participaciones en propiedades, mejor conocidos como “Bricks” a un precio establecido.

Objetivos Específicos:

- Gestionar las funciones necesarias para administrar un carrito de compras (Agregar bricks, eliminar bricks, concretar compra, modificar stocks, etc).
- Manejar control de inventario cuando dos o más usuarios quieren añadir el mismo brick con segundos o minutos de diferencia.
- Practicar la elaboración de API Rest básicas para un caso de prueba determinado (Carrito de compras).

Descripción del Proyecto Solicitado:

Consultar documento de instrucciones.

Tecnologías Utilizadas:

Backend:

- NextJS: Es un framework de ReactJS que permite crear aplicaciones web de una forma práctica por medio de bloques que contienen distinta funcionalidad.
- JavaScript: Lenguaje de programación utilizado durante prácticamente la totalidad del proyecto.

- MongoDB: Motor de base de datos no relacional para la gestión de los datos del proyecto.

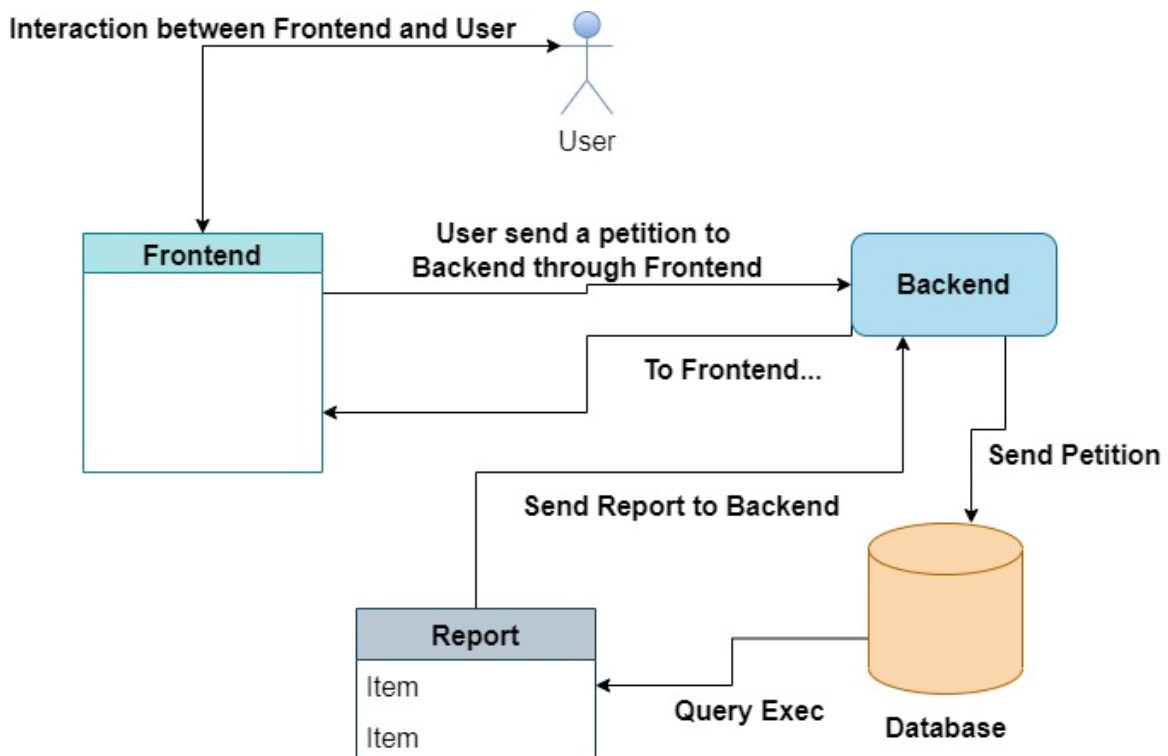
Frontend:

- ExpressJS: Es un framework basado en NodeJS caracterizado por ser ligero y contener requerimientos mínimos para poder ser empleado en proyectos de desarrollo web. Incluye manejo de vistas, capacidad de enrutamiento y uso de librerías para hacer conexiones con el backend y poder llevar a cabo el intercambio de datos.
- JavaScript: Lenguaje de programación utilizado durante prácticamente la totalidad del proyecto.

Diagrama de Diseño Técnico:

Se incorpora un pequeño diagrama de diseño técnico, donde muestro de forma generalizada cuáles son los componentes con los que este proyecto cuenta y la forma en cómo interactúan entre ellos una vez que el sistema está en ejecución.

Es importante recordar que el usuario está interactuando con el sistema por medio del Frontend; y genera peticiones aquí hacia el Backend dependiendo de lo que desee realizar.

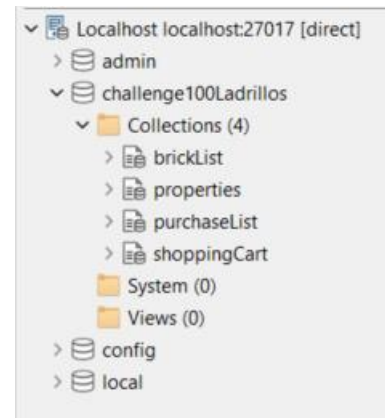


Base de Datos:

La base de datos está desarrollada en MongoDB, utilizando un enfoque no relacional.

Consiste de 4 colecciones distintas:

- brickList: Contiene un listado de los bricks que están disponibles para su venta. Es en esta colección donde se lleva el control del stock de los bricks disponibles para su venta y de los bricks que ya han sido vendidos; así como los que están en proceso de venta, es decir, aquéllos que están dentro de algún carrito de compras.
- properties: En esta colección se almacenan las propiedades a las cuales están asociados los bricks. Dentro de esta colección también se incluyen campos como la ubicación de la propiedad y el stock disponible.
- purchaseList: Esta colección lleva un registro de las ventas de bricks concretadas. De inicio está vacía, se llena conforme se van ejecutando y concretando ventas.
- shoppingCart: Es aquí donde se almacenan los registros de los bricks que el usuario está añadiendo a su carrito de compras. Está vacía de comienzo, se llena conforme el usuario añada bricks a su carrito; y se vacía una vez que se concreta la venta o se cancela una venta por rechazo de los términos y condiciones.



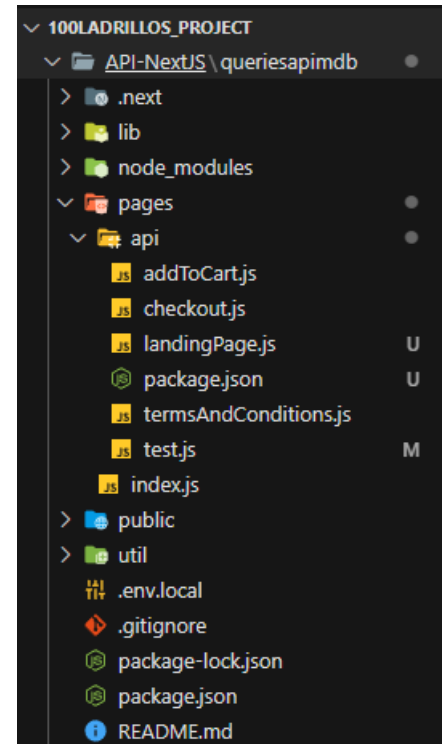
Estructura del Repositorio de Código:

Backend:

El código correspondiente al Backend se encuentra en la carpeta `queriesapimdb`, dentro de esta carpeta se encuentra el proyecto de NextJS, el cual está compuesto de múltiples directorios, de los cuales muchos de estos fueron generados por el framework.

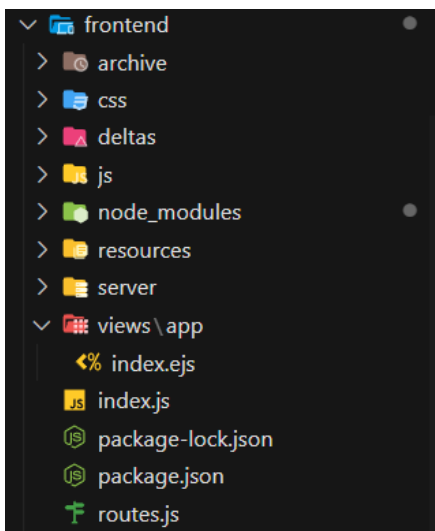
El directorio dentro del cual se realiza toda la lógica de la aplicación es `pages`, adentro de este archivo se generan archivos de JavaScript que funcionan como endpoints, dependiendo de lo que se desee realizar. Existen 5 endpoints distintos, cada uno realiza una funcionalidad distinta:

- `test.js` → Este endpoint fue creado solo con el propósito de probar la conexión a la base de datos una vez que se creó la conexión a MongoDB por medio de NextJS. Solamente retorna en un JSON lo que se tiene dentro de cada colección.
- `landingPage.js` → Endpoint que es el encargado de cargar la información inicial que será mostrada dentro del frontend para posteriormente ser formateada y de esta manera tener solo los datos necesarios para mostrarlos. Los datos procesados en este punto son mostrados en el Frontend simple que se creó en formato de tabla.
- `addToCart.js` → Este endpoint es el más importante de este proyecto, dado que gestiona todas las consultas correspondientes a la adición de los bricks al carrito de compras. Entre las tareas que realiza se encuentran:
 - Validación de stocks de bricks: Verifica que haya existencias disponibles para su compra y que no se hayan vendido al momento de querer seleccionarlos. La forma en como se gestiona esta parte es por medio del campo `OnSaleProcess` de la colección `brickList`, que lleva un conteo de las unidades que están almacenadas en carritos de compra. Para verificar que aun hay existencias disponibles se realiza una resta de los bricks disponibles menos los que están en proceso de venta. Si es mayor que cero entonces aun quedan bricks disponibles y procede a añadirlo al carrito.
 - Añadir un brick al carrito: Se utiliza la colección `shoppingCart` para gestionar los bricks que un usuario va añadiendo poco a poco al carrito de compras.
 - Remover un brick que ya está en el carrito de compras.



- Dependiendo de la acción que vaya a realizar, actualiza los campos correspondientes en cada una de las colecciones que sean necesarias de actualizar.
- termsAndConditions.js → Endpoint que solo va a ser empleado si el usuario rechaza los términos y condiciones. De hacerlo, el endpoint será el responsable de actualizar los campos onSaleProcess de la colección brickList, restableciéndolos para que otro usuario pueda adquirir esos bricks; y vacía la colección shoppingCart.
- checkout.js → Este endpoint finaliza una operación de compra de bricks dentro del sistema. Actualizará stocks, calculará totales a pagar, prepara un resumen del carrito de compras y genera un reporte de compra que será guardado en la colección purchaseList una vez finalizada la ejecución.

Frontend:



El Frontend no es requisito indispensable para este proyecto, se comenzó a trabajar un poco con el Frontend, pero no alcanzó el tiempo para hacer toda la funcionalidad del carrito de compras.

El Frontend está construido en ExpressJS, se creó el proyecto de Express desde cero, se instalaron los paquetes necesarios de NodeJS para poder hacer el desarrollo de éste.

El directorio del proyecto contiene múltiples carpetas y archivos; pero la más importante es views, que a su vez puede contener dentro una o múltiples carpetas; para este caso hay una carpeta que lleva por nombre app debido a que únicamente realicé el index de la aplicación; pero

normalmente en un proyecto completo existe al menos una carpeta para cada vista de proyecto. Dentro de esta carpeta se almacenan archivos .ejs que contendrán las vistas necesarias para la visualización de los elementos del proyecto, pero que estén relacionadas con el contexto de la carpeta en la que están almacenados. En el caso de la carpeta app, ésta contiene solo un archivo index.ejs.

Otros archivos importantes para el desarrollo del proyecto son:

- index.js → Este archivo es importante porque es el que contiene los parámetros iniciales para nuestra aplicación. Algunos de los elementos destacables que contiene son los siguientes:
 - Se inicializan las librerías necesarias.
 - Se especifica el archivo de rutas necesario para el funcionamiento de los procesos de enrutamiento.
 - Se establecen los parámetros iniciales para el funcionamiento de la aplicación, tales como el puerto a utilizar, el nombre de la aplicación,

el directorio de las vistas, directorios de archivos CSS y JS; el motor de vistas, un listener para verificar que el puerto esté funcionando, entre otros.

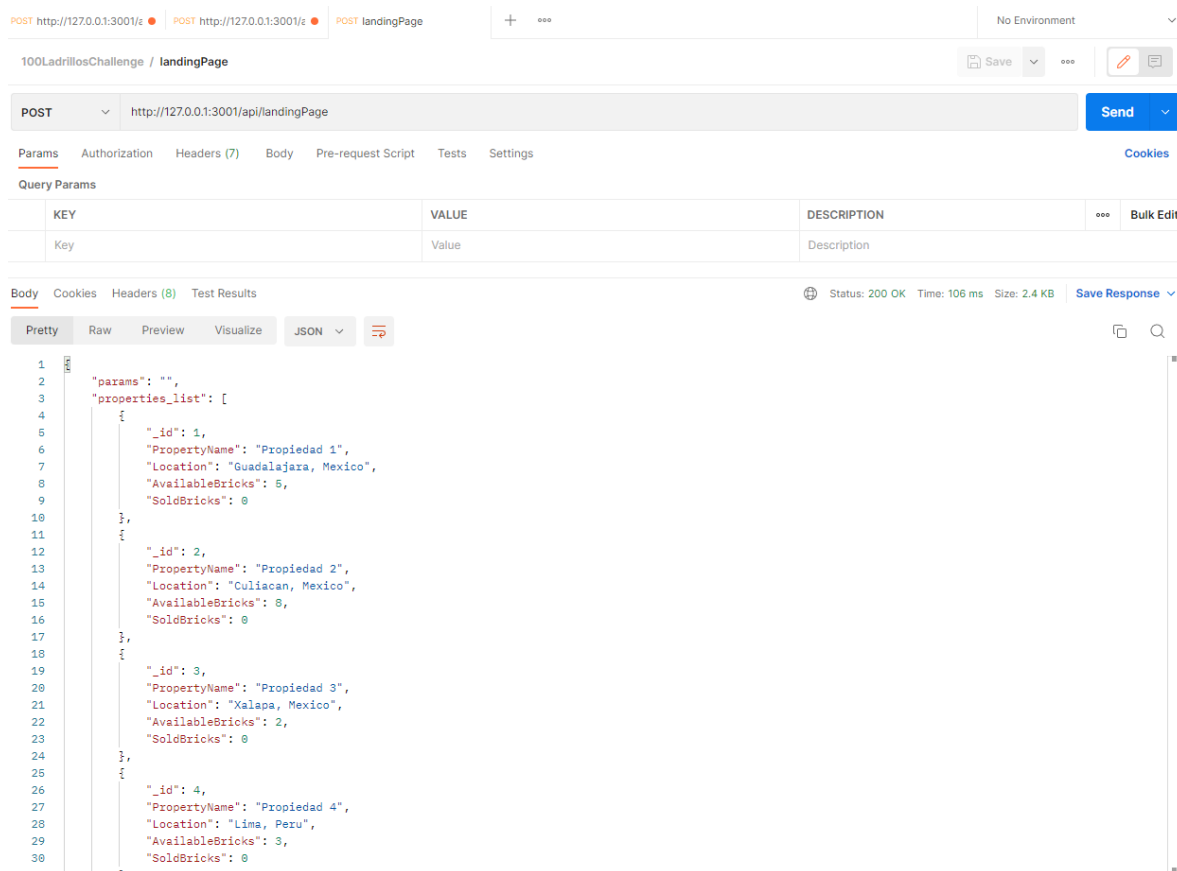
- routes.js → Este archivo es muy importante porque es el que deberá de contener todas las rutas del proyecto, tanto a archivos CSS y JS como a las vistas en sí. Se especifican las URL's que deberán de utilizarse y se coloca la respuesta que deberán de tener. También se emplea axios para realizar la comunicación con la API de NextJS (Backend), que nos brinda la capacidad de realizar el intercambio necesario de los datos para poder realizar la proyección de los datos requerida dentro del Frontend.

Resultados Obtenidos:

Los resultados fueron obtenidos por medio de Postman, que nos permitió hacer peticiones a la API con o sin envío de parámetros para obtener un resultado determinado, dependiendo de lo que se desee realizar.

Los resultados por endpoint se muestran a continuación:

landingPage:



The screenshot shows a Postman interface with a POST request to `http://127.0.0.1:3001/api/landingPage`. The response status is 200 OK, with a time of 106 ms and a size of 2.4 KB. The response body is a JSON object with the following structure:

```
1  {
2    "params": "",
3    "properties_list": [
4      {
5        "_id": 1,
6        "PropertyName": "Propiedad 1",
7        "Location": "Guadalajara, Mexico",
8        "AvailableBricks": 5,
9        "SoldBricks": 0
10     },
11     {
12       "_id": 2,
13       "PropertyName": "Propiedad 2",
14       "Location": "Culiacan, Mexico",
15       "AvailableBricks": 8,
16       "SoldBricks": 0
17     },
18     {
19       "_id": 3,
20       "PropertyName": "Propiedad 3",
21       "Location": "Xalapa, Mexico",
22       "AvailableBricks": 2,
23       "SoldBricks": 0
24     },
25     {
26       "_id": 4,
27       "PropertyName": "Propiedad 4",
28       "Location": "Lima, Peru",
29       "AvailableBricks": 3,
30       "SoldBricks": 0
31     }
32   ]
33 }
```

Como dato, solo se hizo el Frontend de la pantalla de inicio, quedando dos tablas que muestran las propiedades y los bricks disponibles, respectivamente.

127.0.0.1:3000/app

100 Ladrillos Challenge

Lista de Propiedades

#	Nombre	Ubicación	Bricks Disponibles
1	Propiedad 1	Guadalajara, Mexico	5
2	Propiedad 2	Culiacan, Mexico	8
3	Propiedad 3	Xalapa, Mexico	2
4	Propiedad 4	Lima, Peru	3
5	Propiedad 5	Santiago, Chile	1

Lista de Bricks Disponibles

Nombre	Propiedad	Precio unitario	Descripción	Bricks Disponibles	Agregar al Carrito
GDL1	1	1200		2	<button>Agregar al Carrito</button>
GDL2	1	3200		2	<button>Agregar al Carrito</button>
GDL3	1	4500		1	<button>Agregar al Carrito</button>
CUL1	2	890		2	<button>Agregar al Carrito</button>
CUL2	2	1400		2	<button>Agregar al Carrito</button>
CUL3	2	2450		3	<button>Agregar al Carrito</button>
CUL4	2	320		1	<button>Agregar al Carrito</button>
XLP1	3	1600		1	<button>Agregar al Carrito</button>

addToCart:

Antes de la ejecución:

POST http://127.0.0.1:3001/ε POST http://127.0.0.1:3001/ε POST landingPage 100LadrillosChallenge POST addToCart No Environment

100LadrillosChallenge / addToCart

POST http://127.0.0.1:3001/api/addToCart?FUNCTION=1&ID=6&BRICK_NAME=CUL3

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> FUNCTION	1	
<input checked="" type="checkbox"/> ID	6	
<input checked="" type="checkbox"/> BRICK_NAME	CUL3	
Key	Value	Description

Response

Generó la siguiente respuesta en la consola de Postman:

Body Cookies Headers (7) Test Results Status: 200 OK Time: 185 ms Size: 294 B Save Response

Pretty Raw Preview Visualize JSON

```

1  {
2    "success": "Brick añadido al carrito satisfactoriamente!!!"
3  }

```

Ahora comparemos lo que almacenó en la base de datos:

```
1 db.getCollection("shoppingCart").find({})
2
```

Raw Shell Output Find Query (line 1) ×

Documents 1 to 1

shoppingCart > BrickName

_id	BrickName	Property	Price	Quantity
6	CUL3	2	2450	1

Se almacenó el elemento dentro de la base de datos, por lo que podemos comprobar que está correcto el planteamiento hecho hasta ahora.

Ahora añadiremos otra vez el mismo brick:

```
1 db.getCollection("shoppingCart").find({})
2
```

Raw Shell Output Find Query (line 1) ×

Documents 1 to 1

shoppingCart > BrickName

_id	BrickName	Property	Price	Quantity
6	CUL3	2	2450	2

Se incrementó la cantidad en 1, ya que se tenía previamente añadido el mismo brick.

Ahora añadiremos algunos otros bricks al carrito, quedando de la siguiente manera:

```
1 db.getCollection("shoppingCart").find({})
2
```

Raw Shell Output Find Query (line 1) ×

Documents 1 to 2

shoppingCart > BrickName

_id	BrickName	Property	Price	Quantity
6	CUL3	2	2450	3
11	LIM2	4	2300	1

Como se puede apreciar, se tiene un brick perteneciente a otra propiedad, por lo que se genera un nuevo elemento en el carrito de compras.

Procederemos a añadir una unidad más de este brick, ya que solo había existencias de uno, para forzar a que marque error y no deje añadirlo:

100LadrillosChallenge / addToCart

POST http://127.0.0.1:3001/api/addToCart?FUNCTION=1&ID=11&BRICK_NAME=LIM2

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION
FUNCTION	1	
ID	11	
BRICK_NAME	LIM2	
Key	Value	Description

Body Cookies Headers (7) Test Results Status: 200 OK Time: 162 ms Size: 326 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "error": "Lo sentimos, el brick que seleccionaste está agotado. Favor de seleccionar otro."
3 }
```

Como podemos ver, ya no se puede añadir más elementos de ese brick al carrito, ya que en ese momento ya se encuentra ocupado.

removeFromCart:

Ahora, partiendo de la prueba anterior, vamos a proceder a eliminar elementos del carrito.

Eliminaremos primero de un elemento del cual tenemos más de un brick añadido en nuestro carrito. Partimos con el siguiente query:

POST http://127.0.0.1:3001/api/addToCart?FUNCTION=2&ID=6&BRICK_NAME=CUL3

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION
FUNCTION	2	
ID	6	
BRICK_NAME	CUL3	
Key	Value	Description

Body Cookies Headers (7) Test Results Status: 200 OK Time: 95 ms Size: 290 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "success": "Brick retirado del carrito correctamente!!!"
3 }
```

Verificamos en la base de datos, teníamos 3 unidades de este brick, ahora solo contamos con 2 y se redujo en 1 el contador de OnSaleProcess:

localhost:27017 > challenge100Ladrillos

```
1 db.getCollection("brickList").find({})
2
```

Raw Shell Output Find Query (line 1) x

Documents 1 to 13

brickList > BrickName

_id	BrickName	Property	Price	AvailableBricks	SoldBricks	Description	OnSaleProcess
1.0	GDL1	1.0	1200.0	2.0	0.0		0.0
2.0	GDL2	1.0	3200.0	2.0	0.0		0.0
3.0	GDL3	1.0	4500.0	1.0	0.0		0.0
4.0	CUL1	2.0	890.0	2.0	0.0		0.0
5.0	CUL2	2.0	1400.0	2.0	0.0		0.0
6.0	CUL3	2.0	2450.0	3.0	0.0		2.0
7.0	CUL4	2.0	320.0	1.0	0.0		0.0
8.0	XLP1	3.0	1600.0	1.0	0.0		0.0
9.0	XLP2	3.0	230.0	1.0	0.0		0.0
10.0	LIM1	4.0	4500.0	1.0	0.0		0.0
11.0	LIM2	4.0	2300.0	1.0	0.0		1.0
12.0	LIM3	4.0	1320.0	1.0	0.0		0.0
13.0	SCL1	5.0	6000.0	1.0	0.0		0.0

localhost:27017 > challenge100Ladrillos

```
1 db.getCollection("shoppingCart").find({})
2
```

Raw Shell Output Find Query (line 1) x

Documents 1 to 2

shoppingCart > Quantity

_id	BrickName	Property	Price	Quantity
6	CUL3	2	2450	2
11	LIM2	4	2300	1

Ahora, eliminaremos del carrito el elemento con _id=11 de nombre LIM2, ya que solo tenemos una unidad éste deberá eliminarlo completamente de la base de datos:

localhost:27017 > challenge100Ladrillos

```
1 db.getCollection("shoppingCart").find({})
2
```

Raw Shell Output Find Query (line 1) x

50 Documents 1 to 1

shoppingCart > BrickName

_id	BrickName	Property	Price	Quantity
6	CUL3	2	2450	2

localhost:27017 > challenge100Ladrillos

```
1 db.getCollection("brickList").find({})
2
```

Raw Shell Output Find Query (line 1) x

50 Documents 1 to 13

brickList > BrickName

_id	BrickName	Property	Price	AvailableBricks	SoldBricks	Description	OnSaleProcess
1.0	GDL1	1.0	1200.0	2.0	0.0		0.0
2.0	GDL2	1.0	3200.0	2.0	0.0		0.0
3.0	GDL3	1.0	4500.0	1.0	0.0		0.0
4.0	CUL1	2.0	890.0	2.0	0.0		0.0
5.0	CUL2	2.0	1400.0	2.0	0.0		0.0
6.0	CUL3	2.0	2450.0	3.0	0.0		2.0
7.0	CUL4	2.0	320.0	1.0	0.0		0.0
8.0	XLP1	3.0	1600.0	1.0	0.0		0.0
9.0	XLP2	3.0	230.0	1.0	0.0		0.0
10.0	LIM1	4.0	4500.0	1.0	0.0		0.0
11.0	LIM2	4.0	2300.0	1.0	0.0		0.0
12.0	LIM3	4.0	1320.0	1.0	0.0		0.0
13.0	SCL1	5.0	6000.0	1.0	0.0		0.0

Como se puede ver, solo quedó un registro, por lo que el requisito se cumplió satisfactoriamente.

termsAndConditions:

Este endpoint solo se puede probar en caso de que el usuario no acepte los términos y condiciones al momento de querer realizar la compra de sus bricks. De no aceptarlos este endpoint vaciará el carrito y restablecerá el campo OnSaleProcess.

Partiremos de la ejecución del endpoint anterior, por lo que los valores en la BD se mantienen iguales.

Se ejecuta el endpoint:

100LadrillosChallenge / termsAndConditions

POST http://127.0.0.1:3001/api/termsAndConditions

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (7) Test Results Status: 200 OK Time: 147 ms Size: 273 B Save Response

Pretty Raw Preview Visualize JSON

```

1  "success": "Carrito de compras vacio."
2
3

```

La base de datos quedó así:

localhost:27017 > challenge100Ladrillos

db.getCollection("brickList").find({})

Raw Shell Output Find Query (line 1) x

50 Documents 1 to 13

_id	BrickName	Property	Price	AvailableBricks	SoldBricks	Description	OnSaleProcess
1.0	GDL1	1.0	1200.0	2.0	0.0		0.0
2.0	GDL2	1.0	3200.0	2.0	0.0		0.0
3.0	GDL3	1.0	4500.0	1.0	0.0		0.0
4.0	CUL1	2.0	890.0	2.0	0.0		0.0
5.0	CUL2	2.0	1400.0	2.0	0.0		0.0
6.0	CUL3	2.0	2450.0	3.0	0.0		0.0
7.0	CUL4	2.0	320.0	1.0	0.0		0.0
8.0	XLP1	3.0	1600.0	1.0	0.0		0.0
9.0	XLP2	3.0	230.0	1.0	0.0		0.0
10.0	LIM1	4.0	4500.0	1.0	0.0		0.0
11.0	LIM2	4.0	2300.0	1.0	0.0		0.0
12.0	LIM3	4.0	1320.0	1.0	0.0		0.0
13.0	SCL1	5.0	6000.0	1.0	0.0		0.0

Cero

localhost:27017 > challenge100Ladrillos

db.getCollection("shoppingCart").find({})

Raw Shell Output Find Query (line 1) x

50 Documents -- to --

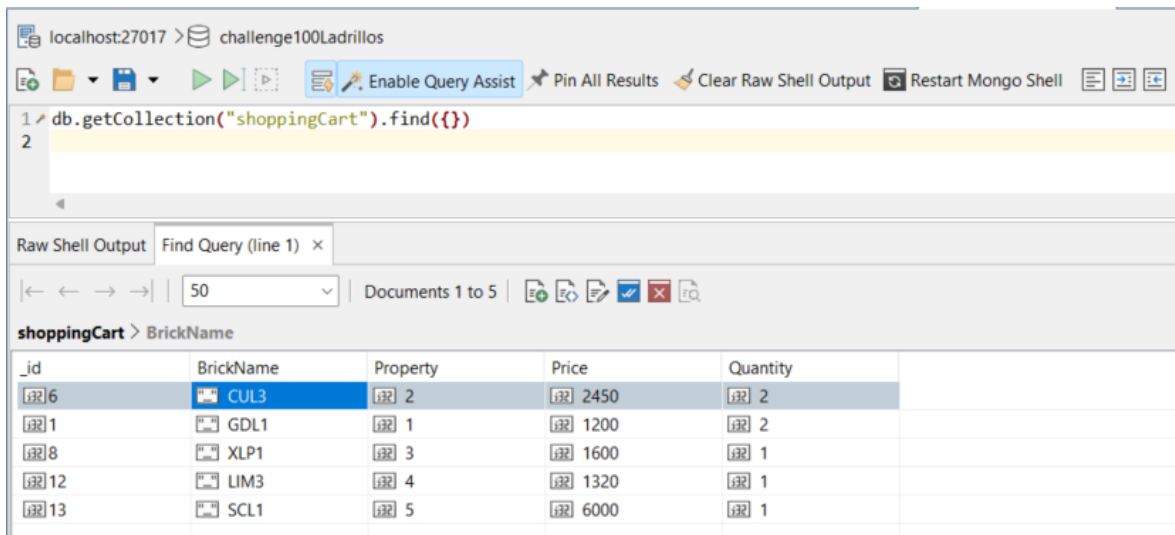
shoppingCart

_id

Como podemos ver, se restablecieron los valores de OnSaleProcess correspondientes a todos los elementos de la colección shoppingCart, la cual quedó vacía.

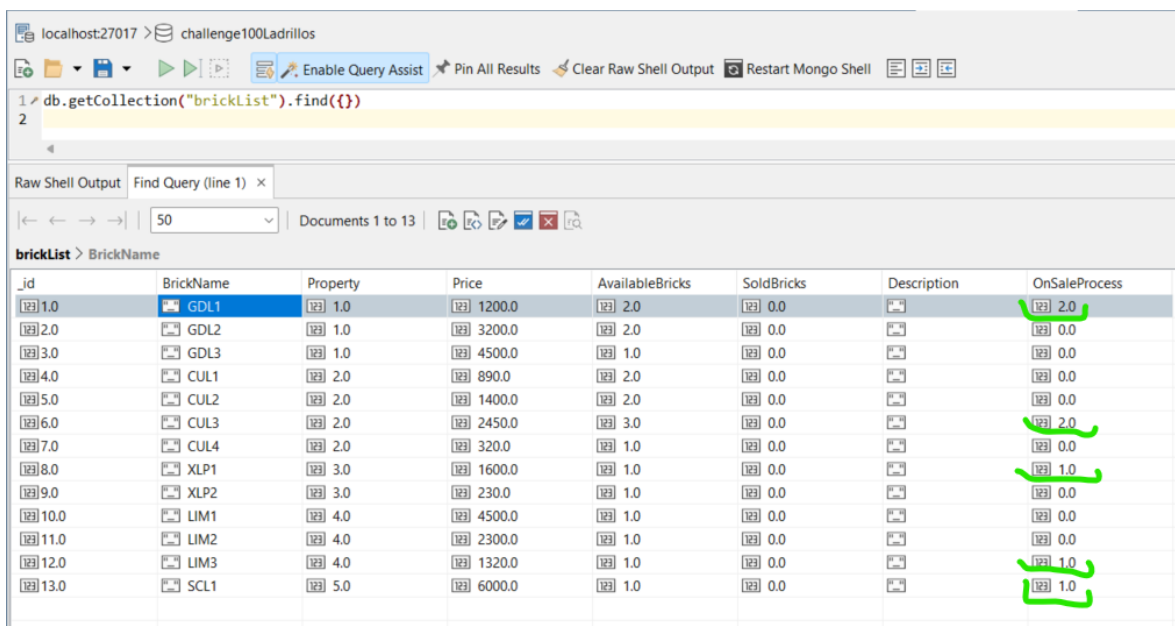
checkout:

Para probar este endpoint se volvió a llenar el carrito de compras con diversos bricks y de esta manera concretar una venta.



_id	BrickName	Property	Price	Quantity
6	CUL3	2	2450	2
1	GDL1	1	1200	2
8	XLP1	3	1600	1
12	LIM3	4	1320	1
13	SCL1	5	6000	1

Solo con fines comparativos muestro la colección de brickList para verificar que los bricks que añadimos al carrito estén en proceso de venta:



_id	BrickName	Property	Price	AvailableBricks	SoldBricks	Description	OnSaleProcess
1.0	GDL1	1.0	1200.0	2.0	0.0		2.0
2.0	GDL2	1.0	3200.0	2.0	0.0		0.0
3.0	GDL3	1.0	4500.0	1.0	0.0		0.0
4.0	CUL1	2.0	890.0	2.0	0.0		0.0
5.0	CUL2	2.0	1400.0	2.0	0.0		0.0
6.0	CUL3	2.0	2450.0	3.0	0.0		2.0
7.0	CUL4	2.0	320.0	1.0	0.0		0.0
8.0	XLP1	3.0	1600.0	1.0	0.0		1.0
9.0	XLP2	3.0	230.0	1.0	0.0		0.0
10.0	LIM1	4.0	4500.0	1.0	0.0		0.0
11.0	LIM2	4.0	2300.0	1.0	0.0		0.0
12.0	LIM3	4.0	1320.0	1.0	0.0		1.0
13.0	SCL1	5.0	6000.0	1.0	0.0		1.0

Ahora podemos ejecutar la función 1 del endpoint de checkpoint, que es la que nos generará un reporte de ventas que nos va a generar un pequeño resumen con el fin de poderlo mostrar en el Frontend, la ejecución en Postman queda de la siguiente manera:

POST http://127.0.0.1:3001/api/checkout?FUNCTION=1

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> FUNCTION	1	
Key	Value	Description

Body Cookies Headers (7) Test Results Status: 200 OK Time: 33 ms Size: 692 B Save Response

Pretty Raw Preview Visualize JSON

```

15      "Quantity": 2
16    },
17    {
18      "_id": 8,
19      "BrickName": "XLP1",
20      "Property": 3,
21      "Price": 1600,
22      "Quantity": 1
23    },
24    {
25      "_id": 12,
26      "BrickName": "LM3",
27      "Property": 4,
28      "Price": 1320,
29      "Quantity": 1
30    },
31    {
32      "_id": 13,
33      "BrickName": "SCL1",
34      "Property": 5,
35      "Price": 6000,
36      "Quantity": 1
37    }
38  ],
39  "subtotal": "13624.80",
40  "iva": "2595.20",
41  "total": 16220,
42  "success": "Información desplegada con éxito"
43

```

El JSON generado se puede utilizar sin ningún problema en el Frontend que se fuera a desarrollar para esta aplicación.

Finalmente, probaremos la segunda función, que corresponderá a finalizar la compra. Aquí lo que va a hacer es actualizar stocks en la colección de brickList, se almacenará un reporte en la colección purchaseList y se vaciará la colección shoppingCart.

Para fines ilustrativos, actualmente no hay ningún reporte en la colección purchaseList:

localhost:27017 > challenge100Ladrillos

Enable Query Assist Pin All Results Clear Raw Shell Output Restart Mongo Shell

```

1 db.getCollection("purchaseList").find({})
2

```

Raw Shell Output Find Query (line 1) x

50 Documents -- to --

purchaseList

_id

Ahora ejecutamos la función correspondiente. Estos son los cambios que surgieron una vez ejecutado:

The screenshot shows a REST client interface with a POST request to `http://127.0.0.1:3001/api/checkout?FUNCTION=2`. The response status is 200 OK. The response body is a JSON object: `{ "success": "Compra finalizada con éxito" }`.

Colección brickList:

The screenshot shows a MongoDB Shell interface with the command `db.getCollection("brickList").find({})`. The results show a list of 13 documents in the `brickList` collection. The columns are: `_id`, `BrickName`, `Property`, `Price`, `AvailableBricks`, `SoldBricks`, `Description`, and `OnSaleProcess`.

_id	BrickName	Property	Price	AvailableBricks	SoldBricks	Description	OnSaleProcess
1.0	GDL1	1.0	1200.0	0.0	2.0		0.0
2.0	GDL2	1.0	3200.0	2.0	0.0		0.0
3.0	GDL3	1.0	4500.0	1.0	0.0		0.0
4.0	CUL1	2.0	890.0	2.0	0.0		0.0
5.0	CUL2	2.0	1400.0	2.0	0.0		0.0
6.0	CUL3	2.0	2450.0	1.0	2.0		0.0
7.0	CUL4	2.0	320.0	1.0	0.0		0.0
8.0	XLP1	3.0	1600.0	0.0	1.0		0.0
9.0	XLP2	3.0	230.0	1.0	0.0		0.0
10.0	LIM1	4.0	4500.0	1.0	0.0		0.0
11.0	LIM2	4.0	2300.0	1.0	0.0		0.0
12.0	LIM3	4.0	1320.0	0.0	1.0		0.0
13.0	SCL1	5.0	6000.0	0.0	1.0		0.0

Colección shoppingCart, la cual queda vacía una vez finalizada la venta:

The screenshot shows a MongoDB Shell interface with the command `db.getCollection("shoppingCart").find({})`. The results show an empty list in the `shoppingCart` collection.

Colección purchaseList, la cual almacena los reportes de venta. Se muestra en formato JSON para que todos los componentes sean visibles:

```
localhost:27017 > challenge100Ladrillos

1 db.getCollection("purchaseList").find({})

Raw Shell Output Find Query (line 1) x

1 50 Documents 1 to 1

1 {
2   "_id" : ObjectId("631dcf0ae320a26788f27249"),
3   "Date" : "Sun Sep 11 2022 07:05:30 GMT-0500 (hora de verano central)",
4   "Hour" : "7",
5   "BricksSold" : [
6     {
7       "ID" : NumberInt(6),
8       "BrickName" : "CUL3",
9       "Quantity" : NumberInt(2),
10      "UnitPrice" : NumberInt(2450),
11      "Total" : NumberInt(4900)
12    },
13    {
14      "ID" : NumberInt(1),
15      "BrickName" : "GDL1",
16      "Quantity" : NumberInt(2),
17      "UnitPrice" : NumberInt(1200),
18      "Total" : NumberInt(2400)
19    },
20    {
21      "ID" : NumberInt(8),
22      "BrickName" : "XLP1",
23      "Quantity" : NumberInt(1),
24      "UnitPrice" : NumberInt(1600),
25      "Total" : NumberInt(1600)
26    },
27    {
28      "ID" : NumberInt(12),
29      "BrickName" : "LIM3",
30      "Quantity" : NumberInt(1),
31      "UnitPrice" : NumberInt(1320),
32      "Total" : NumberInt(1320)
33    },
34    {
35      "ID" : NumberInt(13),
36      "BrickName" : "SCL1",
37      "Quantity" : NumberInt(1),
38      "UnitPrice" : NumberInt(6000),
39      "Total" : NumberInt(6000)
40    }
41  ],
42   "Subtotal" : "13624.80",
43   "IVA" : "2595.20",
44   "Total" : NumberInt(16220)
45 }
46
```

Con esto quedan finalizadas las pruebas del proyecto “100 Ladrillos Challenge”.

¿Qué mejoras se pueden añadir en una versión posterior?

- Finalizar el desarrollo del Frontend, falta construir el resto de las vistas necesarias para completar el carrito de compras.
- Incorporar a la vista inicial del Frontend una opción para poder actualizar sin recargar la página la información del carrito de compras y los stocks.
- Agregar más información y datos necesarios sobre los bricks a las bases de datos con el fin de hacer un sistema más robusto.
- Añadir un login (Inicio de sesión y Contraseña) al sistema. Esto ayudará también a incorporar la posibilidad de tener múltiples carritos de compra en el sistema.
- Buscar la forma de que el sistema sea completamente seguro y con protección de datos personales.
- Incorporar la funcionalidad referente a un servicio de pagos en línea propio que tenga soporte para tarjetas VISA, MasterCard o American Express; o un sistema de terceros como Mercado Pago o PayPal.
- Entre otras...