Advanced Databases

Laboratory #3

An heterogenous DB system

UNIVERSITAT DE BARCELONA

Diciembre 2021

Arthur Font Gouveia – Niub: 20222613

Eduard Vives Isern – Niub: 20206970

Tabal de contenidos

Introducción a la realización de la Práctica 3

Para este laboratorio se nos ha sido asignada la base de datos tipo **Key-value stores**.

Para crear una aplicación sencilla que utilice tanto una base de datos relacional como una no relacional, el DBMS que hemos elegido para este tipo de BD NoSQL, tras buscar entre los más populares, ha sido Redis.

Para conectarnos a la base de datos de Redis hemos utilizado “Node.js”. Este es un entorno de tiempo de ejecución de JavaScript (de ahí su terminación en .js), mediante el cual nos hemos conectado al DBMS desde el código.

Paralelamente, hemos creado una BD relacional utilizando MySQL, y también nos hemos conectado a esta BD relacional desde el código.

Después de investigar los detalles, característica y usos de las bases de datos Key-value stores, nos hemos decidido a diseñar e implementar un componente de una aplicación sencilla de carrito de compras con NodeJS usando Redis y MySQL.

Nos hemos decidido por una aplicación con esta funcionalidad, debido a que es uno de los casos de uso más frecuentes y eficientes de las bases de datos Key-value.

Si tomamos en consideración las ventajas de este tipo de bases de datos, uno de los casos de uso más populares al que se presta naturalmente, entre otros, es el almacenamiento de los detalles de la sesión del usuario y sus preferencias en aplicaciones web, en un almacén de valores clave. Toda la información es accesible a través de la clave del usuario, y los almacenes de valor-clave se prestan a lecturas y escrituras rápidas.

Para ello en este trabajo aprovechando el poder de Redis hemos creado una aplicación básica de carrito de compras de comercio electrónico con Node.js. Por lo general, los datos del carrito de compras se almacenan en el lado del cliente como una cookie. Las cookies son pequeños archivos de texto almacenados en el directorio del navegador de un usuario web o en la carpeta de datos. La ventaja de hacer esto es que no necesitaría almacenar tales datos temporales en su base de datos. Esto requerirá que envíe las cookies con cada solicitud web.

De este modo, como ya hemos comentado, almacenando los datos del carrito de la compra en Redis es una buena idea, ya que puede recuperar los artículos muy rápido en cualquier momento y conservar estos datos si es necesario.

Para poder completar este objetivo, hemos tomado como código base una publicación pública proporcionada por el propio Redis, en la cual nos proporcionan un frontend básico para la tienda y una primera implementación de la base de datos

de Redis. Esto nos permite gestionar múltiples clientes a la vez, cada uno con su carrito.

En el siguiente link puede encontrar dicha información:
https://developer.redis.com/howtos/shoppingcart/

Al almacenar los datos del carrito de compras en el lado del cliente como una cookie, si quieren hacerse pruebas en un mismo ordenador, cada cliente estará asignado a un buscador distinto (Chrome, Firefix…).

BD Relacional MySQL

Esta aplicación aparte de la BD NoSQL, el sistema de BD también utiliza una BD Relacional. Hemos intentado tener en cuenta todos los detalles como si se tratara de una aplicación real utilizada por una empresa. Esto se observará en la creación de la base de datos de MySQL (tablas, cardinalidades entre las relaciones, tipo asignado a cada atributo, atributos que son primary key, cuáles no pueden ser nullos, cuáles son únicos…). Por otro lado, también influye en las operaciones CRUD que hemos decidido implementar y en cómo hemos decidido implementarlas. Un ejemplo es la operación referente a la eliminación de un producto de la base de datos de MySQL, el cual comentaremos a continuación en el apartado de las CRUD OP y la relación entre las dos BD.

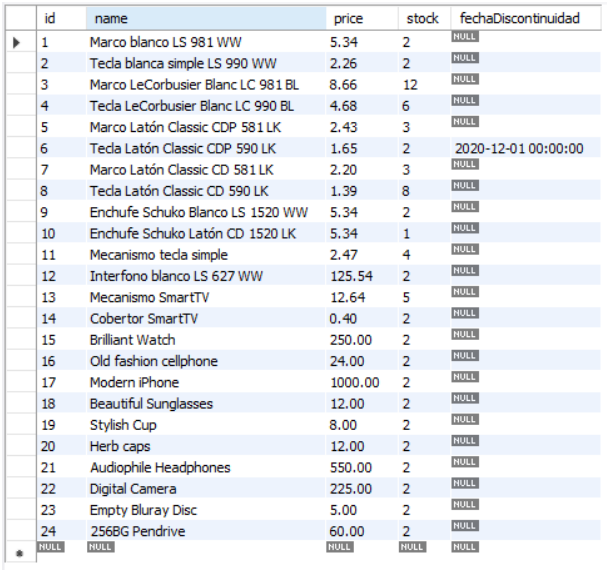
En MySQL hemos hecho una base de datos para una empresa ficticia que comercializa principalmente material eléctrico, aunque también hemos añadido que dispone de productos de carácter general, para poder dar más juego a la visualización del frontend de la página y a las posibilidades de testear las funciones CRUD de insertar, eliminar y modificar.

Esta empresa tiene varios tipos de productos que pueden encargar los clientes a través de un comercial. También se podrán hacer búsquedas del material, gestionar pedidos y entregas de los productos, realizando encargos a fábrica si fuera necesario…

Centrándonos en los productos, que es la parte más relevante de este trabajo, la entidad producto nos describe las características mínimas que tiene un producto de la empresa. Estos son una ID, un nombre, un precio, un stock y una fecha de discontinuidad.

La fecha de discontinuidad indica el momento en que el producto deja de estar en producción, por tanto, puede no estar presente (campo opcional) y esto significaría que el producto sigue en producción. Hemos añadido este último atributo para poder controlar productos que se han discontinuado sin eliminarlos de la Base de Datos, ya que, aunque no lo estemos produciendo, tenemos un histórico de encargos

relativos a ese producto. Hemos decidido hacerlo de esta forma, aparte de para no perder los históricos una vez se dejen de fabricar, también porque en las Bases de Datos es poco aconsejable hacer DELETE, es decir, eliminaciones de campos.



id	name	price	stock	fechaDiscontinuidad
1	Marco blanco LS 981 WW	5.34	2	NULL
2	Tecla blanca simple LS 990 WW	2.26	2	NULL
3	Marco LeCorbusier Blanc LC 981 BL	8.66	12	NULL
4	Tecla LeCorbusier Blanc LC 990 BL	4.68	6	NULL
5	Marco Latón Classic CDP 581 LK	2.43	3	NULL
6	Tecla Latón Classic CDP 590 LK	1.65	2	2020-12-01 00:00:00
7	Marco Latón Classic CD 581 LK	2.20	3	NULL
8	Tecla Latón Classic CD 590 LK	1.39	8	NULL
9	Enchufe Schuko Blanco LS 1520 WW	5.34	2	NULL
10	Enchufe Schuko Latón CD 1520 LK	5.34	1	NULL
11	Mecanismo tecla simple	2.47	4	NULL
12	Interfono blanco LS 627 WW	125.54	2	NULL
13	Mecanismo SmartTV	12.64	5	NULL
14	Cobertor SmartTV	0.40	2	NULL
15	Brilliant Watch	250.00	2	NULL
16	Old fashion cellphone	24.00	2	NULL
17	Modern iPhone	1000.00	2	NULL
18	Beautiful Sunglasses	12.00	2	NULL
19	Stylish Cup	8.00	2	NULL
20	Herb caps	12.00	2	NULL
21	Audiophile Headphones	550.00	2	NULL
22	Digital Camera	225.00	2	NULL
23	Empty Bluray Disc	5.00	2	NULL
24	256BG Pendrive	60.00	2	NULL
NULL	NULL	NULL	NULL	NULL

Ejemplo de datos de los productos en la base de datos de MySQL. Visualizado mediante el software MySQL Workbench.

** Hemos poblado todas las tablas con datos adecuados, como para poder caracterizar el tema propuesto.*

Diagrama E-R

Junto a los archivos entregados hemos adjuntado también la siguiente imagen, para poder disponer de ella con mayor resolución.

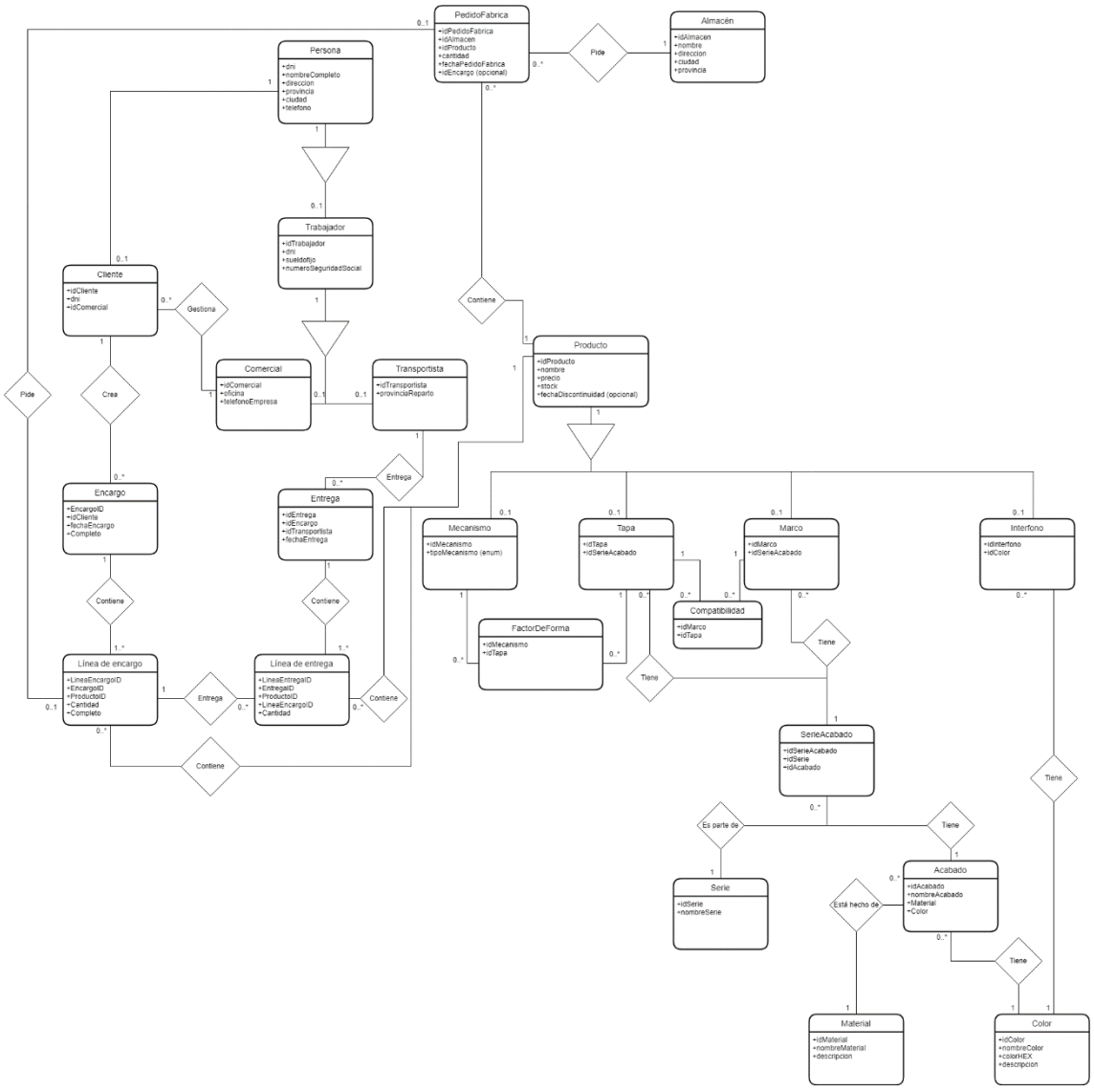


Diagrama de Entidad-Relación

Diagrama en workbench

Junto a los archivos entregados hemos adjuntado también la siguiente imagen, para poder disponer de ella con mayor resolución.

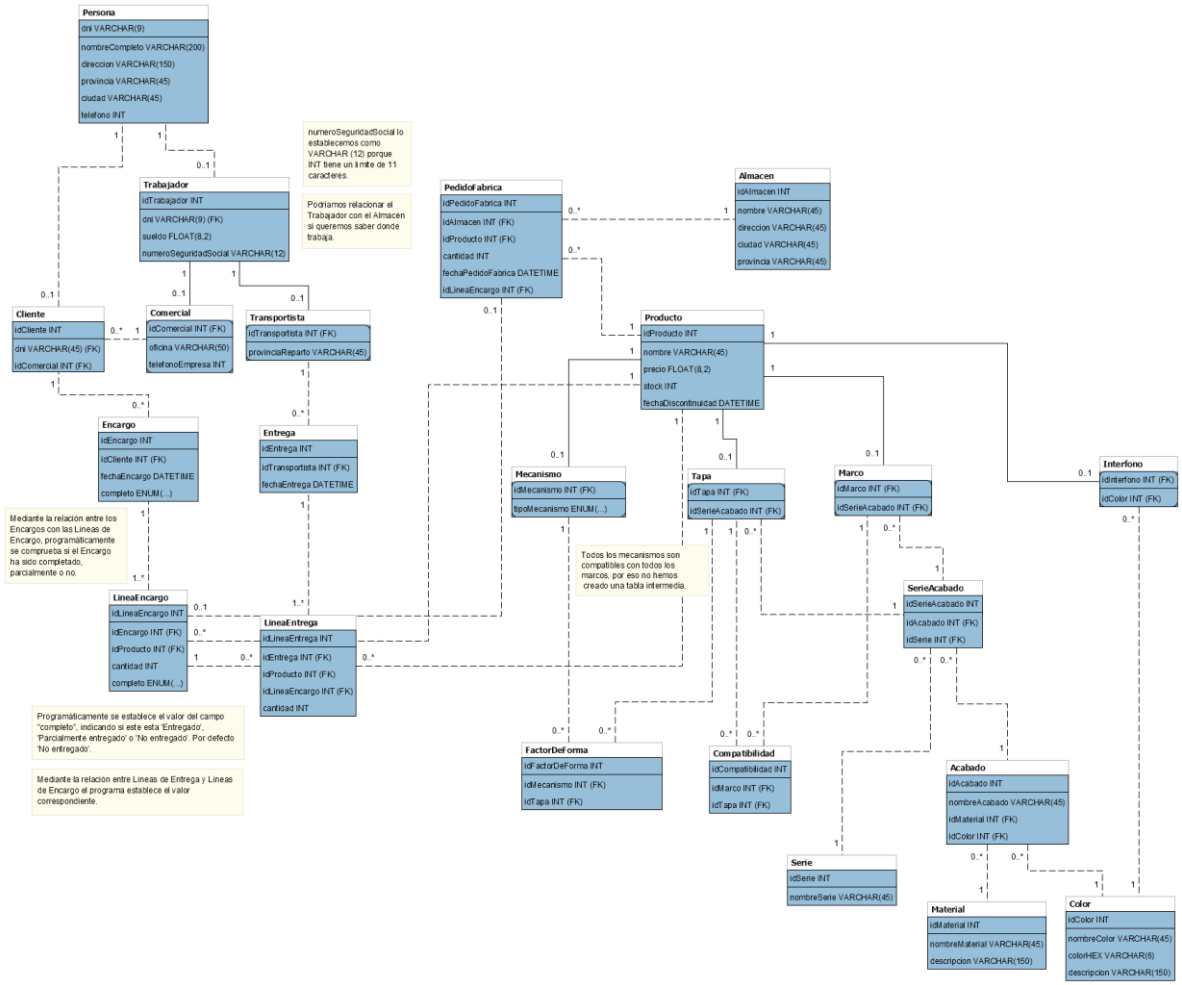


Diagrama de Entidad-Relación

* Todas las tablas que lo requieren tienen id única (no compuesta).
* Todas las relaciones de herencia o Foreign Key tienen nombres únicos.

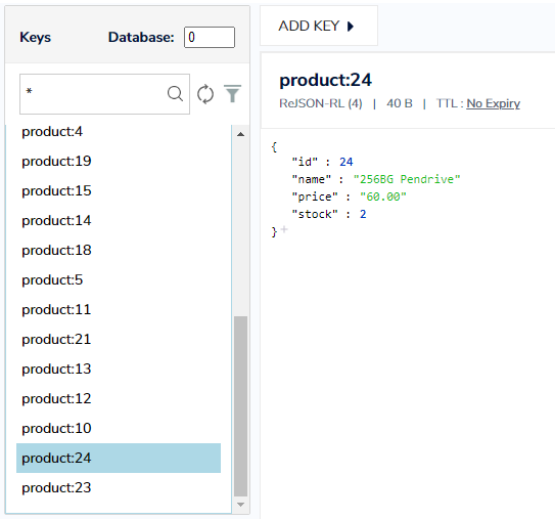
Relaciones entre las dos BD

Para establecer relaciones entre los datos de ambas BD, utilizamos como atributo clave la id de los productos.

En MySQL guardamos la información sobre la gestión de la empresa y todos los productos, sus características, de que tipo son, su compatibilidad con otros productos, material, color…

A diferencia de MySQL, en Redis solo mantenemos las características básicas e imprescindibles para poder gestionar los productos y mostrarlos en el carrito de la compra de un usuario. Estas son id, nombre, precio y stock.

Guardando esa información sobre el producto en Redis, tenemos un rápido acceso a ella pudiéndola mostrar en el carrito del usuario.



Ejemplo de estructura y datos de un producto en Redis.
Visualizado mediante el software RedisInsight.

La otra relación que debe cumplirse entre las dos BD y que gestionamos en las CRUD Op, es referente al stock de cada producto. En pocas palabras, el stock completo de un producto en MySQL debe coincidir con el valor del stock de ese producto en el catálogo de Redis más las unidades que haya de ese producto en los carritos de los usuarios.

CRUD Operations

Hemos implementado las operaciones CRUD de inserción, eliminación y actualización. Las tres operaciones afectan tanto a la BD NoSQL como a la BD relacional al mismo tiempo y, por tanto, todas las operaciones son atómicas, es decir, una modificación que pueda afectar a ambos sistemas de BD debe llevarse a cabo por completo, o no llevarse a cabo.

Las situaciones que hemos considerado para decidir que operaciones CRUD implementar son las siguientes (numeradas para poder referirnos a ellas más fácilmente en el contenido del documento):

1. **DELETE**: Si eliminas un producto, se debe eliminar en la base de datos de MySQL y en el catálogo de productos de Redis, eliminándolo también de todos los carritos en los que estuviera.

   Un comentario importante sobre esta operación que modifica las dos BD, es que, aunque podría tratarse como un **DELETE** en las dos BD, en realidad se trata de un **UPDATE + DELETE**. Esto se debe a que, como hemos dicho antes, no eliminamos los productos de MySQL, sino que los discontinuamos añadiendo un valor al campo fecha de discontinuidad. Esta acción en MySQL actúa como si eliminásemos el producto, y en Redis nunca dispondremos de productos que en MySQL tengan un valor en el campo de fecha de discontinuidad, ya que estos están considerados como eliminados.

2. **UPDATE**: Si modificas el atributo de un producto (nombre, precio o stock) hay que modificarlo en la base de datos de MySQL y en el catálogo de productos de Redis, modificándolo también en todos los carritos que contengan ese producto.

3. **INSERT**: Si añades un producto, se debe añadir el producto en la BD de MySQL y en el catálogo de productos de Redis.

Por otro lado, también tenemos operaciones CRUD que afectan solo a la base de datos de Redis:

4. **DELETE**: Podemos eliminar cualquier producto de nuestro carrito de la compra, si disminuimos su cantidad hasta 0.

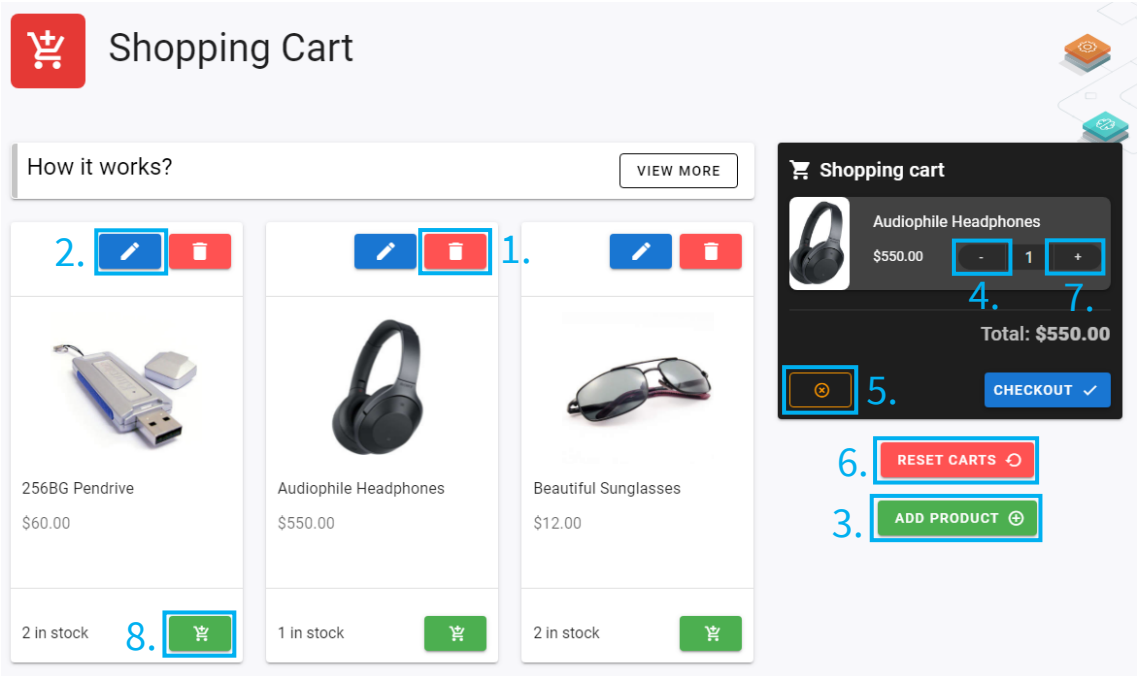
5. **DELETE + UPDATE**: Podemos vaciar nuestro carrito de la compra. Lo cual implica eliminar todos los productos que haya en el carrito y devolver ese stock al stock de cada producto en la tienda.

6. **DELETE + UPDATE**: También disponemos de la posibilidad de vaciar todos los carritos de Redis, sumando todas las unidades de cada producto en los carritos al stock de la tienda.

7. **UPDATE**: Podemos aumentar o disminuir las unidades de un producto en nuestro carrito de la compra, lo cual también modifica las unidades de ese producto en el stock de la tienda.

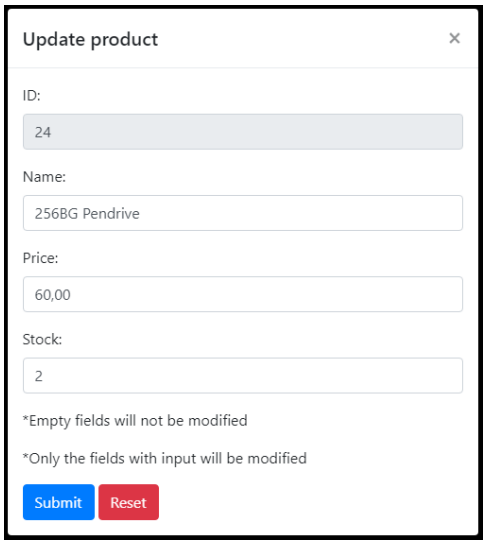
8. **INSERT**: Podemos añadir cualquier producto a nuestro carrito de la compra, siempre y cuando haya stock disponible.

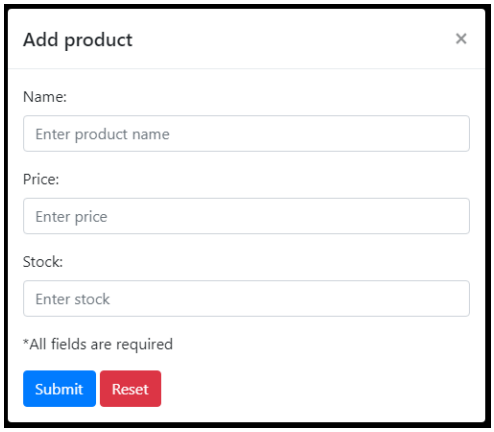
La ejecución de todas estas operaciones se vuelve mucho más sencillo mediante el uso del Frontend de la aplicación.



Las siguientes imágenes corresponden a los formularios que hemos creado mediante BootstrapVue para facilitar las operaciones de editar un producto (2) y añadir un producto a la base de datos (3).

Debajo de cada formulario se encuentran comentarios importantes sobre los campos requeridos y como serán modificados los atributos según los inputs.
En el caso del formulario para editar un producto, no permitimos desde backend ni frontend modificar la id de este.





Cambios en el proyecto base

Respecto al proyecto base proporcionado por Redis hemos tenido que aplicar numerosos cambios. Antes de nada hemos tenido que comprender la implementación y funcionamiento del proyecto, así como el flujo de ejecución y relación entre el cliente y el servidor, incluyendo los Controllers que gestionan cada una de las CRUD Operations del Servidor.

En el proyecto base los productos del catálogo de Redis estaban fijados y eran obtenidos de un archivo JSON que se encontraba en el mismo directorio del servidor. Hemos reimplementado la obtención y gestión de todos los productos de Redis (CRUD Op…) para trabajar con los productos de la base de datos de MySQL.

Eso ha sido lo más complejo, ya que hemos tenido muchas dificultades para gestionar las llamadas y esperas de las operaciones asíncronas a MySQL, para conseguir que las CRUD Op sean atómicas. Así como la gestión de los formatos de datos para que pudieran compartirse entre las dos BD.
A parte de eso, hemos creado las opciones en el frontend y backend, así como los controllers… para poder crear nuevos productos en MySQL que sean identificados y gestionados por Redis junto a los creados previamente, y la modificación y eliminación de los productos existentes en MySQL y Redis.

Aunque no es relevante, hemos corregido algunos detalles del proyecto de Redis para que sea Responsive (se adapte a cualquier resolución de PC, tablet o móvil), ya que tenía algunos errores que hemos podido detectar.

Repositorio de GitHub

Para poder gestionar todo este proyecto con más facilidad hemos creado un repositorio público en GitHub donde cada miembro del grupo trabajaba sobre su rama y los cambios comprobados se subían posteriormente a la main.

El repositorio es el siguiente: https://github.com/eduvives/basic-redis-shopping-chart-nodejs

Para mantener buenas prácticas y facilitar la compresión del código al otro compañero, el código se encuentra mínimamente comentado y cada en commit realizado se encuentra detalladamente explicados los cambios hechos.

El proyecto también cuenta con un README que puede observarse en la página principal del repositorio. Este README ha sido actualizado tras cada actualización del código que lo requiriera.

Este se encuentra divido en tres secciones:

- La primera muestra la información técnica del proyecto.
- La segunda indica cómo funciona el proyecto.
- Por último, explica los pasos a seguir para ejecutar el proyecto en local (prerrequisitos…).

Archivos entregados

Junto a este documento hemos adjuntado las imágenes del Diagrama E-R y el Diagrama en workbench. Los cuales incluyen todas las entidades necesarias para su caracterización y explotación.

También hemos entregado dos ficheros con los nombres:

1. e-commerce_company.sql
2. Esquema Workbench.mwb

El primero corresponde al fichero .sql de la base de correspondiente a nuestro esquema, con todas las tablas suficientemente pobladas con datos adecuados, como para poder caracterizar el tema propuesto.

El segundo es el esquema E-R implementado en MySQL Workbench.

La aplicación completa se encuentra en el siguiente enlace de nuestro repositorio de GitHub (Shopping Cart app in NodeJS with Redis and MySQL):

https://github.com/eduvives/basic-redis-shopping-chart-nodejs