LocalStore

Se desea desarrollar una plataforma de gestión integral para una web de ventas online en la cual participaremos dentro del equipo de 'DATA'. Este equipo será el encargado de diseñar e implementar el sistema de bases de datos que tenga la capacidad de gestionar la información de la plataforma, así como el diseño e implementación de los modelos y la interfaz de mapeo que permita la comunicación con las bases de datos. Se ha decidido dividir el proyecto en cuatro subproyectos: base de datos general, cache y sesiones, logística y finalmente análisis e informes.

El cliente ha comentado que la plataforma deberá ser capaz de gestionar la información asociada a productos, clientes, compras y proveedores. Así mismo ha proporcionado un listado inicial de la información que quiere almacenar para cada una de estas entidades. No obstante, se prevé que este listado pueda variar en el tiempo y por entrada.

Cliente: Nombre, direcciones de facturación, direcciones de envío, tarjetas de pago, fecha de alta, fecha de último acceso.

Producto: Nombre, código del producto del proveedor, precio sin y con IVA, coste de envío, descuento por rango de fechas, dimensiones, peso, proveedores/almacenes, y otros atributos específicos del producto.

Compra: Productos, cliente, precio de compra, fecha de compra, dirección de envío.

Proveedor: Nombre, direcciones almacenes.

La primera fase del desarrollo de la plataforma se centrará en el modelado de las entidades producto, cliente, proveedor y compra. Se pide diseñar e implementar una base de datos que albergue la información asociada a dichas entidades, así como el diseño e implementación de un ODM específico que administre la información asociada a cada entidad y la comunicación con la base de datos.

Cada producto, cliente, proveedor y compra tendrá asociado un identificador único. Para las direcciones que se proporcionan se ha de incluir en el modelo la geolocalización en formato *GeoJSON* mediante la función *geoCityGeoJSON*. Esta función utiliza una API pública con acceso limitado y por tanto no se pueden realizar consultas de forma masiva. Consultar haciendo uso de sleeps y guardar los resultados.

Dado que se prevé que el listado de atributos pueda aumentar a lo largo del tiempo y sea necesario incluirlos en la base de datos, se proporcionará un archivo de configuración por cada modelo del ODM que permita especificar por cada modelo aquellos atributos obligatorios (que tienen que estar para cada entrada) y admitidos (atributos válidos/aceptados que no tienen porque estar en cada entrada).

Requisitos de la práctica

Diseñar e implementar el/los modelo/s que gestione la información relativa a las entidades producto, cliente, almacén y compra. Dichos modelos deben asegurar que los datos que se proveen son correctos (están los datos requeridos y no hay más datos que los admitidos).

El/los modelo/s, deben implementar:

- Un método que permita almacenar los datos en BBDD (*modelo.save()*) de modo que si es un documento nuevo lo inserte y si es un documento ya existente, solo actualice aquellos campos que hayan sido modificados. No se podrán eliminar documentos de la BBDD en este sistema.
- Un método de clase que permita realizar consultas en BBDD (*modelo.query()*) y devuelva el resultado de la consulta en formato modelo.
    - No será necesario implementar una nueva metodología de consulta sino que el método simplemente recibirá los mismos argumentos que reciben los métodos de consulta de *pymongo*. Todas las consultas se realizarán utilizando el método *aggregate*. Por lo tanto, la consulta que se pase como argumento deberá ser un pipeline.
    - El método *query()* deberá devolver un objeto cursor que apunte al primer documento de la lista de documentos. Este objeto tendrá un método *next()* que devolverá el primer documento en formato modelo. También tendrá una variable *alive* que permitirá conocer si existen más documentos en el cursor.
- Un método *modelo.update*() que permita modificar los datos en memoria del modelo. Se deberá realizar un control de las variables modificadas de modo que cuando se actualice la información en la BBDD, solo se envíen los datos modificados.
- Un método de clase *modelo.init_class()* que permita definir los datos que deben inicializarse de forma obligatoria en el modelo y cuáles son los datos que se pueden aceptar. Además, se proporcionará la conexión a la BBDD para que sea accesible a todos los objetos de la clase y la propia clase.

Consideraciones adicionales:

- Las localizaciones deberán ser implementadas siguiendo el formato geoJSON y deberá utilizarse indexado 2dsphere.

Además, se debe proveer una colección de consultas que permita acceder a la información procesada. Estas consultas deberán estar en formato *pipeline* para poder realizar las consultas a través del método *aggregate* de mongoDB. Se ha de tener en cuenta que no todas las consultas se podrán ejecutar mediante el método *query* de los modelos ya que no están diseñados para combinar información de varios modelos diferentes o devolver información agregada. Es por ello que las consultas se comprobarán usando directamente la función *aggregate* de *pymongo*.

1. Listado de todas las compras de un cliente
2. Listado de todos los proveedores para un producto.
3. Listado de todos los productos diferentes comprados por un cliente
4. Calcular el peso y volumen total de los productos comprados por un cliente un día determinado.
5. Calcular el número medio de envíos por mes y almacén.
6. Listado con los tres proveedores con más volumen de facturación. Mostrar proveedor y volumen de facturación.

7. Listado de almacenes cerca de unas coordenadas determinadas (100km de distancia máxima) ordenadas por orden de distancia.
8. Listado de compras con destino dentro de un polígono cuyos vértices vienen definidos por coordenadas.

Normativa de realización, entrega y evaluación de la práctica:

- La práctica se realizará y entregará en grupos de hasta dos integrantes.
- La práctica se realizará en python y haciendo uso de mongoDB.
- La práctica deberá ir acompañada de las pruebas necesarias para comprobar el buen funcionamiento de la funcionalidad que se pide.
- La entrega se compondrá de un único fichero ZIP, que contendrá el directorio del proyecto con un listado de las dependencias necesarias.
- Se considerará suspensa toda práctica cuyo fichero comprimido no contenga los ficheros fuente.
- La entrega deberá hacerse mediante el campus virtual antes del domingo 21 de octubre de 2020 a las 23:59 horas (hora peninsular en España).
- Las prácticas entregadas fuera de plazo serán calificadas sobre 9. Por cada día de retraso en la entrega se reducirá el rango de calificación en 0,2 puntos.
- La entrega se compondrá de un único fichero .py renombrado con el número de la práctica P#, seguido del número del grupo G#, y finalmente seguido con el nombre y el primer apellido de los alumnos integrantes del grupo, separados mediante guiones bajos '_'.

  Ejemplo: *P1_G25_Quijote_de_la_Mancha-Sancho_Panza.zip*
- Cualquier sospecha de COPIA entre dos o más prácticas o de código obtenido en internet derivará en la calificación de 0 para todos los alumnos involucrados en la evaluación en curso y la siguiente. En caso de que el alumno tenga duda de que el código pueda ser susceptible de ser entendido como copia, consultar con el profesor antes de la entrega.