Esta clase va a ser

- grabada

Clase 17. DESARROLLO AVANZADO DE BACKEND

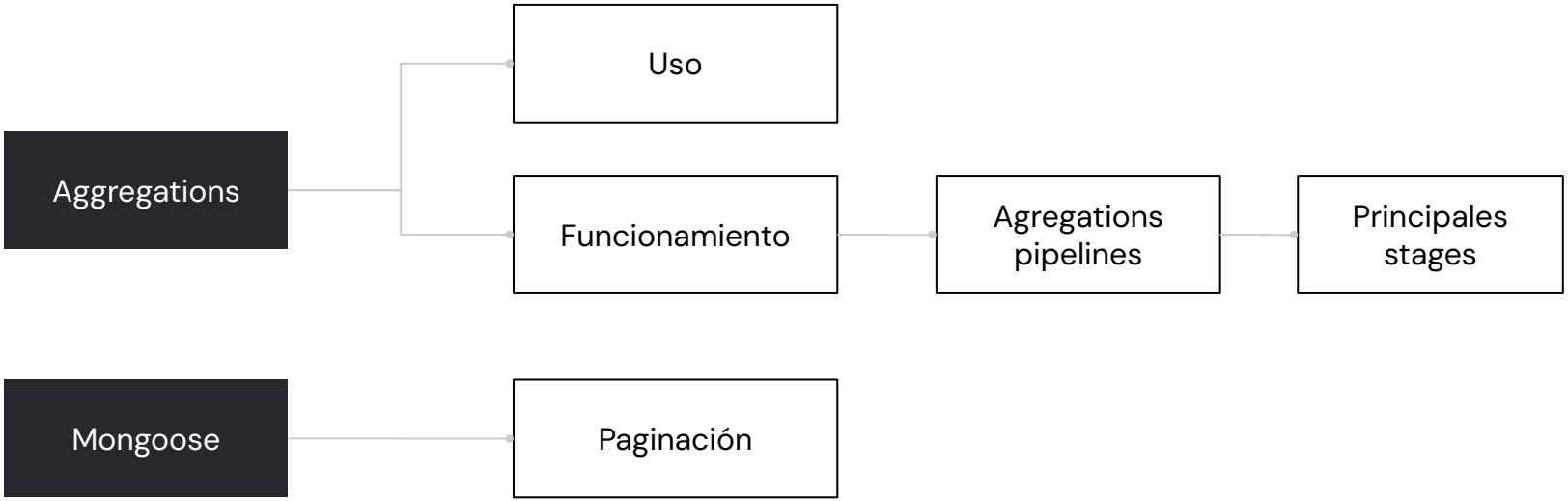
Mongo avanzado II

CODERHOUSE

Objetivos de la clase

- Comprender el concepto de aggregation
- Realizar caso práctico para una aggregation
- Comprender el concepto de paginación y utilización de mongoose-paginatev2

MAPA DE CONCEPTOS



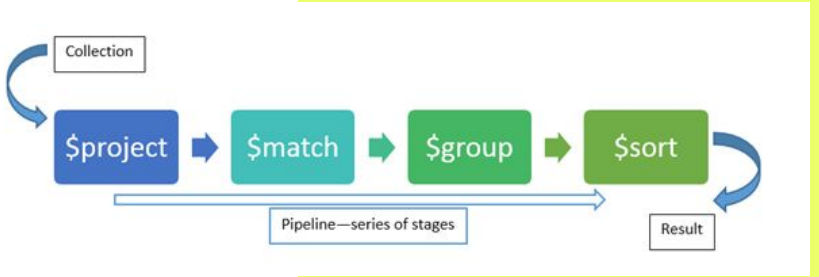
Aggregations

Aggregation

Consiste en la realización de múltiples operaciones, generalmente sobre múltiples documentos.

Pueden utilizarse para:

- ✓ Agrupar documentos con base en un criterio específico.
- ✓ Realizar alguna operación sobre dichos documentos, con el fin de obtener un solo resultado.
- ✓ Analizar cambios de información con el paso del tiempo.



CODERHOUSE

Funcionamiento

Los aggregation pipelines consistirán en un conjunto de pasos (stages), donde cada paso corresponderá a una operación a realizar.

Podemos definir tantas stages como necesitemos con el fin de llegar a los resultados esperados.

Los documentos resultantes de la stage que finalice, se utilizan como “input” de la siguiente stage, y así sucesivamente hasta llegar al final.

Un ejemplo de un pipeline de aggregation puede ser:

1. Primero filtra los documentos que tengan un valor x mayor a 20
2. Luego ordénalos de mayor a menor
3. Luego en un nuevo campo devuelve el valor máximo
4. Luego en un nuevo campo devuelve el valor mínimo
5. Luego en un nuevo campo devuelve la suma total de todos los documentos

Principales stages disponibles en un aggregation pipeline

$count : Cuenta el número de documentos disponibles que se encuentren en la stage actual.

$group: Permite agrupar los documentos disponibles en nuevos grupos según un criterio especificado. **cada grupo cuenta con un _id nuevo,** además de los valores acumulados.

$limit: Limita el número de documentos que saldrán de dicha stage.

$lookup: Permite realizar un “left join” (combinación de campos), de una colección de la misma base de datos a los documentos de la stage actual.

Puedes revisar la lista completa de stages

Principales stages disponibles en un aggregation pipeline

$set / $addFields : Agregan una nueva propiedad a los documentos que se encuentren en dicha stage.

$skip: Devuelve sólo los documentos que se encuentren después del offset indicado.

$sort: Ordena los documentos en la stage actual.

$match: Devuelve sólo los documentos que cumplan con un criterio de búsqueda, podemos colocar filtros comunes aquí

$merge: escribe los resultados del pipeline en una colección. Debe ser la última stage del pipeline para poder funcionar.

Puedes revisar la lista completa de stages

ACTIVIDAD EN CLASE

Agrupación de estudiantes

Realizar las siguientes consultas en una colección de estudiantes.

Los estudiantes deben contar con los datos:

- ✓ first_name : Nombre
- ✓ last_name : Apellido
- ✓ email: correo electrónico
- ✓ gender: género
- ✓ grade: calificación
- ✓ group : grupo

(El profesor puede proporcionarte algunos datos de prueba)

ACTIVIDAD EN CLASE

Agrupación de estudiantes

Una vez generados tus datos de prueba:

1. Obtener a los estudiantes agrupados por calificación del mejor al peor
2. Obtener a los estudiantes agrupados por grupo.
3. Obtener el promedio de los estudiantes del grupo 1B
4. Obtener el promedio de los estudiantes del grupo 1A
5. Obtener el promedio general de los estudiantes.
6. Obtener el promedio de calificación de los hombres
7. Obtener el promedio de calificación de las mujeres.

Break

¡10 minutos y volvemos!

Paginación con mongoose

Paginación

Cuando recién trabajamos con nuestros primeros datos, es maravilloso ver cómo nuestras búsquedas pueden devolvernos todos los datos que necesitamos.

Sin embargo, esta “maravilla” comienza a convertirse en un problema cuando el número de datos que tenemos incrementa considerablemente.

Recordemos que los datos que nosotros obtenemos, al final tenemos que enviarlos a través del internet, para que el cliente la utilice.

¿Qué tan lento será el proceso de enviar 5000 usuarios en una sola vuelta? ¿Y 10 mil?

Paginación utilizando mongoose-paginate-v2

CODERHOUSE

Nociones de mongoose-paginate-v2

- ✓ docs: Los documentos devueltos en la página
- ✓ totalDocs: Los documentos totales de la consulta realizada.
- ✓ limit: Límite de resultados por página.
- ✓ page: Página actual en la que nos encontramos
- ✓ totalPages: Páginas totales que pueden ser solicitadas en la búsqueda.
- ✓ hasNextPage: Indica si es posible avanzar a una página siguiente.
- ✓ nextPage: Página siguiente en la búsqueda
- ✓ hasPrevPage: Indica si es posible retroceder a una página anterior.
- ✓ prevPage: Página anterior en la búsqueda.
- ✓ pagingCounter: Número de documento en relación con la página actual.

Para pensar

¿Qué diferencia habría entre un mongoose-paginate y un skip, offset, limit en una consulta find?

¿Qué debo elegir?

Hands on lab

En esta instancia de la clase **crearemos una paginación elemental con los estudiantes del ejercicio pasado,** a partir de un ejercicio práctico

¿De qué manera?
El profesor te demostrará cómo hacerlo y tú lo puedes ir replicando en tu computadora. Si surgen dudas las puedes compartir para resolverlas en conjunto de la mano de los tutores.

Tiempo estimado: **35-40 minutos**

CODERHOUSE

HANDS ON LAB

Sistema de paginación de estudiantes

¿Cómo lo hacemos? **Se creará una vista simple con Handlebars donde se podrán mostrar los estudiantes**

- ✓ Los estudiantes serán mostrados en la vista “/students”
- ✓ Debe existir un enlace “Anterior”” para regresar a los estudiantes anteriores, **siempre que haya una página anterior**
- ✓ Debe existir un enlace “Siguiente” para continuar con la paginación de estudiantes, **siempre que haya una página siguiente**
- ✓ Debe indicarse la página actual.
- ✓ Todo debe vivir en un servidor de express escuchando en el puerto 8080.

Entrega Final de tu Proyecto

Deberás entregar el proyecto que has venido armando, cambiando persistencia en base de datos, además de agregar algunos endpoints nuevos a tu Api

CODERHOUSE

Profesionalizando la BD

Objetivos generales

- ✓ Contarás con Mongo como sistema de persistencia principal
- ✓ Tendrás definidos todos los endpoints para poder trabajar con productos y carritos.

Objetivos específicos

- ✓ Profesionalizar las consultas de productos con filtros, paginación y ordenamientos
- ✓ Profesionalizar la gestión de carrito para implementar los últimos conceptos vistos.

Formato

- ✓ Link al repositorio de Github, sin la carpeta de node_modules

Sugerencias

- ✓ Permitir comentarios en el archivo
- ✓ La lógica del negocio que ya tienes hecha no debería cambiar, sólo su persistencia.
- ✓ Los nuevos endpoints deben seguir la misma estructura y lógica que hemos seguido.

Video explicativo de la entrega final

ENTREGA DEL PROYECTO FINAL

Se debe entregar

- ✓ Con base en nuestra implementación actual de productos, modificar el método GET / para que cumpla con los siguientes puntos:
  - Deberá poder recibir por query params un limit (opcional), una page (opcional), un sort (opcional) y un query (opcional)
    - -limit permitirá devolver sólo el número de elementos solicitados al momento de la petición, en caso de no recibir limit, éste será de 10.
    - page permitirá devolver la página que queremos buscar, en caso de no recibir page, ésta será de 1
    - query, el tipo de elemento que quiero buscar (es decir, qué filtro aplicar), en caso de no recibir query, realizar la búsqueda general
    - sort: asc/desc, para realizar ordenamiento ascendente o descendente por precio, en caso de no recibir sort, no realizar ningún ordenamiento

ENTREGA DEL PROYECTO FINAL

Se debe entregar

- ✓ El método GET deberá devolver un objeto con el siguiente formato:

```
{
        status:success/error
        payload: Resultado de los productos solicitados
        totalPages: Total de páginas
        prevPage: Página anterior
        nextPage: Página siguiente
        page: Página actual
        hasPrevPage: Indicador para saber si la página previa existe
        hasNextPage: Indicador para saber si la página siguiente existe.
        prevLink: Link directo a la página previa (null si hasPrevPage=false)
        nextLink: Link directo a la página siguiente (null si hasNextPage=false)
}
```

- ✓ Se deberá poder buscar productos por categoría o por disponibilidad, y se deberá poder realizar un ordenamiento de estos productos de manera ascendente o descendente por precio.

ENTREGA DEL PROYECTO FINAL

Se debe entregar

- ✓ Además, agregar al router de carts los siguientes endpoints:
  - DELETE api/carts/:cid/products/:pid deberá eliminar del carrito el producto seleccionado.
  - PUT api/carts/:cid deberá actualizar el carrito con un arreglo de productos con el formato especificado arriba.
  - PUT api/carts/:cid/products/:pid deberá poder actualizar SÓLO la cantidad de ejemplares del producto por cualquier cantidad pasada desde req.body
  - DELETE api/carts/:cid deberá eliminar todos los productos del carrito
  - Esta vez, para el modelo de Carts, en su propiedad products, el id de cada producto generado dentro del array tiene que hacer referencia al modelo de Products. Modificar la ruta /:cid para que al traer todos los productos, los traiga completos mediante un “populate”. De esta manera almacenamos sólo el Id, pero al solicitarlo podemos desglosar los productos asociados.

ENTREGA DEL PROYECTO FINAL

Se debe entregar

- ✓ Modificar la vista index.handlebars en el router de views ‘/products’, creada en la pre-entrega anterior, para visualizar todos los productos con su respectiva paginación. Además, cada producto mostrado puede resolverse de dos formas:
  - Llevar a una nueva vista con el producto seleccionado con su descripción completa, detalles de precio, categoría, etc. Además de un botón para agregar al carrito.
    Sugerencia de la ruta: “/products/:pid”.
  - Contar con el botón de “agregar al carrito” directamente, sin necesidad de abrir una página adicional con los detalles del producto.
- ✓ Además, agregar una vista en ‘/carts/:cid (cartId) para visualizar un carrito específico, donde se deberán listar SOLO los productos que pertenezcan a dicho carrito.

¿Preguntas?

Muchas gracias.