**Aggregation**

Overview:

* The Aggregation Framework in MongoDB provides a powerful set of tools for performing data aggregation operations.
* It allows for processing and analyzing data within collections to obtain aggregated results.
* Aggregation operations are performed using a pipeline-based approach, where documents pass through multiple stages.
* Each stage performs a specific operation on the documents, such as filtering, grouping, sorting, projecting, etc.
* Aggregation operations can be executed using the aggregate() method in the mongoTemplate object, providing an aggregation pipeline and the name of the collection to perform the aggregation on.
* Aggregation results can be further processed or mapped to specific data structures for easier consumption.
* The typical stage of Aggregation Pipeline is Group,Match,Sort,aggregate
* Match will filter the data, Group will group the data and can perform the operations like count,sum,min,max and Sort is used for sorting in the ascending or descending order

Code:

1. Filter by last name and Sort it

* This method retrieves a list of employees based on their last name.
* It first creates a MatchOperation to filter documents where the empLastName field matches the provided last name.
* Then, it creates a SortOperation to sort the results based on the empLastName field in descending order.
* Finally, it creates an Aggregation object, adds the matchOperation and sortOperation to it, and executes the aggregation using mongoTemplate.aggregate().
* The results are then mapped to a list of Employee objects and returned.

@GetMapping("/byLastName/{empLastName}")

List<Employee> findByLastName(@PathVariable String empLastName)

{

//Criteria criteria =new Criteria("empLastName");

//MatchOperation

MatchOperation matchOperation= Aggregation.match(new Criteria("empLastName").is(empLastName));

//SortOperation

SortOperation sortOperation=Aggregation.sort(Sort.by(Sort.Direction.DESC,"empLastName"));

//Aggregation

Aggregation aggregation=Aggregation.newAggregation(matchOperation,sortOperation);

AggregationResults output=mongoTemplate.aggregate(aggregation,"employee", Employee.class);

return output.getMappedResults();

}

1. Filter by last name,count it and sort it

* This method is similar to findByLastName but also includes counting the number of employees with the specified last name.
* It creates a MatchOperation to filter documents by last name.
* Then, it creates a GroupOperation to group the filtered documents and count the occurrences.
* There is no sorting operation applied in this method.
* The aggregation is executed similarly to the previous method, and the results are mapped to a list of EmpCount objects, which likely have a field to hold the count.

@GetMapping("/byLastNameCount/{empLastName}")

List<Employee> findByLastNameCount(@PathVariable String empLastName)

{

//MatchOperation

MatchOperation matchOperation= Aggregation.match(new Criteria("empLastName").is(empLastName));

//Group

GroupOperation groupOperation=Aggregation.group().count().as("count");

//SortOperation

SortOperation sortOperation=Aggregation.sort(Sort.by(Sort.Direction.DESC,"empLastName"));

//Aggregation

Aggregation aggregation=Aggregation.newAggregation(matchOperation,groupOperation);

AggregationResults output=mongoTemplate.aggregate(aggregation,"employee", EmpCount.class);

return output.getMappedResults();

}

1. Max, min, sum of ages

* This method retrieves aggregated data related to the ages of all employees.
* It creates a GroupOperation to group all documents without any specific criteria.
* Within the group operation, it calculates the sum, maximum, and minimum age of all employees.
* There is no sorting operation applied in this method.
* The aggregation is executed similarly to the other methods, and the results are mapped to a list of EmpCount objects.

@GetMapping("/allAgesOperations")

List<Employee> findAllAgesOperations() {

// Group

GroupOperation groupOperation = Aggregation.group()

.sum("age").as("sum") //”age” is the variable which should match to the DB column and “sum” should match the variable of mapping class

.max("age").as("maxAge")

.min("age").as("minAge");

// Aggregation

Aggregation aggregation = Aggregation.newAggregation(groupOperation);

AggregationResults output = mongoTemplate.aggregate(aggregation, "employee", EmpCount.class); //The employee is the object and EmpCount is the mapping class name

return output.getMappedResults();

}

**Pagination**

Pagination is a technique used to divide a large set of data into smaller, manageable subsets or pages. It's commonly used in applications where displaying all data at once is not practical due to performance or user experience considerations.

**Why Use Pagination?**

* Improved Performance: Retrieving and displaying a large dataset all at once can be resource-intensive and slow. Pagination helps improve performance by fetching and displaying data in smaller chunks.
* Enhanced User Experience: Displaying data in manageable pages makes it easier for users to navigate through large datasets, improving usability and reducing cognitive load.
* Reduced Network Traffic: Sending smaller amounts of data per request reduces network traffic, leading to faster response times and better scalability.

Pagination in MongoDB:

In MongoDB, pagination is typically implemented using the skip() and limit() methods along with sorting for consistency across pages. Here's how it works:

Skip and Limit:

* The skip() method is used to skip a specified number of documents and retrieve results starting from the specified position.
* The limit() method is used to limit the number of documents returned in a query.

**Code:**

* The method uses the MongoDB Aggregation Framework for pagination.
* It creates two aggregation operations:
* skipOperation: Skips a specified number of documents based on the current page number and page size.
* limitOperation: Limits the number of documents returned to the specified page size.
* These operations are then added to the aggregation pipeline.
* The aggregation pipeline is executed using the mongoTemplate.aggregate() method.
* The results are mapped to a list of Person objects and returned

@PostMapping("/person/save")

public String savePerson(@RequestBody AddPersonRequest request)

{

Person person = personRepositroy.findByName(request.getName()+" "+0);

if(person != null)

{

return "Person Already exists";

}

for(int i=0;i<50;i++)

{

person = new Person();

person.setId(UUID.randomUUID().toString());

person.setName(request.getName()+" "+i);

person.setAddress(request.getAddress()+" "+i);

personRepositroy.save(person);

}

return "Persons are saved";

}

@GetMapping("/persons")

public List<Person> getPersonsPerPage(@RequestParam int pageNumber, @RequestParam int pageSize) {

//pagesize=no of documents per page

//pageNumber=current page number

AggregationOperation skipOperation = Aggregation.skip((long) pageNumber \* pageSize);

AggregationOperation limitOperation = Aggregation.limit(pageSize);

// Aggregation

Aggregation aggregation = Aggregation.newAggregation(

skipOperation,

limitOperation

);

AggregationResults<Person> output = mongoTemplate.aggregate(aggregation, "person", Person.class);

return output.getMappedResults();

}

}