Implement Mongo DB Filters In .NET Using MongoDB.Driver

In this article, we will understand how to use various filters available in **MongoDB.Driver** using .NET and C#. We will use the MongoDB.Driver package to perform various operations. This article focuses on MongoDB operations, so we will not implement a high-level architecture for the .NET project. Instead, we will create a .NET 8 Web API project and implement all operations directly in the controller rather than creating separate service or repository classes. You can implement those architectures based on your requirements.

If you want to understand how to configure MongoDB with a .NET project and perform various operations on the database and collections, refer to the previous article: Implementing MongoDB with .NET. Implementing MongoDB with .NET.

Create Collection Instance

```
private readonly IMongoCollection<Employee> _employeeCollection;

public MongoFiltersController(IMongoClient mongoClient, IConfiguration configura
{
    var databaseName = configuration.GetValue<string>("MongoDbSettings:DatabaseN
    var employeeCollection = configuration.GetValue<string>("MongoDbSettings:Col
    var database = mongoClient.GetDatabase(databaseName);
    _employeeCollection = database.GetCollection<Employee>(employeeCollection);
}
```

This code sets up a MongoDB collection for use within a controller. The constructor takes an **IMongoClient** and **IConfiguration** as parameters. It retrieves the database and collection names from the configuration file, and gets the database instance using **mongoClient.GetDatabase(databaseName),** and then get the Employee collection with the **database.GetCollection(employeeCollection)**. This collection is stored in the **_employeeCollection** field, making it available for use in other methods within the controller. This setup allows you to perform CRUD operations on the Employee collection.

Below is the **Employee** collection used in the examples.

```
public class Employee
{
    [BsonId]
    [BsonRepresentation(BsonType.ObjectId)]
    public string Id { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
    public string Designation { get; set; }
    public decimal Salary { get; set; }
    public DateTime DateOfBirth { get; set; }
    public string[] Skills { get; set; }
}
```

Equality Filter

```
[HttpGet("eq")]
public async Task<IActionResult> Equal(string name)
{
    var filter = Builders<Employee>.Filter.Eq(emp => emp.Name, name);
    var results = await _employeeCollection.Find(filter).ToListAsync();
    return Ok(results);
}
```

The **Eq** method in the MongoDB .NET driver is used to create an equality filter. This filter selects documents where a specified field matches a given value. The method is part of the **FilterDefinitionBuilder** class, accessible through Builders. Filter.

For instance, **Eq** can be used to find all documents in a collection where a specific field, like Name, equals a given value, such as “John Doe”. This filter is then used in queries to retrieve matching documents from the MongoDB collection. The **Eq** method is essential for precise filtering, making it easy to query documents based on exact field value matches.

Not Equal Filter

```
[HttpGet("ne")]
public async Task<IActionResult> NotEqual(string name)
{
    var filter = Builders<Employee>.Filter.Ne(emp => emp.Name, name);
    var results = await _employeeCollection.Find(filter).ToListAsync();
    return Ok(results);
}
```

The **Ne** method in the MongoDB .NET driver creates a filter to select documents where a specified field is not equal to a given value. It’s used to exclude documents based on exact field value matches, providing essential functionality for precise data filtering in MongoDB queries.

Greater Than Filter

```
[HttpGet("gt")]
public async Task<IActionResult> GreaterThan(int age)
{
    var filter = Builders<Employee>.Filter.Gt(emp => emp.Age, age);
    //var filter = Builders<Employee>.Filter.Gt(emp => emp.DateOfBirth, DateTime
```

```
    var results = await _employeeCollection.Find(filter).ToListAsync();
    return Ok(results);
}
```

The **Gt** method in the MongoDB .NET driver constructs a filter to select documents where a specified numeric field (such as **int**, **long**, **double**) or date/time field (**DateTime**, **DateTimeOffset**) is greater than a given value. It supports these data types for precise data comparisons in MongoDB databases.

Greater Than or Equal Filter

```
[HttpGet("gte")]
public async Task<IActionResult> GreaterThanOrEqual(int age)
{
    var filter = Builders<Employee>.Filter.Gte(emp => emp.Age, age);
    //var filter = Builders<Employee>.Filter.Gte(emp => emp.DateOfBirth, DateTim
    var results = await _employeeCollection.Find(filter).ToListAsync();
    return Ok(results);
}
```

The **Get** method in the MongoDB .NET driver constructs a filter to select documents where a specified numeric field (such as **int**, **long**, **double**) or date/time field (**DateTime**, **DateTimeOffset**) is greater than or equal to a given value. This method expands on the functionality of Gt by including documents where the field value matches the specified value as well.

Less Than Filter

```
[HttpGet("lt")]
public async Task<IActionResult> LessThan(int age)
```

```
{
    var filter = Builders<Employee>.Filter.Lt(emp => emp.Age, age);
    //var filter = Builders<Employee>.Filter.Lt(emp => emp.DateOfBirth, DateTime
    var results = await _employeeCollection.Find(filter).ToListAsync();
    return Ok(results);
}
```

The **Lt** method in the MongoDB .NET driver constructs a filter to select documents where a specified numeric field (such as **int**, **long**, **double**) or date/time field (**DateTime**, **DateTimeOffset**) is less than a given value. This method is essential for querying MongoDB collections based on numeric or temporal criteria, facilitating precise data comparisons.

Less Than or Equal Filter

```
[HttpGet("lte")]
public async Task<IActionResult> LessThanOrEqual(int age)
{
    var filter = Builders.Employee.Filter.Lte(emp => emp.Age, age);
    //var filter = Builders<Employee>.Filter.Lte(emp => emp.DateOfBirth, DateTim
    var results = await _employeeCollection.Find(filter).ToListAsync();
    return Ok(results);
}
```

The **Lte** method in the MongoDB .NET driver constructs a filter to select documents where a specified numeric field (such as **int**, **long**, **double**) or date/time field (**DateTime**, **DateTimeOffset**) is less than or equal to a given value. This method supports both numeric and temporal comparisons, allowing for precise data filtering in MongoDB collections.

In Filter

```
[HttpGet("in")]
public async Task<IActionResult> In([FromQuery] string[] names)
{
    var filter = Builders<Employee>.Filter.In(emp => emp.Name, names);
    var results = await _employeeCollection.Find(filter).ToListAsync();
    return Ok(results);
}
```

The **In** method in MongoDB's .NET driver creates a filter to select documents where a specified field (Name in this case) matches any of the values in a provided array (names). This method simplifies querying MongoDB collections based on multiple possible matches for the specified field, such as retrieving documents where an employee's name matches any name in the names array.

Not In Filter

```
[HttpGet("nin")]
public async Task<IActionResult> NotIn([FromQuery] string[] names)
{
    var filter = Builders<Employee>.Filter.Nin(emp => emp.Name, names);
    var results = await _employeeCollection.Find(filter).ToListAsync();
    return Ok(results);
}
```

The **Nin** method in MongoDB's .NET driver constructs a filter to select documents where a specified field (Name in this case) does not match any of the values in a provided array (names). This method is useful for excluding documents from MongoDB collections based on multiple potential matches for the specified field.

And Filter

```
[HttpGet("and")]
public async Task<IActionResult> And(string name, int age)
{
    var filter = Builders<Employee>.Filter.And(
        Builders.Employee.Filter.Eq(emp => emp.Name, name),
        Builders<Employee>.Filter.Eq(emp => emp.Age, age));
    var results = await _employeeCollection.Find(filter).ToListAsync();
    return Ok(results);
}
```

The **And** method in MongoDB's .NET driver constructs a filter to select documents where multiple conditions must all be true simultaneously. In this example, it combines two conditions using **Eq** to filter Employee documents where **Name** equals a specified name and **Age** equals a specified age. This method is useful for creating complex queries that require multiple criteria to be met for document selection in MongoDB collections.

Or Filter

```
[HttpGet("or")]
public async Task<IActionResult> Or(string name, int age)
{
    var filter = Builders.Employee.Filter.Or(
        Builders<Employee>.Filter.Eq(emp => emp.Name, name),
        Builders<Employee>.Filter.Eq(emp => emp.Age, age));
    var results = await _employeeCollection.Find(filter).ToListAsync();
    return Ok(results);
}
```

The **Or** method in MongoDB's .NET driver constructs a filter to select documents where at least one of multiple conditions is true. In this example, it combines two conditions using **Eq** to filter Employee documents where **Name** equals a specified name or **Age** equals a specified age. This method is

essential for creating queries that need to include documents meeting any of several criteria in MongoDB collections.

Exists Filter

```
[HttpGet("exists")]
public async Task<IActionResult> Exists(string fieldName)
{
    var filter = Builders.Employee.Filter.Exists(fieldName);
    // or
    // var filter = Builders<Employee>.Filter.Exists(x=>x.Designation);
    var results = await _employeeCollection.Find(filter).ToListAsync();
    return Ok(results);
}
```

The **Exists** method in MongoDB's .NET driver constructs a filter to select documents where a specified field exists or does not exist. This example demonstrates filtering Employee documents where a specified field exists. Additionally, the method can be modified to filter documents where a specified field does not exist by passing false as the second parameter. This method is essential for querying MongoDB collections based on the presence or absence of specific fields.

Type Filter

```
[HttpGet("type")]
public async Task<IActionResult> Type(string fieldName, BsonType bsonType)
{
    var filter = Builders<Employee>.Filter.Type(fieldName, bsonType);
    var results = await _employeeCollection.Find(filter).ToListAsync();
    return Ok(results);
}
```

The **Type** method in MongoDB's .NET driver constructs a filter to select documents where a specified field matches a given **BSON** type. This method is crucial for querying MongoDB collections based on data types, ensuring precise data retrieval according to specific **BSON** type criteria.

Regular Expression Filter

```
[HttpGet("regex")]
public async Task<IActionResult> Regex()
{
    var pattern = "^J"; // Names starting with 'J'
    //var pattern = "n$"; // Names ending with 'n'
    //var pattern = ".[aei]."; // Names containing 'a', 'e', or 'i'
    //..... and many more
    var filter = Builders<Employee>.Filter.Regex(emp => emp.Name, new BsonRegula
    var results = await _employeeCollection.Find(filter).ToListAsync();
    return Ok(results);
}
```

The **Regex** method in MongoDB's .NET driver creates a filter to find documents where a specified field matches a pattern defined by a regular expression. This method works with string fields, allowing you to perform advanced text searches and filtering. It helps you find documents with text that fit complex patterns, giving you powerful ways to search your data.

All Filter

```
[HttpGet("all")]
public async Task<IActionResult> All([FromQuery] string[] skills)
{
    var filter = Builders<Employee>.Filter.All(emp => emp.Skills, skills);
    var results = await _employeeCollection.Find(filter).ToListAsync();
    return Ok(results);
}
```

The **All** method in MongoDB's .NET driver constructs a filter to select documents where an array field contains all the elements of a specified array. In this example, it filters Employee documents where the **Skills array** contains all the provided skills. This method is useful for querying MongoDB collections based on array contents.

ElemMatch Filter

```
[HttpGet("elemMatch")]
public async Task<IActionResult> ElemMatch(string skill)
{
    var filter = Builders<Employee>.Filter.ElemMatch(emp => emp.Skills, skill);
    var results = await _employeeCollection.Find(filter).ToListAsync();
    return Ok(results);
}
```

The **ElemMatch** method in MongoDB's .NET driver constructs a filter to select documents where at least one element of an array field matches a specified condition. In this example, it filters Employee documents where at least one element of the **Skills array** matches the provided skill. This method is essential for querying MongoDB collections based on conditions applied to array elements.

Size Filter

```
[HttpGet("size")]
public async Task<IActionResult> Size(int size)
{
    var filter = Builders<Employee>.Filter.Size(emp => emp.Skills, size);
    var results = await _employeeCollection.Find(filter).ToListAsync();
    return Ok(results);
}
```

The **Size** method in MongoDB's .NET driver constructs a filter to select documents where an array field has a specified size. In this example, it filters Employee documents where the **Skills array** has the specified number of elements. This method is useful for querying MongoDB collections based on the size of array fields.

Conclusion

In this article, we explored various MongoDB filters available in the **MongoDB.Driver** package using .NET and C#. These filters allow you to perform a wide range of data querying operations in your MongoDB collections.

You can access the source code of this project on GitHub.