Spring Boot MongoDB CRUD

pting boot and mongoDB example from scratch, i need to handle crud opetaions, pagination, sorting, 1 month, 3 months and 1 year filter data

give code from scratch step by step

Here's a **complete Spring Boot + MongoDB CRUD example** from scratch that includes:

- CRUD operations (`Create`, `Read`, `Update`, `Delete`)
- Pagination and sorting
- Date range filters (`1 month`, `3 months`, `1 year`)

✅ Step 1: Create Spring Boot Project

Use https://start.spring.io with:

- **Dependencies**:
  - Spring Web
  - Spring Data MongoDB
  - Lombok (optional, for reducing boilerplate)

Generate and unzip.

✅ Step 2: `pom.xml`

Ensure you have:

```xml
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-mongodb</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
```

```
    </dependency>
    <dependency>
      <groupId>org.projectlombok</groupId>
      <artifactId>lombok</artifactId>
      <optional>true</optional>
    </dependency>
</dependencies>
```

✅ Step 3: `application.properties`

```properties
spring.data.mongodb.uri=mongodb://localhost:27017/ptingdb
spring.data.mongodb.database=ptingdb
```

Make sure MongoDB is running locally.

✅ Step 4: Create Model (`Document`)

```java
package com.example.pting.model;

import lombok.*;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

import java.time.LocalDateTime;

@Document(collection = "documents")
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class MyDocument {
    @Id
    private String id;

    private String title;
```

```
    private String content;
    private LocalDateTime createdAt;
}
```

✅ Step 5: Create Repository

java

```
package com.example.pting.repository;

import com.example.pting.model.MyDocument;
import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.stereotype.Repository;

import java.time.LocalDateTime;
import java.util.List;

@Repository
public interface MyDocumentRepository extends MongoRepository<MyDocument, String> {
    List<MyDocument> findByCreatedAtAfter(LocalDateTime fromDate);
}
```

✅ Step 6: Create Service

java

```
package com.example.pting.service;

import com.example.pting.model.MyDocument;
import com.example.pting.repository.MyDocumentRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.data.domain.*;
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;
import java.util.Optional;

@Service
@RequiredArgsConstructor
```

```java
public class MyDocumentService {

    private final MyDocumentRepository repository;

    public MyDocument create(MyDocument doc) {
        doc.setCreatedAt(LocalDateTime.now());
        return repository.save(doc);
    }

    public Optional<MyDocument> getById(String id) {
        return repository.findById(id);
    }

    public Page<MyDocument> getAll(int page, int size, String sortBy, boolean desc) {
        Sort sort = desc ? Sort.by(sortBy).descending() : Sort.by(sortBy).ascending();
        Pageable pageable = PageRequest.of(page, size, sort);
        return repository.findAll(pageable);
    }

    public MyDocument update(String id, MyDocument updated) {
        return repository.findById(id).map(doc -> {
            doc.setTitle(updated.getTitle());
            doc.setContent(updated.getContent());
            return repository.save(doc);
        }).orElseThrow(() -> new RuntimeException("Document not found"));
    }

    public void delete(String id) {
        repository.deleteById(id);
    }

    public List<MyDocument> getLastXMonths(int months) {
        LocalDateTime fromDate = LocalDateTime.now().minusMonths(months);
        return repository.findByCreatedAtAfter(fromDate);
    }
}
```

✅ Step 7: Create Controller

java

```java
package com.example.pting.controller;

import com.example.pting.model.MyDocument;
import com.example.pting.service.MyDocumentService;
import lombok.RequiredArgsConstructor;
import org.springframework.data.domain.Page;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/documents")
@RequiredArgsConstructor
public class MyDocumentController {

    private final MyDocumentService service;

    // Create
    @PostMapping
    public MyDocument create(@RequestBody MyDocument doc) {
        return service.create(doc);
    }

    // Read by ID
    @GetMapping("/{id}")
    public MyDocument getById(@PathVariable String id) {
        return service.getById(id).orElseThrow(() -> new RuntimeException("Not found"));
    }

    // Read All with Pagination + Sorting
    @GetMapping
    public Page<MyDocument> getAll(
            @RequestParam(defaultValue = "0") int page,
            @RequestParam(defaultValue = "5") int size,
            @RequestParam(defaultValue = "createdAt") String sortBy,
            @RequestParam(defaultValue = "false") boolean desc
    ) {
        return service.getAll(page, size, sortBy, desc);
    }

    // Update
    @PutMapping("/{id}")
```

```
    public MyDocument update(@PathVariable String id, @RequestBody MyDocument doc) {
        return service.update(id, doc);
    }

    // Delete
    @DeleteMapping("/{id}")
    public void delete(@PathVariable String id) {
        service.delete(id);
    }

    // Filters by time range
    @GetMapping("/filter")
    public List<MyDocument> getByTimeRange(@RequestParam String range) {
        switch (range.toLowerCase()) {
            case "1month":
                return service.getLastXMonths(1);
            case "3months":
                return service.getLastXMonths(3);
            case "1year":
                return service.getLastXMonths(12);
            default:
                throw new IllegalArgumentException("Invalid range");
        }
    }
}
```

✅ Step 8: Run and Test

Start your application and test using **Postman** or **cURL**.

Example:

bash

```
curl -X POST http://localhost:8080/documents -H "Content-Type: application/json" -d '{"title":"Test","content":"Hello World"}'

curl "http://localhost:8080/documents?page=0&size=5&sortBy=createdAt&desc=true"

curl "http://localhost:8080/documents/filter?range=1month"
```

✅ Step 9: MongoDB Sample Data Format

A document in MongoDB will look like:

```json
{
  "_id": "64fxxxx",
  "title": "Test",
  "content": "Hello World",
  "createdAt": "2025-08-06T11:00:00"
}
```

Let me know if you want Swagger/OpenAPI, DTO mapping, or MongoTemplate-based advanced querying.

getLastXMonths also we need to add pagination and sorting right

Yes, you're absolutely right ✅ — `getLastXMonths(int months)` should also support **pagination and sorting**, especially for large datasets.

✅ Step-by-Step Update

We'll enhance the `getLastXMonths(...)` method in the **Service** layer and add support in the **Controller**.

🔧 Step 1: Update `MyDocumentService`

```java
public Page<MyDocument> getLastXMonths(int months, int page, int size, String sortBy,
boolean desc) {
    LocalDateTime fromDate = LocalDateTime.now().minusMonths(months);
    Sort sort = desc ? Sort.by(sortBy).descending() : Sort.by(sortBy).ascending();
    Pageable pageable = PageRequest.of(page, size, sort);
```

```
    return repository.findByCreatedAtAfter(fromDate, pageable);
}
```

> ✅ You now need to update the repository method to support pagination.

🔧 Step 2: Update `MyDocumentRepository`

```java
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;

Page<MyDocument> findByCreatedAtAfter(LocalDateTime fromDate, Pageable pageable);
```

🔧 Step 3: Update Controller

Update `filter` endpoint to support pagination + sorting:

```java
@GetMapping("/filter")
public Page<MyDocument> getByTimeRange(
        @RequestParam String range,
        @RequestParam(defaultValue = "0") int page,
        @RequestParam(defaultValue = "5") int size,
        @RequestParam(defaultValue = "createdAt") String sortBy,
        @RequestParam(defaultValue = "false") boolean desc
) {
    int months;
    switch (range.toLowerCase()) {
        case "1month":
            months = 1;
            break;
        case "3months":
            months = 3;
            break;
        case "1year":
            months = 12;
            break;
```

```
        default:
            throw new IllegalArgumentException("Invalid range");
    }

    return service.getLastXMonths(months, page, size, sortBy, desc);
}
```

✅ Sample URL

```http
GET /documents/filter?range=3months&page=0&size=10&sortBy=title&desc=true
```

This will return:

- Documents created in the last 3 months
- Sorted by `title` in descending order
- Paginated with 10 per page

Let me know if you want filters like `createdAt between` custom dates or combine filters like `title contains`, etc.