**MicroService**

\*) Spring Cloud - Netflix Eureka Server :

Eureka Server is Register and Discovery Server.

It will hold details of Microservices(MS) running (ServiceInstance).

Every MS, must be registered with Eureka Server.

One MS can find/connect with other MS using Eureka Only, ie called as Discovery.

\*) Consider we are writing one MS: PaymentService , this is Registered(published)

with Eureka then Details are given as

----------------------Eureka Server (ServiceInstance)---------------------------

ServiceId InstanceId HOST DETAILS LOAD FACTOR

[Application-Name] [UniqueId For Instance] [IP PORT ] [C.L./M.L.]

PAYMENT-SEVICE PS:859-a0125 192.168.0.5 8080 0/200

PAYMENT-SEVICE PS:95ab05-001 168.25.12.6 9091 0/200

....

----------------------------------------------------------------------------------

\*) Inside Eurka, there will be no document concept (NO XML/NO OTHER FILES OF PROJECT).

Only ServiceInstance details will be available.

\*) ServiceId is actually project name : spring.application.name

\*) InstanceId is unique number in String format (In realtime it is generated)

If there is only one instance then providing InstanceId is optional.

In that case InstanceId=ServiceId.

\*) Host Details are auto-detected by Server where we are running.

Server (System) IP/PORT.

\*) Apache Zookeeper is similer tool for Netflix-Eureka.

================= Eureka Server Setup =========================

1. Create one Spring Starter Project

Name : SpringCloudEurekaServer

Dep : Eureka Server

2. application.properties

server.port=8761

eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false

3. At Starter class : @EnableEurekaServer

4. Run Starter class and enter URL:

http://localhost:8761/

========================================================================

\*) Note:

a) Every MS must be registerd with Eureka Server

for that inside MS project we should add below key

eureka.client.register-with-eureka=true.

But this is optional to add, bcoz Every MS project is connected to Spring Cloud

project that gives default value is true.

So, only one time, inside Eureka Server write eureka.client.register-with-eureka=false

Bcoz, Eureka Server, itself can't be registered.

b) If one MS wants to communicate with another MS by using Eureka

then inside MS we should add below key:

eureka.client.fetch-registry=true

That indicates Eureka Server is supporting to fetch instance details, for

Intra-communication.

Inside Eureka Server write same key with false. Bcoz, Eureka server will never

try to fetch register, it is itself having register.

eureka.client.fetch-registry=false.

Spring Coud has made default as true, as it is mostly used in Multiple MS with value true.

1. Eureka Server

Name : SpringCloudEurekaServer

Dep : Eureka Server

=> At starter class : @EnableEurekaServer

-application.properties--

server.port=8761

eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false

=> Startr main class and enter URL

http://localhost:8761/

-------------------------------------------------------

2. Microservice Application

Name: SpringCloudPaymentService

Dep : Spring web, Eureka Discovery Client.

=> At Starter class level : @EnableEurekaClient

\*) application.properties

server.port=9009

#ServiceId

spring.application.name=PAYMENT-SERVICE

#Publish Application(Register with Eureka)

eureka.client.service-url.default-zone=http://localhost:8761/eureka

\*) RestController

@RestController

public class PaymentRestController {

@GetMapping("/pay")

public String showMsg() {

return "FROM PAYMENT SERVICE";

}

}

===============Execution Order===============================

1. Run Eureka Server

2. Run MS application

3. Goto Eureka Server and refresh

4. Click on instance link under status

5. Modify URL as: http://192.168.0.7:9009/pay

-----------------------------------------------------------------------------------

ClientType:- These client types are used to fetch any Microservice data from Eureka Server

using any service Id, it will get data of MS in ServiceInstance object that holds

details of (ServiceId, InstanceId, HOST, PORT, ....etc)

\*) Different Client Types are:

a) DiscoveryClient (Basic Client)

b) LoadBalancerClient\*\*\*

c) \*\*FeignClient | Abstract Client

=> In simple fetching MS details from Eureka using Client.

=> Then client type provide request details to RestTemplate to make HTTP call.

**Spring Cloud - MS Intra-Communication**

=> One MS (devloped and published) wants to communicate with another MS using

with the help of Eureka Server.

=> We must use any one client to get MS details from Eureka Server.

Possible Clients are:

a) DiscoveryClient (Legacy Client)

b)\*\* LoadBalancerClient (new client)

c)\*\* FeignClient (Abstract client)

=> By using above clients we can get ServiceInstance details of MS from Eureka Server

using ServiceId as input.

=> Once we get ServiceInstance data, then use HTTP Client : RestTemplate(C)

pass URL, MethodType, OutputType..etc, make HTTP Request to Provider application.

=> IP and PORT numbers may get modified/changed based on System and

Deployment Instance count. So, read those details from Eureka

using ServiceInstance.

=> Location and Identifier

URL = Protocol://IP:PORT/Path

URI = Protocol://IP:PORT

URL = URI + PATH

ex: http://localhost:9898/show (URL)

http://localhost:9898 (URI)

---------------Coding Steps using DC(DiscoveryClient)--------------------------------

1. Eureka Server

Name : SpringCloudDCEurekaServer

Dep : Eureka Server

=> At Starter class: @EnableEurekaServer

--application.properties--

server.port=8761

eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false

---------------------------------

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Provider Microservice

Name : SpringCloudDCCartService

Dep : Spring web, Eureka Discovery client

=> At Starter class: @EnableEurekaClient

---application.properties---

server.port=9900

#ServiceId

spring.application.name=CART-SERVICE

#Register with Eureka

eureka.client.service-url.default-zone=http://localhost:8761/eureka

-----------------------------

=> RestController class

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

@RestController

@RequestMapping("/cart")

public class CartRestController {

@GetMapping("/info")

public String getCartInfo() {

return "WELCOME TO CART DATA";

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3.\*\*\* Consumer Microservice

Name : SpringCloudDCPaymentSevice

Dep : Spring web, Eureka Discovery client

=> At Starter class: @EnableEurekaClient

---application.properties---

server.port=8989

spring.application.name=PAYMENT-SERVICE

eureka.client.service-url.default-zone=http://localhost:8761/eureka

---------------------------

=> \*\*\* REST CONSUMER CODE\*\*\*\*

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.cloud.client.ServiceInstance;

import org.springframework.cloud.client.discovery.DiscoveryClient;

import org.springframework.stereotype.Component;

import org.springframework.web.client.RestTemplate;

@Component

public class CartRestConsumer {

//autowire disocvery client

@Autowired

private DiscoveryClient client;

public String getCartInfo() {

// get ServiceInstance list using serviceId

List<ServiceInstance> siList = client.getInstances("CART-SERVICE");

//read manually one instace from index#0

ServiceInstance si = siList.get(0);

//Read URI and Add path that returns url

String url = si.getUri() +"/cart/info";

// create object for RestTemplate

RestTemplate rt = new RestTemplate();

//make HTTP call and get Reponse data

String response = rt.getForObject(url, String.class);

// return response back to Consumer App RestController

return response;

}

}

=> RestController class :

\*\* also create HAS-A Relation between RestController and RestConsumer

then call Consumer method to get data from Cart MS.

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import in.nareshit.raghu.consumer.CartRestConsumer;

@RestController

@RequestMapping("/payment")

public class PaymentRestController {

@Autowired

private CartRestConsumer consumer; //HAS-A

@GetMapping("/data")

public String getPayData() {

return "FROM PAYMENT ==> " + consumer.getCartInfo();

}

}

======== Execution Steps =====================

a. Eureka Server Starter class

b. Provider App Starter class (CartService)

c. Consumer App Starter class (PaymentService)

d. click on Payment Service link in Eureka

Ex: http://192.168.0.7:8989/actuator/info

e. Modify URL path as: /payment/data

<http://192.168.0.7:8989/payment/data>

**MongoDB**

https://docs.mongodb.com/v3.6/reference/command/

1. View all existed dbs

> show dbs

2. see current db

> db

3. Create our own db

> use <dbName>

> use nit

4. view all existed collections

> show collections

5. Create a collection and insert data.

[\*\* If we write any insert command then collection gets created automatically]

> db.<collectionName>.insert({key:val...})

> db.student.insert({"sid":10,"sname":"SAM","sfee":3200.0})

6. Display all Documents(JSON) from a Collection

> db.<collectionName>.find()

> db.<collectionName>.find().pretty()

> db.student.find()

> db.student.find().pretty()

> db.student.find({"sname":"SYED"}).pretty()

7. Delete one Document (JSON) from a Collection

> db.<collectionName>.deleteOne({k:v})

> db.<collectionName>.deleteMany({k:v})

> db.student.deleteOne({"sname":"SYED"})

8. Drop a Collection

> db.<collectionName>.drop()

> db.student.drop()

-------Task--------------------------------------------------------------

Q) Which database type is used for what case? SQL Db and NoSQL Db?

A)

---------------------------------------------------------------------

\*) Note: In Java one Object is equals to in Mognodb it is one JSON Document.

=========code=========================

1. Create Starter Project

Name : SpringBoot2MongoDbFirstApp

Dep : Spring Boot Mongodb, lombok

2. Model class

import org.springframework.data.annotation.Id;

import org.springframework.data.mongodb.core.mapping.Document;

import lombok.Data;

import lombok.NoArgsConstructor;

import lombok.NonNull;

import lombok.RequiredArgsConstructor;

@Data

@NoArgsConstructor

@RequiredArgsConstructor

@Document // Maps Model class objects as JSON Documents

public class Employee {

@Id // makes this variable as ID and auto-generated

private String id;

@NonNull

private Integer eid;

@NonNull

private String ename;

@NonNull

private Double esal;

}

3. Repository Interface

import org.springframework.data.mongodb.repository.MongoRepository;

public interface EmployeeRepository extends MongoRepository<Employee, String> {

}

4. Runner class

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

@Component

public class EmployeeTestRunner implements CommandLineRunner {

@Autowired

private EmployeeRepository repo;

@Override

public void run(String... args) throws Exception {

repo.save(new Employee(101,"A",5000.0));

repo.save(new Employee(102,"B",6000.0));

repo.save(new Employee(103,"C",7000.0));

repo.save(new Employee(104,"D",8000.0));

System.out.println("\_\_\_DONE\_\_\_\_");

}

}

5. application.properties

spring.data.mongodb.host=localhost

spring.data.mongodb.port=27017

spring.data.mongodb.database=nit

6. Start Mongo server

cmd> mongod

7. Run Main class

8. Open mongo client cmd> mongo

and execute below commands

> use nit

> show collections

> db.employee.find().pretty()

====================================================================

\*) If mongo server is not started and trying to run SpringBoot application

then ConnectException: Connection refused: no further information.

====================================================================

Spring Boot MongoDB - Database Operations

====================================================================

=> MongoRepository(I) given by 'spring-boot-starter-data-mongodb'.

=> For this one impl class is provided by data-mongo ie SimpleMongoRepository(C)

> ctrl+shift+T > mongorepository > dbl click on same

> select interface name and press F4 key

> click on 'SimpleMongoRepository' at type Hierarchy

=> One Proxy class is generated at runtime using SimpleMongoRepository

design which looks like (similier to this)

class Proxy45 extends SimpleMongoRepository<Employee,String> {

...//.....

}

=> we can get proxy details using Reflection detals

@Autowired

private EmployeeRepository repo;

and

System.out.println(repo.getClass().getName());

--------------------------------------------------------

CrudRepository<T, ID>

PagingAndSortingRepository<T, ID>

MongoRepository<T, ID>

\*)Note: No tables concept in MongoDB, So, no ddl-auto=\_\_\_ property in MongoDB.

Better use repo.deleteAll() in code.

a) save(Object):object

This method is used to perform either insert/update (based on ID)

and returns same object with ID effected/updated.

We can even provide ID also manually (or use some code for ID generation)

ie at application side we can pass ID(String-Hexa Decimal number).

--Example---

1. application.properties

spring.data.mongodb.host=localhost

spring.data.mongodb.port=27017

spring.data.mongodb.database=nit

spring.data.mongodb.username=nit

spring.data.mongodb.password=raghu

2. Model

import org.springframework.data.annotation.Id;

import org.springframework.data.mongodb.core.mapping.Document;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

import lombok.NonNull;

import lombok.RequiredArgsConstructor;

@Data

@NoArgsConstructor

@RequiredArgsConstructor

@AllArgsConstructor

@Document

public class Student {

@Id

private String id;

@NonNull

private Integer stdId;

@NonNull

private String stdName;

@NonNull

private Double stdFee;

}

3. Repository interface

import org.springframework.data.mongodb.repository.MongoRepository;

public interface StudentRepository

extends MongoRepository<Student, String> {

}

4. Runner class

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

@Component

public class StudentTestOneRunner implements CommandLineRunner {

@Autowired

private StudentRepository repo;

@Override

public void run(String... args) throws Exception {

repo.deleteAll(); // like ddl-auto-create

// this method returns same object with id generated

Student s = repo.save(new Student(101,"SAM",2500.0));// insert

repo.save(new Student(s.getId(),101,"SAM-A",2800.0));// update

//manually provide ID(allowed)

repo.save(new Student("A10102B54DEF",102,"SYED",3500.0));// insert

repo.save(new Student("A10102B54DE0",102,"SYED",3500.0));// insert

System.out.println(s.getId());

System.out.println("\_\_\_Done\_\_\_");

}

}

========================================================

\*)Note: we can even write Customization code for ID generator.

import java.util.UUID;

public class IdGenerator {

public static String getId() {

//JDK 1.5

return UUID.randomUUID().toString()

.replaceAll("-", "")

.substring(0, 8);

}

}

and Runner class code:

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

@Component

public class StudentTestOneRunner implements CommandLineRunner {

@Autowired

private StudentRepository repo;

@Override

public void run(String... args) throws Exception {

repo.deleteAll(); // like ddl-auto-create

// this method returns same object with id generated

Student s = repo.save(new Student(IdGenerator.getId(),101,"SAM",2500.0));// insert

repo.save(new Student(s.getId(),101,"SAM-A",2800.0));// update

//manually provide ID(allowed)

repo.save(new Student(IdGenerator.getId(),102,"SYED",3500.0));// insert

repo.save(new Student(IdGenerator.getId(),102,"SYED",3500.0));// insert

System.out.println(s.getId());

System.out.println("\_\_\_Done\_\_\_");

}

}

==============================================================================

Special Types : Spring Boot - Mongo DB

==============================================================================

3. HAS-A Type (non-Collection)

4. HAS-A Type Collection (List/Map)

-------------------------------------

3. HAS-A Type (non-Collection) : if two model classes are connected using HAS-A

relation then two objects are created for them, even linked with each other.

But, in mongo DB, they are stored as one JSON Document, in below format:

{

// parent JSON

"hasAvaiable" : {

// child JSOn

}

}

---code---

1. Model classes

package in.nareshit.raghu.model;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Vendor {

private Integer id;

private String code;

}

import org.springframework.data.annotation.Id;

import org.springframework.data.mongodb.core.mapping.Document;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@NoArgsConstructor

@AllArgsConstructor

@Document

public class Product {

@Id

private String id;

private Integer pid;

private String pcode;

private Double pcost;

private Vendor vob;

}

2. Repository Interface

import org.springframework.data.mongodb.repository.MongoRepository;

public interface ProductRepositry extends MongoRepository<Product, String> {

}

3. Runner class

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

@Component

public class TestDataRunner implements CommandLineRunner {

@Autowired

private ProductRepositry repo;

@Override

public void run(String... args) throws Exception {

Vendor vob = new Vendor(110, "VA");

Product pob = new Product("A001", 101, "PEN", 50.0, vob);

repo.save(pob);

System.out.println("\_\_\_DONE\_\_");

}

}

4. application.propreties

spring.data.mongodb.host=localhost

spring.data.mongodb.port=27017

spring.data.mongodb.database=nit

spring.data.mongodb.username=nit

spring.data.mongodb.password=raghu

5. Start Mongo Server and at Mongo Client:

> db.product.find().pretty();

--output---

{

"\_id" : "A001",

"pid" : 101,

"pcode" : "PEN",

"pcost" : 50,

"vob" : {

"\_id" : 110,

"code" : "VA"

},

"\_class" : "in.nareshit.raghu.model.Product"

}

==============================================================

4. HAS-A Type Collection (List/Map)

JSON Format: for List Child Type

{

//parent

"hasAvariable" : [ {//child-obj-1}, {//child-obj-2}, {}, {},....]

}

JSON Format: for Map Child Type

{

//parent

"hasAvariable" : { key1:{//child-obj-1}, key2:{//child-obj-2}, key3:{}, ....}

}

---code---

1. Model classes

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Info {

private Integer id;

private String data;

}

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Vendor {

private Integer id;

private String code;

}

import java.util.List;

import java.util.Map;

import org.springframework.data.annotation.Id;

import org.springframework.data.mongodb.core.mapping.Document;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@NoArgsConstructor

@AllArgsConstructor

@Document

public class Product {

@Id

private String id;

private Integer pid;

private String pcode;

private Double pcost;

private List<Vendor> vob;

private Map<String,Info> infobs;

}

2) Repository Interface

package in.nareshit.raghu.repo;

import org.springframework.data.mongodb.repository.MongoRepository;

import in.nareshit.raghu.model.Product;

public interface ProductRepositry extends MongoRepository<Product, String> {

}

3) Runner class:

import java.util.List;

import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

@Component

public class TestDataRunner implements CommandLineRunner {

@Autowired

private ProductRepositry repo;

@Override

public void run(String... args) throws Exception {

repo.deleteAll();

Product pob = new Product("A001", 101, "PEN", 50.0,

List.of(

new Vendor(100, "VA"),

new Vendor(101, "VB"),

new Vendor(102, "VC")

),

Map.of(

"I1", new Info(55,"OK"),

"I2", new Info(56,"ACTIVE"),

"I3", new Info(57,"DONE")

)

);

repo.save(pob);

System.out.println("\_\_\_DONE\_\_");

}

}

4) MongoDB Command and output:

> db.product.find().pretty();

{

"\_id" : "A001",

"pid" : 101,

"pcode" : "PEN",

"pcost" : 50,

"vob" : [

{

"\_id" : 100,

"code" : "VA"

},

{

"\_id" : 101,

"code" : "VB"

},

{

"\_id" : 102,

"code" : "VC"

}

],

"infobs" : {

"I2" : {

"\_id" : 56,

"data" : "ACTIVE"

},

"I1" : {

"\_id" : 55,

"data" : "OK"

},

"I3" : {

"\_id" : 57,

"data" : "DONE"

}

},

"\_class" : "in.nareshit.raghu.model.Product"

}

============================ @Query =======================================

-code--

1. Model

import org.springframework.data.annotation.Id;

import org.springframework.data.mongodb.core.mapping.Document;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@NoArgsConstructor

@AllArgsConstructor

@Document

public class Book {

@Id

private Integer id;

private String bname;

private Integer pages;

private String author;

private String btype;

}

2. Repository interface

import java.util.List;

import java.util.Optional;

import org.springframework.data.mongodb.repository.MongoRepository;

import org.springframework.data.mongodb.repository.Query;

public interface BookRepository extends MongoRepository<Book, Integer> {

//---custom query methods---------

//SQL: SELECT \* FROM BOOK WHERE ID=?

@Query("{id : ?0 }")

Optional<Book> getBookById(Integer id);

@Query("{ pages : { $lt: ?0 } }") // where pages<?

//@Query("{ pages : { $gte: ?0 } }") // where pages>=?

//@Query("{ pages : ?0 }") // where pages=?

List<Book> getBooksByPages(Integer pages);

// where author = ?

@Query("{ author : ?0}")

List<Book> getBooksByAuthor(String author);

// where author = ? and btype =?

@Query("{ author : ?0 , btype: ?1}")

List<Book> getBooksByAuthorAndType(String author,String btype);

// where author = ? or btype =?

//@Query("{$and : [{ author : ?0 } , { btype: ?1}]}")

@Query("{$or : [{ author : ?0 } , { btype: ?1}]}")

List<Book> getBooksByAuthorOrType(String author,String btype);

//SQL: select count(\*) from book where author=?

@Query(value = "{ author : ?0}", count = true)

Integer getBooksCountByAuthor(String author);

//Sorting

@Query(value = "{ author : ?0}", sort ="{ bname:1 }") //ASC

//@Query(value = "{ author : ?0}", sort ="{ bname:-1 }") //DESC

List<Book> getBooksByAuthorortByBname(String author);

}

3. Runner class for Insert

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

//@Component

public class BookInsertRunner implements CommandLineRunner {

@Autowired

private BookRepository repo;

@Override

public void run(String... args) throws Exception {

repo.deleteAll();

repo.save(new Book(100, "Core Java", 200, "SAM", "BASICS"));

repo.save(new Book(101, "Adv Java", 300, "SAM", "WEB"));

repo.save(new Book(102, "SPRING", 480, "SYED", "WEB"));

repo.save(new Book(103, "ANGULAR", 260, "SYED", "UI"));

repo.save(new Book(104, "HTML CSS", 100, "RAM", "UI"));

repo.save(new Book(105, "C++", 180, "SAM", "BASICS"));

repo.save(new Book(106, "C", 100, "RAM", "BASICS"));

repo.save(new Book(107, "SPRING BOOT",850, "RAM", "WEB"));

repo.save(new Book(108, "JQuery", 120, "SYED", "UI"));

repo.save(new Book(109, "DP", 280, "SAM", "BASICS"));

System.out.println("\_\_\_DONE\_\_\_");

}

}

4. Runner class for test

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

@Component

public class TestQueryRunner implements CommandLineRunner {

@Autowired

private BookRepository repo;

@Override

public void run(String... args) throws Exception {

/\*

Optional<Book> opt = repo.getBookById(100);

if(opt.isPresent()) System.out.println(opt.get());

else System.out.println("DATA NOT FOUND");

\*/

//repo.getBooksByAuthor("SAM").forEach(System.out::println);

//repo.getBooksByPages(400).forEach(System.out::println);

//repo.getBooksByAuthorAndType("SAM","BASICS").forEach(System.out::println);

//repo.getBooksByAuthorOrType("SAM","BASICS").forEach(System.out::println);

//Integer count = repo.getBooksCountByAuthor("SAM"); System.out.println(count);

repo.getBooksByAuthorortByBname("SAM").forEach(System.out::println);

}

}