Day32 Custom Queries using Mongo Repository.

1. Micro-service Concepts.
2. Service Discovery (Eureka/Consul/Zoo Keeper)
3. Load Balancing, Rate Limiting
4. Features of Micro-service.

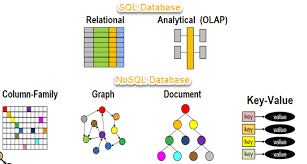
Day31 – No-SQL Database – MongoDB.

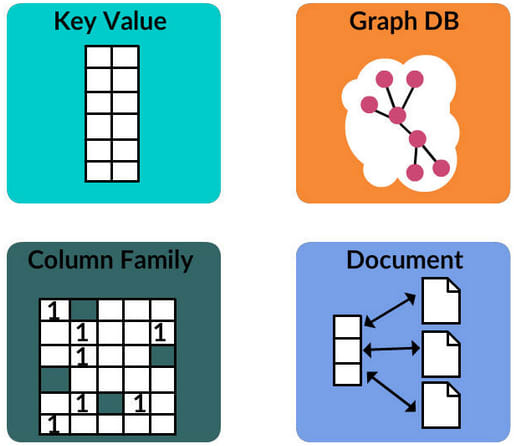
No-SQL DBs are highly dynamic in nature. (Schema less/ flexible schema design)

Mongo DB – Document based Database.

Types of No-SQL DBs

1. MongoDB
2. Redis
3. Cassandra
4. Neo4j





Most Social Media sites like facebook, x, linkedin, Instagram ---

POST – The Person who created/shared it. The content (Plain Text, PDF, JPEG, GIF, Video, URL.) -- Footer (Like button, comment button, share button, etc.,)

Creating Spring Boot application with Spring Mongo Data dependency. (will add mongo drivers & mongo related configs)

CRUD operations using MongoRepository.

Micro-service.

ARS – Airline Reservation System – v2 (monolith version) – Single Spring boot application with all the modules like passengers, reservations, payments, ratings, airlines, flights, etc.,)

Challenges in Monolith

1. Difficult to Test.
2. Adding new features might break existing functionality.
3. Maintaining the codebase.
4. Scaling the application.
5. Bulkier deployable unit.
6. Trying new Tools/DBs involves lot of work.

Micro-service.

Each microservice will have single responsibility.

Netflix OSS –

1. Service Discovery. (Connects all the micro-service) [Eureka/Consul/ Zoo keeper]
2. API Gateway. ( Multiple spring boot app – each will run in it’s own port number) – Routing/ Authentication & Authorization/ rate-limiting, etc.,]

Spring – Cloud [Micro-service]

Ars-v3

1. Eureka-server (Discovery Server)

Eureka-server dependency.

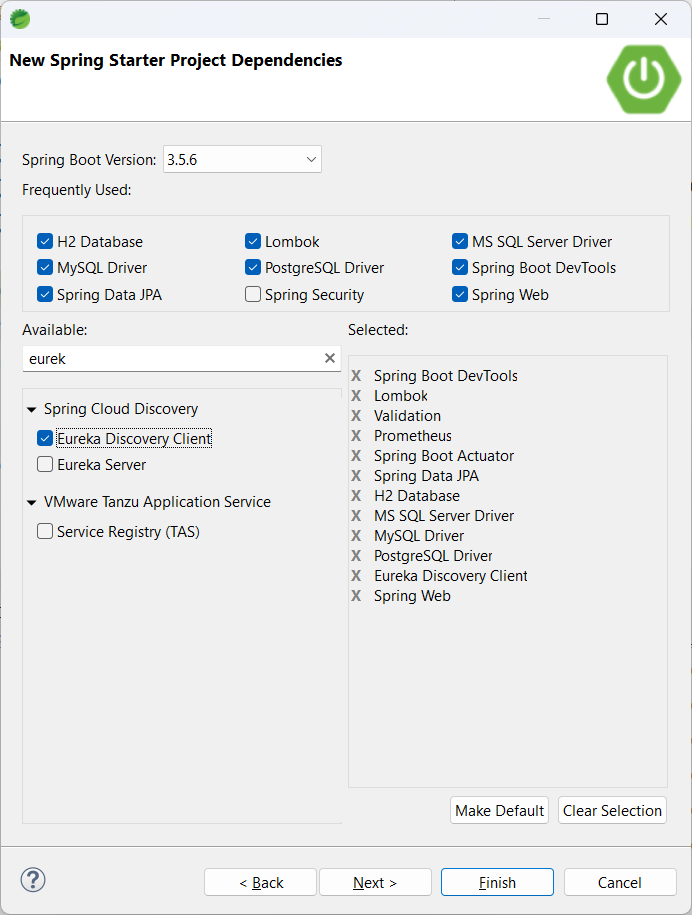
# default port for Eureka server

server.port=8761

eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false

1. Passenger-service



Package Naming Convention

1. Controller
2. Model
3. Repo
4. Service
5. Config
6. Util
7. Enums
8. Dto (Data Transfer Objects)
9. exception

ArrayList, LinkedHashMap, TreeSet

List

Map

Set

Airline-service

Id

Name

url

helpline

@Document(collection = "customers")

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Customer {

@Id

private String id;

private String name;

private String city;

private int age;

private String email;

}

Your base repository:

public interface CustomerRepository extends MongoRepository<Customer, String> {

}

**🔍 Custom Query Examples**

**1. Find by field (derived query method)**

List<Customer> findByCity(String city);

Spring Data builds the query automatically:

{ "city": "Delhi" }

**2. Custom @Query annotation**

@Query("{ 'age': { $gt: ?0 } }")

List<Customer> findCustomersOlderThan(int age);

Equivalent MongoDB query:

{ "age": { "$gt": 30 } }

**3. Multiple conditions**

@Query("{ 'city': ?0, 'age': { $lt: ?1 } }")

List<Customer> findByCityAndYoungerThan(String city, int age);

{ "city": "Bangalore", "age": { "$lt": 25 } }

**4. Using Regular Expressions**

@Query("{ 'name': { $regex: ?0, $options: 'i' } }")

List<Customer> findByNameLike(String namePattern);

This performs a case-insensitive search on the name field.

**5. Custom fields projection**

Only fetch name and email:

@Query(value = "{ 'city': ?0 }", fields = "{ 'name': 1, 'email': 1, '\_id': 0 }")

List<Customer> findNamesAndEmailsByCity(String city);

**6. Delete with custom query**

@Query(value="{ 'city' : ?0 }", delete = true)

void deleteByCity(String city);

**7. Using MongoTemplate (for complex logic)**

@Autowired

private MongoTemplate mongoTemplate;

public List<Customer> getSeniorCustomers(String city, int minAge) {

Query query = new Query();

query.addCriteria(Criteria.where("city").is(city).and("age").gte(minAge));

return mongoTemplate.find(query, Customer.class);

}