Redis OM Spring

❖ Introduction

- Redis OM Spring extends Spring Data Redis to take full advantage of Redis Stack, providing powerful repository and custom object-mapping abstractions. It seamlessly integrates with the Spring ecosystem while leveraging Redis's advanced capabilities for document storage, search, and vector operations.
- **Key Features:**
  - @Document annotation for JSON document mapping
  - Enhanced @RedisHash with native search capabilities
  - Automatic repository implementation with RedisDocumentRepository
  - Declarative search indexes via @Indexed and @Searchable
  - Entity Streams for fluent query building
  - Vector similarity search support
  - Integration with Spring Boot auto-configuration

❖ Setup using Spring Boot

- Prerequisites
  - **Required Components:**
    - Java 17 or higher
    - Spring Boot 3.x (requires Spring Data Redis 3.4.1+)
    - Redis Stack (Redis with RedisJSON and RediSearch modules)
  - Starting Redis Stack:

```
# Using Docker
docker run -p 6379:6379 -p 8001:8001 redis/redis-stack

# Using Docker Compose
docker compose up
```

- Maven Configuration
  - **Add Dependencies:**

```
<dependencies>
    <!-- Redis OM Spring -->
    <dependency>
        <groupId>com.redis.om</groupId>
        <artifactId>redis-om-spring</artifactId>
        <version>1.0.0-RC.1</version>
    </dependency>

    <!-- Redis OM Spring AI (optional) -->
    <dependency>
        <groupId>com.redis.om</groupId>
        <artifactId>redis-om-spring-ai</artifactId>
        <version>1.0.0-RC.1</version>
    </dependency>

    <!-- Spring Boot Starter Web -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!-- Lombok (optional, for cleaner code) -->
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
</dependencies>
```

  - Maven Compiler Plugin Configuration:

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>${maven-compiler-plugin.version}</version>
    <configuration>
        <annotationProcessorPaths>
            <path>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-configuration-
processor</artifactId>
                <version>3.3.0</version>
            </path>
            <path>
                <groupId>org.projectlombok</groupId>
                <artifactId>lombok</artifactId>
                <version>${lombok.version}</version>
            </path>
```

```
            <path>
                <groupId>com.redis.om</groupId>
                <artifactId>redis-om-spring</artifactId>
                <version>1.0.0-RC.1</version>
            </path>
        </annotationProcessorPaths>
    </configuration>
</plugin>
```

- Gradle Configuration

```
repositories {
    mavenCentral()
    maven {
        url
'https://s01.oss.sonatype.org/content/repositories/snapshot
s/'
    }
}

ext {
    redisOmVersion = '1.0.0-RC.1'
}

dependencies {
    implementation "com.redis.om:redis-om-
spring:$redisOmVersion"
    implementation "com.redis.om:redis-om-spring-
ai:$redisOmVersion"
    annotationProcessor "com.redis.om:redis-om-
spring:$redisOmVersion"

    implementation 'org.springframework.boot:spring-boot-
starter-web'
    compileOnly 'org.projectlombok:lombok'
    annotationProcessor 'org.projectlombok:lombok'
}
```

➢ Application Configuration

- Connection Configuration (application.properties):

```
# Redis connection settings
spring.data.redis.host=localhost
spring.data.redis.port=6379
spring.data.redis.password=your_password
spring.data.redis.username=default

# SSL configuration (if needed)
spring.data.redis.ssl.enabled=true
```

```
# Connection pool settings
spring.data.redis.jedis.pool.max-active=8
spring.data.redis.jedis.pool.max-idle=8
spring.data.redis.jedis.pool.min-idle=0
```

- YAML Configuration (application.yml):

```
spring:
  data:
    redis:
      host: localhost
      port: 6379
      password: your_password
      username: default
      ssl:
        enabled: false
      jedis:
        pool:
          max-active: 8
          max-idle: 8
          min-idle: 0
```

- Main Application Class:

```
@EnableRedisDocumentRepositories(basePackages =
"com.example.repositories")
@SpringBootApplication
public class RedisOmApplication {
    public static void main(String[] args) {
        SpringApplication.run(RedisOmApplication.class,
args);
    }
}
```

❖ Defining Entities with Annotations

➢ Basic Document Entity

```
package com.example.model;

import java.util.Set;
import org.springframework.data.annotation.Id;
import org.springframework.data.geo.Point;
import com.redis.om.spring.annotations.Document;
import com.redis.om.spring.annotations.Indexed;
import com.redis.om.spring.annotations.Searchable;
import lombok.*;

@Data
@NoArgsConstructor
@RequiredArgsConstructor(staticName = "of")
```

```
@AllArgsConstructor(access = AccessLevel.PROTECTED)
@Document
public class Person {
    @Id
    @Indexed
    private String id;                      // Auto-generated
ULID

    @Indexed
    @NonNull
    private String firstName;               // Tag index for
exact matching

    @Indexed
    @NonNull
    private String lastName;                // Tag index for
exact matching

    @Indexed
    @NonNull
    private Integer age;                    // Numeric index for
range queries

    @Searchable
    @NonNull
    private String personalStatement;       // Full-text search
index

    @Indexed
    @NonNull
    private Point homeLoc;                  // Geospatial index

    @Indexed
    @NonNull
    private Address address;                // Nested object
indexing

    @Indexed
    @NonNull
    private Set<String> skills;             // Tag index on
array elements
}
```

- **Nested Object Definitions**

```
package com.example.model;

import com.redis.om.spring.annotations.Indexed;
import com.redis.om.spring.annotations.Searchable;
import lombok.Data;
import lombok.NonNull;
```

```
import lombok.RequiredArgsConstructor;

@Data
@RequiredArgsConstructor(staticName = "of")
public class Address {
    @NonNull
    @Indexed
    private String houseNumber;           // Tag index

    @NonNull
    @Searchable(nostem = true)
    private String street;                // Full-text with no
stemming

    @NonNull
    @Indexed
    private String city;                  // Tag index

    @NonNull
    @Indexed
    private String state;                 // Tag index

    @NonNull
    @Indexed
    private String postalCode;            // Tag index

    @NonNull
    @Indexed
    private String country;               // Tag index
}
```

- **Advanced Entity with Vector Search**

```
@Data
@NoArgsConstructor
@RequiredArgsConstructor(staticName = "of")
@Document
public class Product {
    @Id
    private String id;

    @Indexed
    @NonNull
    private String name;

    @Searchable
    @NonNull
    private String description;

    @Indexed
    @NonNull
```

```
    private Double price;

    @Indexed
    @NonNull
    private Set<String> categories;

    @Indexed
    @NonNull
    private Point location;

    // Vector field for similarity search
    @Vectorize(destination = "descriptionEmbedding")
    private String description;

    // The actual vector field (populated automatically)
    private float[] descriptionEmbedding;

    @Indexed
    private Date createdAt;

    @Indexed
    private Boolean inStock;
}
```

- **Annotation Details**
  - **@Document:**
    - Marks class as Redis JSON document
    - Automatically creates search index
    - Supports prefix configuration: @Document(prefixes = "product:")
  - **@Id:**
    - Primary key field
    - Auto-generates ULID if not provided
    - Always indexed automatically
  - **@Indexed:**
    - Creates search index based on field type
    - Tag index for strings and enums
    - Numeric index for numbers
    - Geo index for Point types
    - Tag index for collections
  - **@Searchable:**
    - Creates full-text search index

- Supports language-specific stemming
- Options: nostem, weight, phonetic

- **@Vectorize:**
  - Automatically generates vector embeddings
  - Requires Redis OM Spring AI module
  - Supports various embedding providers

❖ Repositories and CRUD

➢ Basic Repository Interface

```
package com.example.repository;

import com.example.model.Person;
import com.redis.om.spring.repository.RedisDocumentRepository;

public interface PersonRepository extends
RedisDocumentRepository<Person, String> {
    // Inherits all CRUD operations
    // Custom query methods can be added here
}
```

➢ CRUD Operations

```
@Service
public class PersonService {

    @Autowired
    private PersonRepository personRepository;

    public void demonstrateCrudOperations() {
        // CREATE
        Person person = Person.of(
            "John",
            "Doe",
            30,
            "Software developer passionate about Redis",
            new Point(-122.4194, 37.7749),
            Address.of("123", "Main St", "San Francisco",
"CA", "94105", "USA"),
            Set.of("Java", "Spring", "Redis")
        );

        String id = personRepository.save(person);
        System.out.println("Saved person with ID: " + id);
```

```
        // READ
        Optional<Person> found =
personRepository.findById(id);
        if (found.isPresent()) {
            System.out.println("Found person: " +
found.get().getFirstName());
        }

        // READ ALL
        Iterable<Person> allPeople =
personRepository.findAll();
        allPeople.forEach(p ->
System.out.println(p.getFirstName()));

        // UPDATE
        person.setAge(31);
        personRepository.save(person); // Updates existing
record

        // DELETE
        personRepository.deleteById(id);

        // BULK OPERATIONS
        List<Person> people = Arrays.asList(
            Person.of("Alice", "Smith", 25, "Data scientist",
location1, address1, skills1),
            Person.of("Bob", "Johnson", 35, "Product manager",
location2, address2, skills2)
        );

        personRepository.saveAll(people);
        personRepository.deleteAll(); // Clears all data

        // COUNT
        long count = personRepository.count();
        System.out.println("Total people: " + count);
    }
}
```

- **Repository with Custom Query Methods**

```
public interface PersonRepository extends
RedisDocumentRepository<Person, String> {

    // Find by single property
    Optional<Person> findOneByFirstName(String firstName);
    Iterable<Person> findByLastName(String lastName);

    // Find by multiple properties (AND condition)
    Iterable<Person> findByFirstNameAndLastName(String
firstName, String lastName);
```

```
    // Numeric range queries
    Iterable<Person> findByAge(int age);
    Iterable<Person> findByAgeBetween(int minAge, int maxAge);
    Iterable<Person> findByAgeGreaterThan(int age);
    Iterable<Person> findByAgeLessThan(int age);

    // String pattern matching
    Iterable<Person> findByFirstNameStartingWith(String
prefix);
    Iterable<Person> findByLastNameEndingWith(String suffix);
    Iterable<Person> findByFirstNameContaining(String
substring);

    // Collection queries
    Iterable<Person> findBySkills(Set<String> skills);
    Iterable<Person> findBySkillsContaining(String skill);

    // Geospatial queries
    Iterable<Person> findByHomeLocNear(Point point, Distance
distance);

    // Nested object queries (using underscore notation)
    Iterable<Person> findByAddress_City(String city);
    Iterable<Person> findByAddress_StateAndAddress_City(String
state, String city);

    // Full-text search (use 'search' prefix)
    Iterable<Person> searchByPersonalStatement(String text);

    // Custom queries with @Query annotation
    @Query("@age:[25 35] @skills:{Java}")
    Iterable<Person> findJavaDevelopersInAgeRange();

    @Query("@personalStatement:($text)")
    Iterable<Person> searchPersonalStatement(@Param("text")
String text);

    // Pagination support
    Page<Person> findByAgeGreaterThan(int age, Pageable
pageable);

    // Sorting support
    Iterable<Person> findBySkillsOrderByAgeAsc(String skill);
    Iterable<Person>
findByAddress_CityOrderByLastNameDescFirstNameAsc(String
city);
}
```

❖ Query Building

- **Method Name Derivation**
  - Redis OM Spring supports query derivation from method names following Spring Data conventions:
  - **Supported Keywords:**
    - findBy, readBy, queryBy, countBy, deleteBy
    - And, Or
    - Between, LessThan, GreaterThan, LessThanEqual, GreaterThanEqual
    - After, Before (for dates)
    - IsNull, IsNotNull, True, False
    - StartingWith, EndingWith, Containing, NotContaining
    - In, NotIn
    - Near (for geospatial)
    - OrderBy with Asc, Desc

- **Custom Queries with @Query**

```
public interface PersonRepository extends
RedisDocumentRepository<Person, String> {

    // Exact field matching
    @Query("@firstName:{$name}")
    Iterable<Person> findByExactFirstName(@Param("name")
String name);

    // Numeric range
    @Query("@age:[($min) ($max)]")
    Iterable<Person> findByAgeRange(@Param("min") int min,
@Param("max") int max);

    // Full-text search with boosting
    @Query("@personalStatement:($keywords)^2.0")
    Iterable<Person> searchWithBoosting(@Param("keywords")
String keywords);

    // Complex query with multiple conditions
    @Query("(@age:[25 40] @skills:{Java|Spring})
(@address_city:{San Francisco|New York})")
    Iterable<Person> findQualifiedDevelopersInMajorCities();

    // Geospatial query
    @Query("@homeLoc:[$lon $lat $radius mi]")
    Iterable<Person> findWithinRadius(
        @Param("lon") double longitude,
```

```
            @Param("lat") double latitude,
            @Param("radius") double radiusMiles
    );

    // Tag search with wildcards
    @Query("@skills:{program*}")
    Iterable<Person> findBySkillPattern();

    // Sorting and limiting
    @Query(value = "@age:[25 40]", sort = "@lastName", limit =
10)
    Iterable<Person> findTop10YoungAdultsByLastName();
}
```

- **Pagination and Sorting**

```
@Service
public class PersonService {

    @Autowired
    private PersonRepository personRepository;

    public void demonstratePaginationAndSorting() {
        // Create Pageable request
        Pageable pageable = PageRequest.of(
            0,                              // page number (0-
based)
            10,                             // page size
            Sort.by("lastName").ascending() // sorting
        );

        // Find with pagination
        Page<Person> page =
personRepository.findByAgeGreaterThan(25, pageable);

        System.out.println("Total elements: " +
page.getTotalElements());
        System.out.println("Total pages: " +
page.getTotalPages());
        System.out.println("Current page: " +
page.getNumber());
        System.out.println("Page size: " + page.getSize());

        // Process results
        page.getContent().forEach(person ->
            System.out.println(person.getFirstName() + " " +
person.getLastName())
        );

        // Complex sorting
        Sort complexSort = Sort.by(
```

```
            Sort.Order.desc("age"),
            Sort.Order.asc("lastName"),
            Sort.Order.asc("firstName")
        );

        Pageable complexPageable = PageRequest.of(0, 20,
complexSort);
        Page<Person> sortedPage =
personRepository.findAll(complexPageable);
    }
}
```

❖ Integration with Spring Data

➢ Configuration Classes

```
@Configuration
@EnableRedisDocumentRepositories(
    basePackages = "com.example.repositories",
    repositoryImplementationPostfix = "Impl"
)
@EnableConfigurationProperties
public class RedisConfiguration {

    @Bean
    @Primary
    public RedisConnectionFactory redisConnectionFactory() {
        JedisConnectionFactory factory = new
JedisConnectionFactory();
        factory.setHostName("localhost");
        factory.setPort(6379);
        factory.setUsePool(true);
        return factory;
    }

    @Bean
    public RedisTemplate<String, Object>
redisTemplate(RedisConnectionFactory connectionFactory) {
        RedisTemplate<String, Object> template = new
RedisTemplate<>();
        template.setConnectionFactory(connectionFactory);
        template.setKeySerializer(new
StringRedisSerializer());
        template.setValueSerializer(new
GenericJackson2JsonRedisSerializer());
        return template;
    }
}
```

➢ **Integration with Spring Boot Auto-Configuration**

```
@SpringBootApplication
@EnableRedisDocumentRepositories(basePackages =
"com.example.*")
public class Application {

    @Bean
    CommandLineRunner loadTestData(PersonRepository
repository) {
        return args -> {
            // Clean existing data
            repository.deleteAll();

            // Load sample data
            List<Person> people = createSamplePeople();
            repository.saveAll(people);

            System.out.println("Loaded " + repository.count()
+ " people");
        };
    }

    private List<Person> createSamplePeople() {
        return Arrays.asList(
            Person.of("John", "Doe", 30,
                "Software engineer with Redis expertise",
                new Point(-122.4194, 37.7749),
                Address.of("123", "Market St", "San
Francisco", "CA", "94105", "USA"),
                Set.of("Java", "Spring", "Redis",
"Microservices")),

            Person.of("Jane", "Smith", 28,
                "Full-stack developer passionate about modern
web technologies",
                new Point(-74.0060, 40.7128),
                Address.of("456", "Broadway", "New York",
"NY", "10013", "USA"),
                Set.of("JavaScript", "React", "Node.js",
"MongoDB"))
        );
    }

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

➢ Health Checks and Monitoring

```
@Component
public class RedisHealthIndicator implements HealthIndicator {

    @Autowired
    private RedisConnectionFactory connectionFactory;

    @Override
    public Health health() {
        try {
            RedisConnection connection =
connectionFactory.getConnection();
            if (connection != null) {
                connection.ping();
                connection.close();
                return Health.up()
                    .withDetail("redis", "Available")
                    .build();
            }
        } catch (Exception e) {
            return Health.down()
                .withDetail("redis", "Not available")
                .withException(e)
                .build();
        }
        return Health.down()
            .withDetail("redis", "Connection is null")
            .build();
    }
}
```

❖ Entity Streams

➢ Basic Entity Stream Usage

```
@Service
public class PersonService {

    @Autowired
    private EntityStream entityStream;

    public List<Person> findAllPeople() {
        return entityStream
            .of(Person.class)
            .collect(Collectors.toList());
```

```
    }

    public List<Person> findByAgeRange(int minAge, int maxAge)
{
        return entityStream
            .of(Person.class)
            .filter(Person$.AGE.between(minAge, maxAge))
            .sorted(Person$.AGE, SortOrder.ASC)
            .collect(Collectors.toList());
    }

    public List<Person> findQualifiedDevelopers() {
        return entityStream
            .of(Person.class)
            .filter(Person$.AGE.greaterThanOrEqualTo(25))
            .filter(Person$.SKILLS.contains("Java"))
            .filter(Person$.ADDRESS_CITY.eq("San Francisco"))
            .sorted(Person$.LAST_NAME, SortOrder.ASC)
            .limit(10)
            .collect(Collectors.toList());
    }
}
```

- **Advanced Stream Operations**

```
@Service
public class AdvancedPersonService {

    @Autowired
    private EntityStream entityStream;

    // Complex filtering with multiple conditions
    public List<Person> findSeniorDevelopersInTechHubs() {
        Set<String> techCities = Set.of("San Francisco",
"Seattle", "Austin", "New York");
        Set<String> seniorSkills = Set.of("Architecture",
"Leadership", "Mentoring");

        return entityStream
            .of(Person.class)
            .filter(Person$.AGE.greaterThan(35))
            .filter(Person$.ADDRESS_CITY.in(techCities))
            .filter(Person$.SKILLS.containsAny(seniorSkills))
            .sorted(Person$.AGE, SortOrder.DESC)
            .collect(Collectors.toList());
    }

    // Geospatial filtering
    public List<Person> findPeopleNearLocation(double lat,
double lon, double radiusMiles) {
        Point center = new Point(lon, lat);
```

```
        Distance radius = new Distance(radiusMiles,
Metrics.MILES);

        return entityStream
            .of(Person.class)
            .filter(Person$.HOME_LOC.near(center, radius))
            .sorted(Person$.LAST_NAME, SortOrder.ASC)
            .collect(Collectors.toList());
    }

    // Full-text search
    public List<Person> searchByDescription(String keywords) {
        return entityStream
            .of(Person.class)

.filter(Person$.PERSONAL_STATEMENT.matches(keywords))
            .collect(Collectors.toList());
    }

    // Aggregation operations
    public Map<String, Long> getSkillDistribution() {
        return entityStream
            .of(Person.class)
            .map(Person$.SKILLS)
            .flatMap(skills -> skills.stream())
            .collect(Collectors.groupingBy(
                skill -> skill,
                Collectors.counting()
            ));
    }

    // Statistical operations
    public OptionalDouble getAverageAge() {
        return entityStream
            .of(Person.class)
            .mapToInt(Person$.AGE)
            .average();
    }

    public Map<String, Double> getAverageAgeByCity() {
        return entityStream
            .of(Person.class)
            .collect(Collectors.groupingBy(
                person -> person.getAddress().getCity(),
                Collectors.averagingInt(Person::getAge)
            ));
    }
}
```

- **Stream with Projections**

```
@Service
public class ProjectionService {

    @Autowired
    private EntityStream entityStream;

    // Project to specific fields only
    public List<String> getAllNames() {
        return entityStream
            .of(Person.class)
            .map(person -> person.getFirstName() + " " +
person.getLastName())
            .collect(Collectors.toList());
    }

    // Create custom DTOs
    public List<PersonSummary> getPersonSummaries() {
        return entityStream
            .of(Person.class)
            .map(person -> new PersonSummary(
                person.getId(),
                person.getFirstName() + " " +
person.getLastName(),
                person.getAge(),
                person.getAddress().getCity()
            ))
            .collect(Collectors.toList());
    }

    // Group by and aggregate
    public Map<String, List<String>> groupPeopleByCity() {
        return entityStream
            .of(Person.class)
            .collect(Collectors.groupingBy(
                person -> person.getAddress().getCity(),
                Collectors.mapping(
                    person -> person.getFirstName() + " " +
person.getLastName(),
                    Collectors.toList()
                )
            ));
    }
}

// DTO for projections
@Data
@AllArgsConstructor
public class PersonSummary {
    private String id;
    private String fullName;
    private Integer age;
```

```
    private String city;
}
```

❖ Advanced Features

➢ Vector Similarity Search

```
@Document
@Data
@NoArgsConstructor
@RequiredArgsConstructor(staticName = "of")
public class Document {
    @Id
    private String id;

    @Indexed
    @NonNull
    private String title;

    @Searchable
    @NonNull
    private String content;

    @Vectorize(destination = "contentEmbedding", embedder =
"openai")
    private String content;

    // Vector field (populated automatically)
    private float[] contentEmbedding;

    @Indexed
    private Set<String> tags;
}

@Service
public class DocumentSearchService {

    @Autowired
    private DocumentRepository documentRepository;

    @Autowired
    private EntityStream entityStream;

    // Vector similarity search
    public List<Document> findSimilarDocuments(String
queryText, int limit) {
        return
documentRepository.findSimilarByContentEmbedding(
            queryText,      // Will be automatically
vectorized
```

```
            limit,          // Number of results
            0.7            // Minimum similarity threshold
        );
    }

    // Hybrid search (vector + traditional filters)
    public List<Document> findSimilarDocumentsWithFilters(
            String queryText,
            Set<String> requiredTags,
            int limit) {

        return entityStream
            .of(Document.class)
            .filter(Document$.TAGS.containsAll(requiredTags))
            .vectorSearch(Document$.CONTENT_EMBEDDING,
queryText)
            .limit(limit)
            .collect(Collectors.toList());
    }
}
```

- **Bloom Filters**

```
@Document
@Data
public class User {
    @Id
    private String id;

    @Indexed
    private String email;

    @Bloom(name = "emails", capacity = 100000, errorRate =
0.01)
    private String email;

    @Indexed
    private String username;
}

@Service
public class UserService {

    @Autowired
    private UserRepository userRepository;

    // Check if email might exist (fast probabilistic check)
    public boolean mightEmailExist(String email) {
        return userRepository.mightContain("emails", email);
    }
```

```
    // Add email to bloom filter
    public void addEmailToBloom(String email) {
        userRepository.add("emails", email);
    }
}
```

➢ Custom Repository Implementation

```
// Custom repository interface
public interface PersonRepositoryCustom {
    List<Person> findComplexQuery(String criteria);
    void bulkUpdateAges(Map<String, Integer> updates);
}

// Implementation
@Component
public class PersonRepositoryImpl implements
PersonRepositoryCustom {

    @Autowired
    private RedisTemplate<String, Object> redisTemplate;

    @Autowired
    private EntityStream entityStream;

    @Override
    public List<Person> findComplexQuery(String criteria) {
        // Custom implementation using native Redis commands
        // or complex EntityStream operations
        return entityStream
            .of(Person.class)
            .filter(/* complex filter logic */)
            .collect(Collectors.toList());
    }

    @Override
    public void bulkUpdateAges(Map<String, Integer> updates) {
        updates.forEach((id, newAge) -> {
            Optional<Person> person = findById(id);
            if (person.isPresent()) {
                person.get().setAge(newAge);
                save(person.get());
            }
        });
    }
}

// Extended repository interface
public interface PersonRepository extends
    RedisDocumentRepository<Person, String>,
    PersonRepositoryCustom {
```

```
    // Combines generated and custom methods
}
```

This comprehensive guide covers all major aspects of Redis OM Spring, from basic setup to advanced features like vector similarity search and custom repository implementations.