Innovative: International Multi-disciplinary Journal of Applied Technology
(ISSN 2995-486X) VOLUME 02 ISSUE 12, 2024

Spring Data Repository, Core Concepts and Usage Technology

Ovxunov Iqboljon Abdunabiyevich
Head of the Department of Computer Engineering, Andijan State University

Fazliddinov Ibrohim Odiljon o‘g‘li
4th grade student of Andijan State University

Abstract:

This article explores the importance of the “Repository Pattern” in the Spring Boot development environment and its role in data access processes. The paper provides detailed information on the key concepts of repositories, their integration with “Spring Data JPA,” and various types of “Repositories”.

Keywords: Java programming language, Spring Boot framework, Repository, Java Spring Boot, Spring Data JPA, Interface.

The spring data repository pattern is essential for simplifying database access in spring boot applications. It provides an abstract layer that separates business logic from database interactions, making code more modular and maintainable. This article explores the core concepts of the spring data repository, its role in the spring boot environment, and best practices for using it effectively.

The process of working with data can be complex. Writing SQL queries correctly and using them efficiently can be challenging, requiring separate code for each “CRUD” operation, leading to code duplication and making management difficult. Additionally, if you write your database operations directly, the testing process becomes complicated and time-consuming, or optimizing SQL queries and using indexes may become complex, potentially slowing down access to data. Especially, managing transactions correctly and establishing connections can be complicated, impacting the integrity and accuracy of the data.

As part of the Spring ecosystem, “Spring Data " is designed to simplify and accelerate working with databases. Through “Repository interfaces,” “Spring Data” provides easy and efficient access to databases. This article will elaborate on the core concepts of the Repository, how it works, and how to utilize it effectively.

A “Repository” is an interface for storing and retrieving data. It provides a central place to manage all the logic for accessing data, abstracting the complexities of storing and retrieving data from the rest of the application. It helps to separate business logic from the database, allowing developers to write less code when working with data.

In “Java Spring Boot,” the “Repository” is a central concept in the Spring data system, simplifying access to data. It is part of the data access layer, providing an abstraction to interact with data sources like databases.

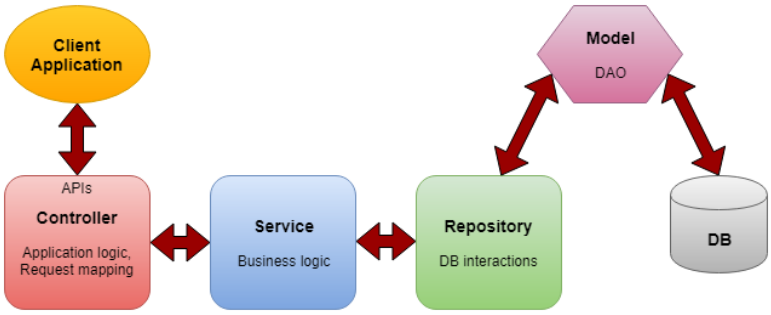

Figure 1. UML (Unified Modeling Language) diagram for the website development process2

Repository – Basics and Analysis.

Data access abstraction means that repositories abstract the logic of data access, allowing developers to interact with the underlying database without writing boilerplate code. In processes based on interfaces, a repository is typically defined as an interface extending one of the “Spring Data” interfaces such as “JpaRepository” or “CrudRepository.” This enables the execution of “CRUD” (Create, Read, Update, Delete) operations. To facilitate automatic implementation, Spring generates repository implementations at runtime based on the method signatures defined in the interface. For creating custom queries, one can specify method names or use the “@Query” annotation for more complex queries. When integrated with Spring, repositories can utilize Spring features such as dependency injection and transaction management.

Creating and using the jpa repository: The jpa repository interface is central to spring data jpa, providing out-of-the-box implementations for common database operations. To implement a repository:

1-Define an interface that extends the jpa repository.

2-Specify the entity type and its primary key.

3-Use method naming conventions to automatically generate queries, or use the ***@Query*** *annotation for custom sql.*

The “Repository” is primarily used with “Spring Data JPA.” “Spring Data JPA” simplifies database interaction based on the “Java Persistence API” (JPA) and provides convenient opportunities for implementing the Repository pattern.

“Spring Data JPA” is part of the Spring Data project and simplifies working with databases through the “Java Persistence API” (JPA). It offers comprehensive capabilities for executing database operations in “Spring Boot.” The main features of “Spring Data JPA” include managing “entities,” where Java classes corresponding to database tables are created as “Entities.” Each “Entity” represents a single record in the database.

Best practices for using spring data jpa

1-Use transaction management: annotate repository methods with ***@Transactional*** *to manage transactions properly and ensure data consistency. Optimize performance: use pagination and limit*

the results for queries that return large datasets to prevent memory overload. Avoid the n+1 query problem: use Fetch joins or **@Entitygraph** *to load associations when necessary. Keep the repository layer separate: ensure that business logic remains outside of the repository layer to keep the data access layer clean.*

Analyzing information about repository interfaces, it is noted that methods necessary for executing "CRUD" operations are automatically created by extending the "JpaRepository" or "CrudRepository" interfaces with "Spring Data JPA."

Transaction Management: Managing transactions is straightforward and effective. With "Spring Data JPA," database operations can be combined. Below is a practical application of the working process of "Spring Data JPA" and "Repository."

1. *Entity class*

```
import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class User {
    @Id
    private Long id;
    private String name;

    // Getters va Setters
}
```

2. *Repository Interface*

```
import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {
    User findByName(String name);
}
```

Now, let's analyze the "Repository Interface."

An interface in Java is a blueprint of a class. It can contain static constants and abstract methods. The interface mechanism achieves abstraction in Java. In a Java interface, there can be no method bodies, only abstract methods. It is used to achieve abstraction and multiple inheritance in Java. More specifically, interfaces can have abstract methods and variables but cannot have method bodies.

Within the Repository Interface, a new interface class was created and named "UserRepository." The next step is to connect the class, inheriting from "JpaRepository," to perform transactions with the underlying database. The "JpaRepository" class accepts the "User entity" and the type of this "Entity"'s ID. This allows for transactions between the database and the "User class."

In conclusion, the "Repository" provides us with the ability to work comprehensively with data. This includes managing transactions with the database, creating custom SQL queries, performing partial checks and limitations on data and provide a powerful abstraction for data access, allowing developers to interact with databases through simple interfaces. This abstraction not only reduces the need for custom sql but also simplifies transaction management and enables scalable data access.

References:

1. Ovkhunov, I.A. (2021). Improving pedagogical conditions for developing a responsible attitude to virtual learning in future teachers. Psychology and education, 58(1), 4035-4041.
2. Ovkhunov, I.A. Information and Communication Technologies in Pedagogical Education. Information and Communication Technologies in Pedagogical Education. №. 2. P. 107-110.

3. Arabbaev, A.A. Information and Communication Technologies in Pedagogical Education. Information and Communication Technologies in Pedagogical Education. 3: 64-67.
4. Bauer, C., & King, G. (2015). Java Persistence with Hibernate. Manning Publications. ISBN: 978-1617290459.
5. Walls, C. (2016). Spring Boot in Action. Manning Publications. ISBN: 978-1617292545.
6. Bloch, J. (2018). Effective Java. Addison-Wesley. ISBN: 978-0134686097.
7. Wong, K. (2018). Spring Microservices in Action. Manning Publications. ISBN: 978-1617293986.
8. Johnson, R. (2018). Expert One-on-One J2EE Design and Development. Wrox Press. ISBN: 978-1119470220.
9. McCool, K.A., Reinders, S., & Robison, V. (2012). Structured Parallel Programming: Patterns for Efficient Computation. Elsevier. ISBN: 978-0124159930.
10. McKinney, G.H. (2018). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. O'Reilly Media. ISBN: 978-1491956249.7