**Hibernate & JPA**

Hibernate-

Hibernate is a Java framework that simplifies the development of Java application to interact with the database. It is an open source, lightweight, ORM (Object Relational Mapping) tool. Hibernate implements the specifications of JPA (Java Persistence API) for data persistence. Hibernate framework uses many objects such as session factory, session, transaction etc. alongwith existing Java API such as JDBC (Java Database Connectivity), JTA (Java Transaction API) and JNDI (Java Naming Directory Interface).

Elements of Hibernate Architecture-

1. Session Factory

The SessionFactory is a factory of session and client of ConnectionProvider. It holds second level cache (optional) of data. The org.hibernate.SessionFactory interface provides factory method to get the object of Session.

1. Session

The session object provides an interface between the application and data stored in the database. It is a short-lived object and wraps the JDBC connection. It is factory of Transaction, Query and Criteria. It holds a first-level cache (mandatory) of data. The org.hibernate.Session interface provides methods to insert, update and delete the object. It also provides factory methods for Transaction, Query and Criteria.

1. Transaction

The transaction object specifies the atomic unit of work. It is optional. The org.hibernate.Transaction interface provides methods for transaction management.

1. Connection Provider

It is a factory of JDBC connections. It abstracts the application from DriverManager or DataSource. It is optional.

1. Transaction Factory

It is a factory of Transaction. It is optional.

JPA

The Java Persistence API (JPA) is a specification of Java. It is used to persist data between Java object and relational database. JPA acts as a bridge between object-oriented domain models and relational database systems. As JPA is just a specification, it doesn't perform any operation by itself.

Connection to H2 database in spring(application properties)-

spring.datasource.url=jdbc:h2:mem:testdb spring.datasource.driverClassName=org.h2.Driver spring.datasource.username=sa spring.datasource.password=password spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

Annotations-

**@Controller:** The @Controller is a class-level annotation. It is a specialization of**@Component**. It marks a class as a web request handler. It is often used to serve web pages. By default, it returns a string that indicates which route to redirect. It is mostly used with **@RequestMapping** annotation.

**@Service:** It is also used at class level. It tells the Spring that class contains the **business logic**.

**@Repository:** It is a class-level annotation. The repository is a **DAOs** (Data Access Object) that access the database directly. The repository does all the operations related to the database.

**@Component**: It is a class-level annotation. It is used to mark a Java class as a bean. A Java class annotated with @Component is found during the classpath. The Spring Framework pick it up and configure it in the application context as a Spring Bean.

**@Bean:** It is a method-level annotation. It is an alternative of XML <bean> tag. It tells the method to produce a bean to be managed by Spring Container.

**@ComponentScan:** It is used when we want to scan a package for beans. It is used with the annotation @Configuration. We can also specify the base packages to scan for Spring Components.

**@Configuration:** It is a class-level annotation. The class annotated with @Configuration used by Spring Containers as a source of bean definitions.

**@Autowired:** Spring provides annotation-based auto-wiring by providing @Autowired annotation. It is used to autowire spring bean on setter methods, instance variable, and constructor. When we use @Autowired annotation, the spring container auto-wires the bean by matching data-type.

**@RequestMapping**: It is used to map the web requests. It has many optional elements like consumes, header, method, name, params, path, produces, and value. We use it with the class as well as the method.

**@GetMapping**: It maps the HTTP GET requests on the specific handler method. It is used to create a web service endpoint that fetches It is used instead of using: @RequestMapping(method = RequestMethod.GET)

**@PostMapping**: It maps the HTTP POST requests on the specific handler method. It is used to create a web service endpoint that creates It is used instead of using: @RequestMapping(method = RequestMethod.POST)

**@PutMapping**: It maps the HTTP PUT requests on the specific handler method. It is used to create a web service endpoint that creates or updates It is used instead of using: @RequestMapping(method = RequestMethod.PUT)

**@DeleteMapping**: It maps the HTTP DELETE requests on the specific handler method. It is used to create a web service endpoint that deletes a resource. It is used instead of using: @RequestMapping(method = RequestMethod.DELETE)

JPA allows to map application classes to tables in database.

* Entity Manager - Once the mappings are defined, entity manager can manage your entities. Entity Manager handles all interactions with the database
* JPQL (Java Persistence Query Language) - Provides ways to write queries to execute searches against entities. Important thing to understand is the these are different from SQL queries. JPQL queries already understand the mappings that are defined between entities. We can add additional conditions as needed.
* Criteria API defines a Java based API to execute searches against databases.

@Entity: Specifies that the class is an entity. This annotation is applied to the entity class.

@NamedQuery: Specifies a static, named query in the Java Persistence query language.

@Id: Specifies the primary key of an entity.

@GeneratedValue: Provides for the specification of generation strategies for the values of primary keys.

protected User(): Default constructor to make JPA Happy

@Transactional: Spring annotation used to simplify transaction management

@PersistenceContext: A persistence context handles a set of entities which hold data to be persisted in some persistence store (e.g. a database). In particular, the context is aware of the different states an entity can have (e.g. managed, detached) in relation to both the context and the underlying persistence store.

EntityManager : Interface used to interact with the persistence context.

entityManager.persist(user): Make user entity instance managed and persistent i.e. saved to database.

entityManager.createNamedQuery: Creates an instance of TypedQuery for executing a Java Persistence query language named query. The second parameter indicates the type of result.

CommandLineRunner- CommandLineRunner interface is used to indicate that this bean has to be run as soon as the Spring application context is initialized.

Hibernate Mapping-

It is one of the key feature of hibernate.They establish the relationship between two database tables as attributes in your model that allows you to easily navigate the associations in your model and criteria queries. You can establish either unidirectional or bidirectional it will not impact your database mapping tables, but it defines in which direction you can use the relationship in your model and criteria queries.

1. One to one –(Annoatation=@onetoone)

It represents the one to one relationship between two tables.

1. One to many/many to one –(Annoation = @onetomany/@manytoone)

It represents the one to many relationship between two tables.

1. Many to many – (Annotation = @manytomany)

It represents the many to many relationship between two tables.

Transaction Management-

A database transaction is a sequence of actions that are treated as a single unit of work. These actions should either complete entirely or take no effect at all. Transaction management is an important part of RDBMS-oriented enterprise application to ensure data integrity and consistency. The concept of transactions can be described with the following four key properties described as ACID −

* Atomicity − A transaction should be treated as a single unit of operation, which means either the entire sequence of operations is successful or unsuccessful.
* Consistency − This represents the consistency of the referential integrity of the database, unique primary keys in tables, etc.
* Isolation − There may be many transaction processing with the same data set at the same time. Each transaction should be isolated from others to prevent data corruption.
* Durability − Once a transaction has completed, the results of this transaction have to be made permanent and cannot be erased from the database due to system failure.