

# Contents

- Need for a abstraction framework layer between DAO and Database
- JPA (Java Persistence API)
- Hibernate
- Entity Class and Annotation
- JPA Repository Maven Dependency Tree
- ResponseEntity

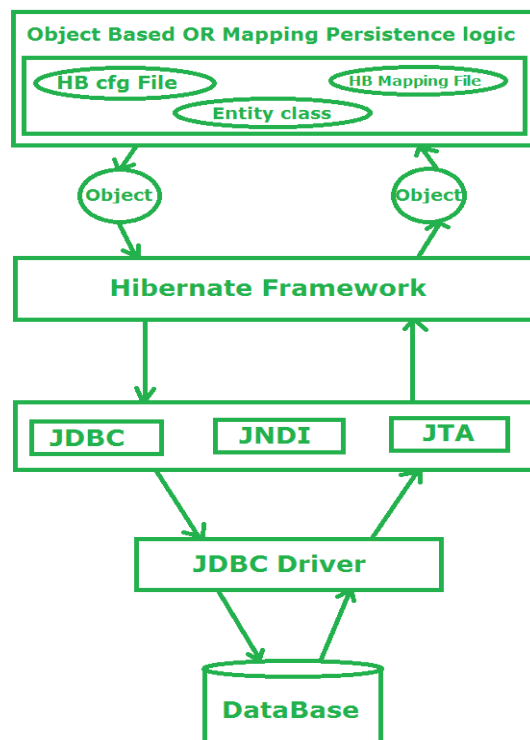
**JPA (Java Persistence API) :** A JPA (Java Persistence API) is a specification of Java which is used to access, manage, and persist data between Java object and relational databases. It is considered as a standard approach for Object Relational Mapping. JPA can be seen as a bridge between object-oriented domain models and relational database systems. Being a specification, JPA doesn't perform any operation by itself. Thus, it requires implementation. So, ORM tools like Hibernate, TopLink, and iBatis implement JPA specifications for data persistence.

**Hibernate ORM :** A Hibernate is a Java framework which is used to store the Java objects in the relational database system. It is an open-source, lightweight, ORM (Object Relational Mapping) tool. Hibernate is an implementation of JPA. So, it follows the common standards provided by the JPA.

## Need for an abstraction framework layer between DAO and Database

**Database :** The most obvious benefit of JDBC over JPA is that it's simpler to understand. The main advantage of JPA over JDBC for developers is that they can code their Java applications using object-oriented principles and best practices without having to worry about database semantics. As a result, development can be completed more quickly, especially when software developers lack a solid understanding of SQL and relational databases. Also, because a well-tested and robust framework is handling the interaction between the database and the Java application, we should see a decrease in errors from the database mapping layer when using JPA.

JPA is a specification. It provides common prototype and functionality to ORM tools. By implementing the same specification, all ORM tools (like Hibernate, TopLink, iBatis) follow the common standards. In the future, if we want to switch our application from one ORM tool to another, we can do it easily.



**Fig: Working Flow of Hibernate framework to save/retrieve the data from the database in form of Object**



51



Why use JPA instead of directly writing SQL query on Java File (i.e. directly to JDBC) ?

Certain projects require engineers to focus more on the object model rather than on the actual SQL queries used to access data stores. The question can actually be interpreted as

Why should one use an ORM framework ?

which can have different answers in different contexts.

Most projects can benefit from having a domain model, with persistence being a second concern. With JPA (implementations) or most other ORM frameworks, it is possible to have all entities i.e. tables in your database, modelled as classes in Java. Additionally, it also possible to embed behavior into these classes and therefore achieve a behaviorally rich domain model. The entities in this model can have multiple purposes, including the purpose of replacing DTOs for transporting data across tiers.

Challenge 1 : If I add Spring boot starter jpa dependency in the POM.xml , do we still need to add mysql-connector-java dependency in our POM.xml ?

Challenge 2 : What's the difference between JPA and Hibernate?

HINT : JPA is just a specification, meaning there is no implementation. You can annotate your classes as much as you would like with JPA annotations, however without an implementation nothing will happen. Think of JPA as the guidelines that must be followed or an interface, while Hibernate's JPA implementation is code that meets the API as defined by the JPA specification and provides the under the hood functionality.

When you use Hibernate with JPA you are actually using the Hibernate JPA implementation. The benefit of this is that you can swap out Hibernate's implementation of JPA for another implementation of the JPA specification. When you use straight Hibernate you are locking into the implementation because other ORMs may use different methods/configurations and annotations, therefore you cannot just switch over to another ORM. *JPA is the dance, Hibernate is the dancer.*

Challenge 3 : Let us create a project to showcase database connection to Spring Boot app now . This will be a spring boot application which implements the crud functionality for a User management system . We can CRUD User entities. Data of users needs to be persisted to a MySQL database using JPA and Hibernate as ORM .

HINT : We will be following the Spring MVC framework .

<https://www.geeksforgeeks.org/spring-mvc-framework/>

<1>Create a Spring Boot Maven project using start.spring.io . We will be creating CRUD API and persisting our data to a MySQL database using JPA with Hibernate as ORM implementation . Which dependencies will we need ?

Note : Don't forget to add the MySQL Connector Java, Spring Data JPA and Spring Boot Starter Validation maven dependency to POM.xml .This is the driver we need to establish the connection between our Spring Boot application and the MySQL database

.

Let's explore the pom and explore the parent dependencies of spring boot starter jpa .  
We will be able to find hibernate core .

<2> Ques : Now just after adding all dependencies are we able to start the application ?

-> No , because we need to provide datasource url for any jpa application

Define the below properties in app.properties .

-url

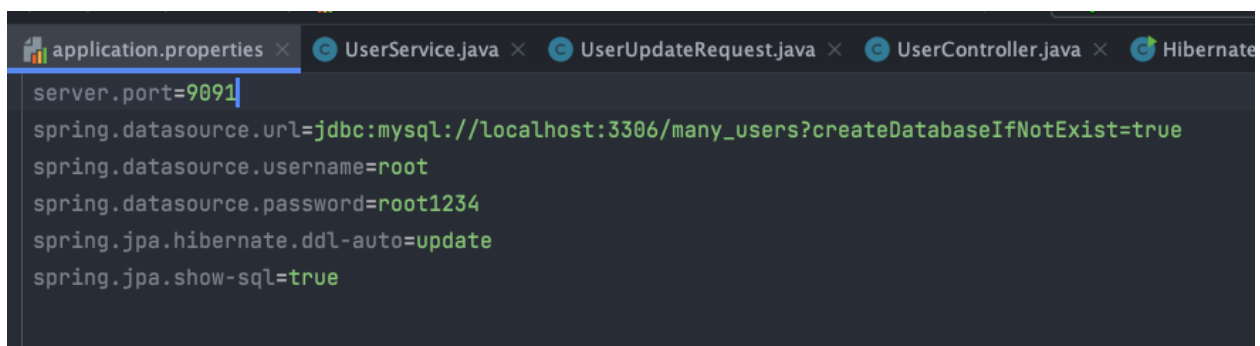
-username

-password

-spring.jpa.hibernate.ddl-auto

-spring.jpa.show-sql

We don't need to create a Database manually as we can tell Spring to use a specific db  
create database if it does not yet exist.



```
application.properties x UserService.java x UserUpdateRequest.java x UserController.java x Hibernate
server.port=9091
spring.datasource.url=jdbc:mysql://localhost:3306/many_users?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=root1234
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

Let's discuss the modes for spring.jpa.hibernate.ddl-auto from the documentation.

<3> We can reuse the Controller class from the JDBC project .

Database and Table will be created itself by JPA . No need for any connection object .

Except the Users.java is an entity class with annotations for Primary Key , auto increment , unique constraint etc . @Entity is the annotation from JPA which tells Hibernate to map this class to a table in the database .

<4> We define DAO differently .The DAO will be UserRepository interface and it will extend from JpaRepository.

Let's explore the JpaRepository interface and create the UserRepository interface which extends it .

In Service class , we use the repository object to call the methods we are extending from JpaRepository and perform the crud operations according to the api request received at the controller level .

Challenge 1 : How to get the maven dependency tree for a dependency ? HINT :

```
[INFO] -----
[INFO] [~/Desktop/JBDL-23-master/L10_Spring JPA]$ mvn dependency:tree -Dincludes=org.apache.logging.log4j:log4j-to-slf4j
[INFO] Scanning for projects...
[INFO] -----< com.example:demo-jpa >-----
[INFO] Building demo-jpa 0.0.1-SNAPSHOT
[INFO] -----[ jar ]-----
[INFO] --- maven-dependency-plugin:3.2.0:tree (default-cli) @ demo-jpa ---
[INFO] com.example:demo-jpa:jar:0.0.1-SNAPSHOT
[INFO] \- org.springframework.boot:spring-boot-starter-web:jar:2.6.1:compile
[INFO]    \- org.springframework.boot:spring-boot-starter:jar:2.6.1:compile
[INFO]       \- org.springframework.boot:spring-boot-starter-logging:jar:2.6.1:compile
[INFO]          \- org.apache.logging.log4j:log4j-to-slf4j:jar:2.17.0:compile
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
```

mvn dependency:tree -Dincludes=org.apache.logging.log4j:log4j-to-slf4j

mvn dependency:tree -Dincludes=<GROUP\_ID> : <ARTIFACT\_ID>

DOCUMENTATION : <https://maven.apache.org/plugins/maven-dependency-plugin/examples/filtering-the-dependency-tree.html>

Challenge 2 : UserRepository does not have @Repository on top .How is UserRepository acting as a bean ?

Challenge 3 : How does Hibernate know by data type which is id ? Is there any annotation to make it auto\_increment ? How to make our email id as unique and not null and length of email not greater than 30 characters ? (explore the @Column annotation)

Challenge 4 : Which spring.jpa.hibernate.ddl-auto mode to use in Production ?

Challenge 5 : Generation type identity vs auto ?

HINT : \* IDENTITY - The auto incremented values are generated by mysql

\* AUTO - The auto incremented values are generated by hibernate . It follows global sequence and stores it in the hibernate\_sequence table .

Challenge 6 : How to change the name of your (i.) table (ii.) column

Challenge 7 : Explore javax.persistence , entity manager

Challenge 8 : What is an Optional object which is returned by findById() of dao object

Challenge 9 : What will be the status code if we hit the post api but the table is not created ? HINT : 500

Challenge 10 : What will be the status code if we hit post api but validation is failed for email ? HINT : 400

Challenge 11 : What is hibernate\_sequence table and why does it get created in our db when we use Generation Type as Auto but not when we have IDENTITY ?Who has created this table?

HINT : Hibernate

Challenge 12 : Is auto increment in your sql query when Generation Type as Auto ? Why ?



**ResponseEntity** : ResponseEntity is meant to represent the entire HTTP response. You can control anything that goes into it: status code, headers, and body.

```
@PostMapping("/user")
public ResponseEntity<String> createUser(@Valid @RequestBody UserCreateRequest userCreateRequest) {
    try{
        userService.createUser(userCreateRequest);
        return new ResponseEntity<String>( body: "User is created", HttpStatus.CREATED);
    }catch (Exception e){
        return new ResponseEntity<>(e.getLocalizedMessage(), HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

