Spring Boot Framework

1. Spring Boot is a java based framework.
2. Spring boot is use for a faster development and also it reduce the development efforts.
3. Using a Spring Boot you can create a production ready application.
4. Spring Boot provide embedded servers like Tomcat server, Data Base server such as H2 Database.
5. Spring boot is a flexible framework where you can customize as per your requirement.
6. Spring Boot is based on Spring Framework.
7. Spring Boot is majorly used for Full stack application and microservices.
8. Spring framework is divided into multiple module. You can use a module as per you project requirement.
   1. Spring IOC/Core
   2. Spring JDBC
   3. Spring ORM
   4. Spring REST
   5. Spring batch
   6. Spring MVC
   7. Spring Cloud
   8. Spring JMS
   9. Spring Security
9. How to implement Module in spring boot project
   1. Add the dependencies of Spring Module into project.
   2. Configure the model into project using predefine configuration.
   3. Use a Spring APIs to implement the functionality.
10. Spring Documentation

<https://docs.spring.io/spring-framework/docs/current/reference/html/>

<https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/>

1. Spring boot is a wrapper of spring framework. Spring framework issue such as setup, configuration and dependency management etc. has been resolve in this spring boot.
   1. Spring Boot provides a starter project which contains the required dependency for the modules.
   2. Spring Boot provides the auto configuration feature using which the module configuration will be happened automatically. You can customize the configuration as per your requirement.
   3. Spring provides the embedded tomcat server and H2 server by default.

**Spring Boot Project Creation**

1. Spring CLI (Command Line Interface)
   1. In this approach you have to install the Spring CLI tool to create spring boot application.

<https://docs.spring.io/spring-boot/docs/current/reference/html/cli.html>

1. Spring STS (Spring Tool Suit)
   1. This is the IDE provided by Spring community. This is an extension for eclipse, IntelliJ etc. IDE.

<https://spring.io/tools>

1. **Spring Initializer**
   1. **Is a web application which is use to create a spring boot project. Using this you can create a spring project which will be downloaded as a zip file.**

[**https://start.spring.io/**](https://start.spring.io/)

Types of application (Way of application development)





**Create Spring Boot Project**

1. Go to Web Application to create spring boot project.

<https://start.spring.io/>



1. Get the Project Zip file and extract it into specific location.
2. Import the project into eclipse.
   1. Open an Eclipse workspace.
   2. Go To “File” Menu - > Select “Import…” option
   3. You have to search for “Maven” option into the wizard of the new window



* 1. Select “Existing Maven Project” option in the list and click on “Next”
  2. Select the extracted folder as a Root Directory (Make sure that select the folder which has pom.xml)
  3. Click on “Finish”

**Spring Core/IOC**

1. **Spring Bean classes.**
2. **SI, CI and DI**
3. **Spring Annotations (@Bean, @Autowire, @Qualifier, @Component, @Primary)**
4. **Autowire**
5. Spring Container

@SpringBootApplication

1. It is a combination of 3 annotations internally
2. @Configuration
   1. To Declare class as a configuration class.
   2. These classes will be loaded inside spring container at the initial stages.
3. @EnableAutoConfiguration
   1. This use to enable the auto configuration of the spring boot application.
   2. This configuration will be perform by spring boot internally by looking into the dependencies added inside the project.
4. @ComponentScan
   1. To scan the spring bean classes from the project.
   2. This will scan the project (given package) and create and set object of spring bean classes inside container.

**Spring Bean Classes**

1. Spring bean classes are the java classes for which spring will create an object and also manages the java objects.
2. These classes can be a build-in class or custom class.
3. To create and maintain the java objects spring will make a use of Spring Container which is application context.
4. To declare any java class as a spring bean class you can use the following annotations
   1. @Component
   2. @RestController
   3. @Controller
   4. @Service
   5. @Repository
   6. @ControllerAdvice etc.

**SI and CI**

1. SI stand for Setter Injection which is a process where you can initialize the instance variable using a setter method
2. CI stand for constructor injection which is a process where you can initialize the instance variable using a constructor.
3. By SI you can give a flexibility to the user to initialize the specific instance variable.
4. By CI you can restrict user to initialize the instance variable.

**Dependency Injection**

1. One class object will be created inside another class is the Dependency Injection.
2. This is also known as HAS-A relation in java.
3. This Dependency Injection can be automated by autowire process.
4. @Autowire: it is a process in which spring will identify the dependency and inject those object into the java class internally (To make dependency injection process internally/automatically).

**Spring Container**

1. Spring container will scan the project for the spring bean class.
2. It will create and hold the spring bean object.
3. It also manages the life cycle of the object.
4. It will also provides the object whenever required.
5. In Spring boot application ApplicationContext is container which can be use to get the object of spring bean class.

**Spring Bean Scope**

1. You can control how many object to be created and when to create object of bean classes.
2. There are different scope of the bean
   1. Singleton
      1. This is the default scope.
      2. It will create only one object of the bean class in a container
   2. Prototype
      1. There will be multiple object created for every request
   3. Request
      1. There will be separate object created for every HttpRequest
   4. Session
      1. There will separate object created for a session (HttpSession)
   5. Gloablesession
      1. There will be one object created in a project.

Spring Boot Project Structure



**Creating object manually for spring container**

1. The Objects of the java classes can be created manually and then those object can be assign to spring to for management.
2. To do this you have to create a method which will return the object of these type of classes and annotate that method with @Bean Annotation.
3. **Example:**

@Bean

**public** DbSetup getDbSetupObject() {

DbSetup setup = **new** DbSetup();

setup.setDBConnection();

**return** setup;

}

**JSON**

1. JSON stands for **J**ava**S**cript **O**bject **N**otation
2. JSON is use as a common language to communicate between the different applications based on different language or platform.
3. JSON used in a key and value pair format. One key and value pair also known as element.
4. In JSON key are always in String format and vales can be in String, Numeric, Boolean, JSON object or JSON array format.
5. JSON can be represented in 2 form
   1. JSON Object
   2. JSON Array

**JSON Object:**

1. JSON object can have a key and value pairs.
2. JSON Object will be represented by curly brackets.
3. Symbol:

{

“Key”: value,

“Key”: value

}

**JSON Array:**

1. JSON array can have a group of values or group of JSON Objects
2. JSON array will be represented by square bracket
3. Symbol:

[ value1, value2, … ]

[

{

Key:value

},

{

Key:value

}

]

Examples of JSON

**JSON Object**

{

“name”: “Abc”,

“nickName”:”Abc”,

“email”: “[abc@gmail.com](mailto:abc@gmail.com)”,

“contact”:998877667788,

“salary”: 45645.45,

“status”: “Active”,

“isCurrentEmp”: true

}

**JSON Array**

[76.67, 45.67, 76.5, 88,44]

[“Abc”, “Xyz”, “Pqr”]

{

“name”: “Xyz”,

“contact”: 8987799797,

“skills”: [“Java” ,”Html” ,”CSS” ,”JS”],

“address”: {

“city”:”pune”,

“pincode”:”998877”,

“state”: “MH”

},

“projects”: [

{

“name”:”Lib Management”,

“technologies”: [“Java”, “Html”, “CSS”],

“desc”: “This application is used to automate”

},

{

“name”:”Student Management”,

“technologies”: [“Aps.net”, “Html”, “CSS”],

“desc”: “This application is used to automate”

}

]

}

**JSON To Java Class and vice versa**

1. JSON to Java class and Java class to JSON can be done internally by Spring.
2. To do this you can use the GSON or Jackson library.
3. Every java program has to convert the JSON into Java object to process the data in a program.
4. For every JSON Object you have to create a Java class. and every JSON array will be represented as a Java Array or collection.











Postman Tool.

1. This tool is use to test the REST APIs.
2. It is use to generate the requests and get the response back.
3. To download it use the following URL

<https://www.postman.com/downloads/>

**REST API using Spring Boot**

1. REST stans for **RE**presentational **S**tate **T**ransport.
2. REST API is also known as Restful API.
3. REST API is use to expose the services (Web services) of one application to the another application.
4. REST APIs accepts a JSON/~~XML~~ and return the JSON/~~XML~~
5. Create REST API in Spring Boot Application
   1. Create a java class and mark it as a rest controller using **@RestController** annotation.
   2. Create a method which return the values as a Rest API output/response.
   3. Annotate the method using an annotation such as @GetMapping, @PostMapping, @PutMapping, @DeleteMapping etc.
   4. Provide the URL of the REST API which is also known as End Point

**Accept the user data from the REST API**

1. User data can be accept using different option.
2. Accepting Data using **Request Parameter**
   1. It is use to get the user data using an request parameter
   2. These parameters are pass from the URL. Which is appended inside request after ‘?’
   3. To accept the parameter value into the spring controller you have to create an input parameter of the method.
   4. And use **@RequestParam** annotation to receive the parameter values inside the input variable.

URL : localhost:8080/user-name**?fn=Abcd&mn=Pqr&ln=Xyz**

`

1. Accept data using **path variable**
   1. It is use to get the user data using a path variable
   2. This is use to pass the Values as a part of URL. Which is also known as dynamic part of the URL
   3. To receive the values in the spring boot application you can use **@PathVariable** annotation.

URL : localhost:8080/user-name/**abc/pqr/xyz**



1. Accept Data using **JSON (Request Body)**
   1. It is use to get the user data using a request body in the form of JSON
   2. This data will be pass as the request body from the Internal of the URL.
   3. To receive this data into sprig boot application you can use **@RequestBody** annotation

URL : localhost:8080/user-data



**Http Methods for REST API**

1. GET
   1. GET method is recommended to get/retrieve the records from the backend application.
   2. To Create **HTTP GET** method you can use a **@GetMapping annotation**
   3. If you use the same method multiple time then it **return an same result** and hence it is also known as **idempotent method**.
2. POST
   1. POST method is use to create/insert a new resources at the backend application.
   2. To create **HTTP POST** method you can use **@PostMapping annotation**
   3. If you use the same method multiple time then it **return an different result** and hence it is also known as **non-idempotent method**.
3. PUT
   1. PUT method is use to update the existing resources from the backend application.
   2. To create **HTTP PUT** method you can use **@PutMapping annotation**
   3. If you use the same method multiple time then it **return an same result** and hence it is also known as **idempotent method**.
4. DELETE
   1. DELETE method is use to delete the existing resources from the backend application.
   2. To create **HTTP DELETE** method you can use **@DeleteMapping annotation**
   3. If you use the same method multiple time then it **return an same result** and hence it is also known as **idempotent method**.

**REST API and REST Endpoint**



To Allow the Cross you have to use the following annotation

**@CorssOrigin(value = “\*”)**

is use to allowed all server to user REST API

**@CorssOrigin(value = “localhost:3000”)**

It allows only server running on 3000 port to access the REST API

**Swagger/Open API Tool**

1. Swagger tool is use to create a REST API documentation.
2. It is also used for the REST API execution (API testing)
3. Swagger is also known as Open API into this new version.
4. <https://springdoc.org/>
5. Implement Swagger into the Backend application.
   1. Adda Dependency into application (pom.xml)

**Following Dependency for Spring Boot less than 3.x.x**

<dependency>

<groupId>org.springdoc</groupId>

<artifactId>springdoc-openapi-ui</artifactId>

<version>1.6.14</version>

</dependency>

**Following Dependency for Spring Boot 3.x.x version above**

<dependency>

<groupId>org.springdoc</groupId>

<artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>

<version>2.5.0</version>

</dependency>

* 1. Use a swagger URL to get UI

<http://localhost:8080/swagger-ui.html>

10-Aptil-24

Task-1

Create a rest API which in which accept the number for the user and print the table for the give number on console.

Task-2

Create a rest API which in which accept the 2 numbers and the operation(+,-,\*,/) for the user and return the output to the client.

Task-3

Create a REST API to achieve the product cart operations

1. Create a REST API to add the product into cart

Product Details: product id, product name, product price, product quantity

2. Create a REST API to delete the product from the cart using product Id

3. Create a REST API to update the quantity of the product.

4. Create a REST API to generate the bill

Response:

{

"products" : [

{

"productid":123,

"productname":"Mobile",

"productprice":15000,

"productquantity":2,

"totalprice":30000

},

{

"productid":133,

"productname":"pendrive",

"productprice":1000,

"productquantity":1,

"totalprice":1000

}

],

"totalamount": 31000,

"taxamount": 3100,

"payableamount":34100

}

5. Create a RESt API to get the list of all the product.

Hints

1. Create array list of type Product class of name cart.

**Spring ORM**

1. This module is use to connect Spring applications with the Database using ORM tool.
2. ORM Tool
   1. ORM stands for **O**bject **R**elational **M**apping
   2. These tools are use to connect with databases and also use to perform Db operation more efficiently.
   3. Examples of ORM tools are JPA, EJB, Hibernate.
   4. In these tools you can connect with DB using the configuration instead of writing a code.
   5. It can generate the DDL, DML and DQL queries internally. And also execute them to get the results.
   6. The SQL checked exception has been converted into unchecked in ORM tools.
   7. It also provided the caching to improve the performance of the DB operations.

****

**Implement the JPA (Hibernate) in Spring Boot Application**

1. Add a Jar file into project

Spring JPA

Driver Jar (MySql Connector)



1. Add the DB Configuration into application.
   1. You can add a configurations into application.properties file present inside the src/main/resources
   2. You can get all the spring configuration in following URL

<https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html>



1. Create an Entity classes in the application.
   1. You have to create a class which has private variable and getter/setter. (DTO class)
   2. There are two mandatory annotations which you have to use inside this class @Entity and @Id
   3. Entity classes will be further connect with the table through the hibernate.
2. Use an API to perform the DB operations

API: **JPARepository interface**

* 1. This is used to execute the Database operations(DML, DQL)
  2. Using APIs you can execute the queries and get the result back into java application.
  3. Steps to create JpaRepository
     1. Create an Interface and extends the JpaRepository Interface on it.
     2. Mark your Repository interface with @Repository annotation.
     3. JpaRepository Interface has multiple methods to perform CURD/CRUD
     4. You have to specify the Entity class and the Primary key of it while creating JpaRepository.
     5. For Entity class there will be a separate Repository created.

**Task on CURD Operation**

1. Manage Student Information

a. Insert Student Details into DB (id, name, email, contact)

b. Update the Student Details by Id

c. Delete the student by id

d. Get the details of the student by id

2. Manage Employee Information

a. Insert employee Details into DB (id, name, salary, city)

b. Update the employee Details by Id

c. Delete the employee by id

d. Get the details of the employee by id

3. Manage Product Information

a. Insert Product Details into DB (id, name, price, quantity, rating (1-5))

b. Update the Product Details by Id

c. Delete the Product by id

d. Get the details of the Product by id

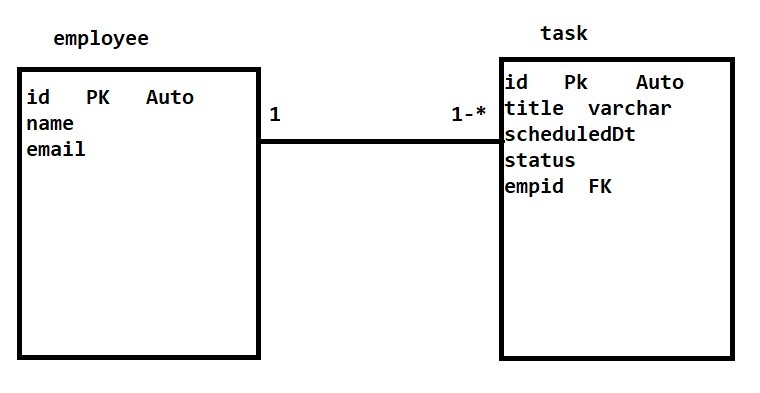
**Using Query in Spring Boot JPA**

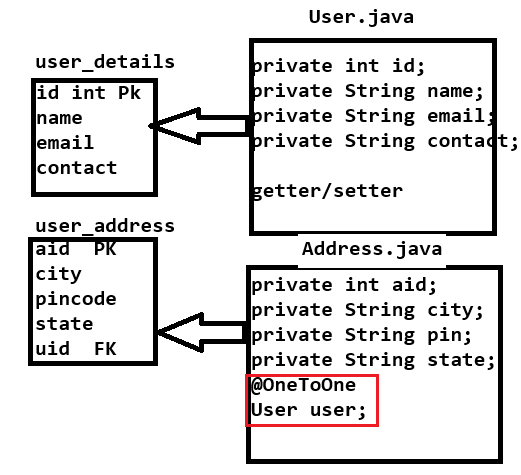
1. To perform the Db operation other than primary key, you must have to write the query manually.
2. There are 2 ways to provide the query in the JPA
   1. HQL/JPQL
      1. In this type you will use the entity class name and variable name instead of column and table name.
      2. This queries are independent to the database, so the changes happened in the table will not effect your queries.
      3. This will gets converted into the SQL internally.
   2. SQL
      1. In this you will use the table and column name like an normal SQL query.
      2. This query is also known as native queries.
      3. While execution hibernate will not apply any conversion on it.
3. Steps To execute manual query
   1. Create a method (abstract method) inside Repository.
   2. Annotate that method with @Query annotation. And define your query.
   3. If query is of DML type then also use @Modifying and @Transactional annotation.
   4. If query is SQL type then set the a native query flag.

**JPA Relational Mapping (HAS-A)**

1. Using this you can achieve a relational mapping between the table. And also can add the primary and foreign key relation between tables.
2. You can create an reference of one class into another class to create a foreign key inside the Database table.
3. Create a reference of a class for which you wanted to create a foreign key, in the class where you wanted to create a foreign key column.
4. And then annotate the reference with the any one of the following annotation
   1. @OneToOne
   2. @OneToMany
   3. @ManyToOne
   4. @ManyToMany







Create a REST API to achieve following requirement.

1. Create a table cart with following column

Pid (pk), name, price, quantity

1. Create a REST API to insert product into cart.
2. REST API to remove the Product from cart by Product Id.
3. REST API to update the product quantity.
4. Create a REST API which will return the Billing Details of the cart product.

{

name: “”,

price: “”,

quantity: “”,

total: “”, // price \* quantity

billingDetails: {

“totalPrice”:”” // addition of all product total

“CGST”:”” // price by 4%

“SGST”:”” // price by 4%

“payableAmount”: “” // addition of above 3 fields

}

}