**SPRING BOOT**

[**1. HOW TO CREATE A SPRING BOOT PROJECT?**](#_3i4y1fskl1va) **2**

[**2. BETTER WAY OF CREATING A PROJECT?**](#_1ikl672ym01l) **2**

[**3. HOW TO TEST THE CREATED PROJECT?**](#_cpalgz3mon3f) **2**

[**4. HOW TO ADD WEB WIZARD IN STS?**](#_o7gs2q8dvubn) **3**

[**5. HOW TO ADD JSP IN STS?**](#_56u6686yxdxx) **3**

[**6. HOW TO RUN JSP ON SERVER?**](#_5yzwvug2kbh7) **3**

[**7. HOW TO IMPLEMENT JPA ?**](#_4wdqitu0114h) **4**

[**8. HOW TO USE JPQL QUERY IN JPA ?**](#_jvvbb7mpuc58) **5**

[**9. HOW TO USE NATIVE QUERY IN JPA ?**](#_swyhbn6dnswo) **6**

# 

# HOW TO CREATE A SPRING BOOT PROJECT?

* 1. Visit <https://start.spring.io> website
  2. Select gradle or maven as project type
  3. Select java or kotlin as language
  4. Give project name, package name, etc.
  5. Select jar or war package type
  6. Select java version
  7. Select dependencies:
     1. Web
     2. Jasper
     3. Security
     4. Devtools
     5. JPA
     6. MYSQL
  8. Click generate. This will create a zip file of the project
  9. Open this folder with an IDE

# BETTER WAY OF CREATING A PROJECT?

* 1. Open spring tool suit STS IDE
  2. Click new and select spring starter project
  3. Fill required fields mentioned above in point number 1

# HOW TO TEST THE CREATED PROJECT?

* 1. Make sure that the annotation above the Application class is *@SpringBootApplication*
  2. Create a Controller java class
     1. Annotate the class with *@Controller*
     2. Create a public method that returns a string
     3. Annotate this method with @ResponseBody. This will show the returned value on the web page, otherwise no value will be shown on the page.
     4. Annotate the method with @RequestMapping("/test").   
        “/test” could be anything you want.  
        This is required to find this method when we run the server.  
        When we run the server, we will write localhost:portnumber/test to call this method
     5. Save the project and click run spring boot app.
     6. Then the server will start
     7. Look for the post number in the logs
     8. Open a web browser and type localhost:portnumber/test to test the method that we have created. Now the string which we have written in the method shall be visible on the web page.

# HOW TO ADD WEB WIZARD IN STS?

* 1. Click help
  2. Click install new software
  3. Select all websites option from the work with drop-down list
  4. Uncheck hide items that are already installed option
  5. Search “web” word
  6. Select eclipse web developer tools package and install it

# HOW TO ADD JSP IN STS?

* 1. Click help
  2. Select eclipse marketplace option
  3. Search “jsp” word
  4. Install eclipse enterprise java and web developer tools package

# HOW TO RUN JSP ON SERVER?

* 1. Click plain src folder
  2. Click main
  3. Inside that create a folder called webapp
  4. Inside that create a folder called views
  5. Create a jsp file inside this folder
  6. Now open src/main/resource
  7. Click application.properties = here we will define the view resolver
  8. Type following lines:  
     spring.mvc.view.prefix=/views/

spring.mvc.view.suffix=.jsp

* 1. Google embedded jasper maven library. Copy paste that library in pom.xml file under dependencies section. This is important for embedding our jsp file in the browser on our server.
  2. Write some function in controller file. The return value of this function should be the name of our jsp file.   
     For eg. if we have a home.jsp file, then, the return value of the above said function should be return “home”.
  3. The name and path of the function could be anything you want.
  4. Now run start the server and hit this path, your jsp file will be loaded in the browser.

# HOW TO IMPLEMENT JPA ?

* 1. Integrate JPA & Mysql dependency
  2. Open application.peoperties file and write following configuration:  
     spring.datasource.url=jdbc:mysql://localhost:3306/testb

spring.datasource.username=root

spring.datasource.password=password

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=update

* 1. Create a model class
     1. Write @Entity annotation on the class name
     2. Write data members of the class
     3. Annotate id data member with   
        @Id  
        @GeneratedValue(strategy = GenerationType.AUTO)
  2. Then create an interface and extend it with CrudRepository<ModelClassName, Integer>
  3. Open application class.
     1. In the main function, get the application context like this:  
        ApplicationContext context = SpringApplication.run(JpaExampleApplication.class, args);
     2. Get bean class like this:  
        Interface interface = context.getBean(Interface.class)
     3. Then create an object of the model class with some data in it.
     4. Finally save the model class in database using the interface instance like this:  
        Interface\_instance.save(model object);
     5. Make sure the the dependencies are correctly created in the pom.xml file. This has wasted many hours, because the default mysql dependency was itself written wrongly.
  4. To create custom query function, write such method in the repository interface.  
     For example, suppose the User model class has a name data member, and we want to query the table by name, then; create a method like this:  
     public List<User> getUserByName(String name);
     1. To query the table by name and ID, create a method in repo interface like this:  
        public List<User> findByNameAndId(String name, int id);
     2. To know more about how to create such query method, google Mysql query creation. Here, we can learn to create methods with AND, OR, LESS THAN, GREATER THAN, etc. attributes.

# HOW TO USE JPQL QUERY IN JPA ?

* 1. To create a query function with a JPQL query, write a function in the repo interface.
  2. Annotate it with @Query annotation
  3. Write following query to get all the elements in the table:  
     @Query("select user from User user")  
       
     Here, User is the name of our model class.  
     Make sure that the spelling and upper-lowercase of the model class is written correctly here.
  4. Write following query to get a user by name from the table:  
     @Query("select u from User u where user.name =:n")

public List<User> getUserByNamee(@Param("n") String name);  
  
Here name is the data member of the User model class.

The Param annotation defines the identity of the name parameter.

Identity of the name parameter is ‘n’.

* 1. Write following query to get a user by name and ID from the table:  
     @Query("select u from User u where u.n =:name and u.id =:id")

public User getUserByNameeAndId(@Param("n") String name, @Param("id") int id);

Here name & id are the data members of the User model class.

The Param annotation defines the identity of the name and id parameter.

Identity of the name parameter is ‘n’.

Identity of the id parameter is ‘id’.

# HOW TO USE NATIVE QUERY IN JPA ?

* 1. To create a query function with a JPQL query, write a function in the repo interface.
  2. Annotate it with @Query annotation
  3. Write following query to get all the elements in the table:

@Query(value = "select \* from user", nativeQuery = true)

public List<User> getAllUserss();

* 1. Write following query to get all the elements by name in the table:  
     @Query(value = "select \* from user where name =:name", nativeQuery = true)

public List<User> getUserByNameee(@Param("name") String name);