Store spring boot application

Add dependencies using spring initializer

* Spring web
* Spring boot dev tools
* My sql driver
* Spring data jpa
* Validation
* Thymeleaf
* lambok

Generate and open in inteljpa

Main>java>

Main.application and run

Application.properties add database and port no

server.port=8088  
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
spring.datasource.url=jdbc:mysql://localhost:3306/store  
spring.datasource.username=root  
spring.datasource.password=Hema@sri18  
  
spring.jpa.show-sql=true  
spring.jpa.hibernate.ddl-auto=update

now create new schema in my sql in the name of “store“ in 3rd line

now run the application using that port no localhost:8088 in 1st line

resourse>Static>html

create a new html file in static folder .html which has some word

now run again

go to bootstrap copy code to to include bootstrap

<!doctype html>  
<html lang="en">  
<head>  
 <meta charset="utf-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1">  
 <title>store</title>  
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous">  
</head>  
<body>  
<div class="container">  
 <h1 class="text-center my-4">Welcome to our website</h1>  
 <a class="btn btn-primary" href="/products">products</a>  
</div>  
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz" crossorigin="anonymous"></script>  
</body>  
</html>

Save and run now there is text title and button

If we click button it will show error because we did not create the url /products

<a class="btn btn-primary" href="/products">

Now we need to create a new table in database called products

**MODEL: instance of this class are to be stared in relational database.**

.so we need to create a model to describe this product table .

Now create a package called model. And create a class called Products. In this class we need to add some annotation.

1st annotation is **@Entity** because this class is the model that allows us to create the table and database and because I want to call the table and database products .

we need this second annotation **@Table** allows us to define the name of the table and database which is products.

**@Id** @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
is the primary key. And then it will be incremented automatically.

//attributes

private int id;  
private String name;  
private String brand;  
private String category;  
private double price;

**@Column(columnDefinition = “TEXT”)**

If we does not use colum type then the type of down column will be varchar

Now generate getter settor method using **@Getter** and **@Setter** annotation using Lombak .

Now we save the file the product table will be created this is because in application property we have spring.jpa.hibernate.ddl-auto=update

This means that when we update the model of this project the database will be updated.

**Images file** should be saved in **resources.**

Check [http://localhost:8088/images/Apple 2023 MacBook Pro.jpg](http://localhost:8088/images/Apple%202023%20MacBook%20Pro.jpg)

Result with image url <http://localhost:8088/images/Apple%202023%20MacBook%20Pro.jpg>

**REPOSITORY:** **allows us to read and write the products table in database**

Now lets create the repository that allows us to read and write the products table in database.we can call repository as service so we can name the folder as service.

Now we need create a new **interface** called **ProductRepository** in **service** package.it should extends JpaRepository<Product,Integer> here Product is model class name and Integer is model class primary key datatype. ProductRepository helps to read and write products from database.

**Controller: allows us to perform CRUD operations on products.**

Create **controller package** and in it **ProductController Class** include **@controller** annotationsbecause we need to access to this **controller url** that starts with **/product** now we need to add **@Request Mapping.**

@Controller  
@RequestMapping("/products")  
public class productController {  
 @Autowired  
 private ProductRepository repo;

Here using auto wire we requested productRepositopry from service package or container

Create Method that allows us to read the products from database.

public String showProductList(Model model) {: This line declares a method named showProductList that returns a String. It takes a parameter of type Model named model. In Spring MVC, the Model object is used to pass data between the controller and the view.

The showProductList method is the controller method.

The view template file (index.html) is the view.

List<Product> products = repo.findAll();: This line retrieves a list of all products from the repository (repo) using the findAll method. The repository (repo) is presumably an instance of a repository interface (e.g., ProductRepository) that extends Spring Data's CrudRepository or similar interface. The findAll method fetches all records from the database associated with the Product entity and returns them as a list.

model.addAttribute("products", products);: This line adds an attribute named "products" to the Model object (model). The attribute's value is set to the list of products retrieved in the previous step. This makes the products list available to the view layer, allowing it to be rendered in the UI.

return "products/index";: This line specifies the logical view name that the controller method is responsible for. In this case, the logical view name is "products/index". The controller will look for a view template named index.html (or similar, depending on the view resolver configuration) in the products directory under the configured view resolver's template directory. The actual view template file will render the data passed in the Model object, allowing the user to view the list of products.

This controller method should be accessible using http Get method that’s why we are using **@GetMapping** it is accessible by the **url** “**/products” or “/products/”**

@GetMapping("")  
public String showProductList(Model model){  
 List<Product> products = repo.findAll();  
 model.addAttribute("products", products);  
 return "products/index";

Now lets create a package called products inside this create the index.html file where this page will return to.that should be available in template folder. Copy the content in the index.html created before and paste it and alter it. As product <h1> <button> create product **url** /products/create.

Product/index – **to display**

After this create a table and its column headings

<table class ="table">  
 <thead>  
 <tr>  
 <th>ID</th>  
 <th>Name</th>  
 <th>Brand</th>  
 <th>Category</th>  
 <th>Price</th>  
 <th>Image</th>  
 <th>Created At</th>  
 <th>Action</th>  
 </tr>  
 </thead>

</table>

Add column body

Now fill the colum body using

<tr th:each="product : $(products)">

Here **th:each** helps to iterate thourgh each product in list. **product** is a object reference to Product Class to acces its methods and properties.**$products** is list in controllerProducts List products

<td th:text="${product.id}"></td>  
<td th:text="${product.name}"></td>  
<td th:text="${product.brand}"></td>  
<td th:text="${product.category}"></td>  
<td th:text="${product.price}"></td>  
<td>  
 <img th:src="@{'/images/' + ${product.imageFileName}}" alt="..."  
 witdh="100">  
</td>  
<td th:text="${product.createdAt.toString().substring(0,10)}"></td>  
<td style="white-space:nowrap">  
 <a class="btn btn-primary btn-sm"  
 th:href="@{/product/edit(id=$product.id)}">Edit</a>  
 <a class="btn btn-danger btn-sm"  
 th:href="@{/product/delete(id=$product.id)}" onclick="return confirm('Are You Sure?')">Delete</a>  
</td>

Displaying each row id name brand category price imaged with url image date and another colum with two buttons edit using url edit with their id and delete using url delete with their id now run the program the inserted data will be displayed in product/index.html page

The product will be displayed in ascending to display the new products front use

List<Product> products = repo.findAll(Sort.*by*(Sort.Direction.*DESC*, "id"));

In controllerProducts class.

/create-**To create**

To create and update products we need to create an new model that allows the user to sumit the product details it is called DTO model or data transfer object model it will be similar to the product model we created but it should not include the id and instead of the imageFileName which is string we need the file itself.

Now create the new model called ProductDto class in model package ,then we need to create the different fields of this class.

Now in product Controller we need to create a new method that dispalys a new form that allows user to create new products.in ProductController.java

@GetMapping("/create")  
public String showCreatePage(Model model){  
 ProductDto productDto = new ProductDto();  
 model.addAttribute("productDto", productDto);  
 return "products/createProduct";  
}

Now we create the html file that this method return to createProduct.html in templates >product>createProduct copy paste the content in index.html.

Delete the head and button and replace with the row that contains one column.

<div class="container">  
 <div class="row">  
 <div class="col-md-8 mx-auto rounded border p-4 m-4">  
 <h2 class="text-center mb-5">New Product</h2>

<form method="post" enctype="multipart/form-data"  
 th:object="${productDto}">

Create the form this form is submitted using the post method and we need enctype cause we will upload the image file then we will bind the productDto object that we will receive from the controller to this form **th:object**  here we have the thyme leaf attribute then lets create the first row of this form

<div class="row mb-3">  
 <label class="col-sm-4 col-form-label">Name</label>  
 <div class="col-sm-8">  
 <input class="form-control" th:field="${productDto.name}">  
 </div>  
</div>

Her we have a label = Name and a input field which is bound to productDto.name

<p th:if="${#fields.has error('name')}" th:errorclass="text-danger"  
th:errors="${productDto.name}"></p>  
</div>

This paragragh is used to display any **validation error** related **to productDto.name** so if we have **errors related to name** then we will display **the validation error** related to name  **text danger** is to display the para in red color.

Then lets create the rows of this form.

Same for brand.

For categories we option to select

Same for everything

Now run the program welcome page>products button>(/products)method getmapping return >createproducts button>(/product/create>)>submit error because we did not create method to create the post request lets now create post mapping method

@PostMapping("/create")  
public String createProduct(  
 @Valid @ModelAttribute ProductDto productDto,  
 BindingResult result){  
 return "redirect:/products";  
}

CreateProduct method to create the product that will accessible by the url(/product/create )using te post method which return the user to redirect to /products that is the list of products page.

This method requires an object of productDto which is the object that bound to the form.this productDto object will be filled using submit button of the form.

**@valid**  annotation is to validate the Data of productDto object

**BindingResult** is to check whether it has any validation error we need to add parameter of type bindingResult we can call it **result**.This result object allows us to check whether productDto has any validation error with the data available in productDto.

In **ProductDto.java** we didn’t add any validation for image file so if we need to add it manually in controller .(private MultipartFile imageFile;)

if(productDto.getImageFile().isEmpty()){  
 result.addError(new FieldError("productDto","imageFile","The image file is required"));  
 }  
if(result.hasErrors()){  
 return "products/createProduct";  
}

now create the products now you will redirect to the product list page but updated product doesn’t displayed because we didn’t include it in database to include