**CREATION:**

-New Spring Boot Project

-Group and package name must be the same: **com.lorrained.projectname**

-Maven, War, 17

-Spring Boot Dev Tools, Spring Web, MySQL Driver, Spring Data JPA, Jbcrypt, Spring-boot-starter-validation

-Add a **controllers** package: **com.lorrained.projectname.controllers**

-Add a **models** package: **com.lorrained.projectname.models**

-Add a **repositories** package: **com.lorrained.projectname.repositories**

-Add a **services** package: **com.lorrained.projectname.services**

-Add **WEB-INF folder** in src/main/webapp. Any .jsp files go within here.

**ADD INS:**

**-In src/main/resources/applications.properties, add:**

# Where are jsp files? HERE!

spring.mvc.view.prefix=/WEB-INF/

# Data Persistence

spring.datasource.url=jdbc:mysql://localhost:3306/<<YOUR\_SCHEMA\_NAME>>

spring.datasource.username=root

spring.datasource.password=root

spring.jpa.hibernate.ddl-auto=update

# For Update and Delete method hidden inputs

spring.mvc.hiddenmethod.filter.enabled=true

**-In dependencies :**

<!-- DEPENDENCIES FOR STARTING SPRING PROJECTS-->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-tomcat</artifactId>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<!-- DEPENDENCIES FOR DISPLAYING JSPS AND USING JSTL TAGS -->

<dependency>

<groupId>org.apache.tomcat.embed</groupId>

<artifactId>tomcat-embed-jasper</artifactId>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

</dependency>

<!-- DEPENDENCIES FOR INTEGRATING SQL DATABASE AND USING JPA -->

<!-- Note: Project will not run until a schema has been created and the

proper settings in application properties are present for

connecting to a database. -->

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<!-- DEPENDENCY FOR USING VALIDATION ANNOTATIONS -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>

</dependency>

<!-- DEPENDENCY FOR USING BCRYPT -->

<dependency>

<groupId>org.mindrot</groupId>

<artifactId>jbcrypt</artifactId>

<version>0.4</version>

</dependency>

<!-- DEPENDENCIES FOR BOOTSTRAP -->

<dependency>

<groupId>org.webjars</groupId>

<artifactId>webjars-locator</artifactId>

<version>0.46</version>

</dependency>

<dependency>

<groupId>org.webjars</groupId>

<artifactId>bootstrap</artifactId>

<version>5.2.3</version>

</dependency>

**-In .jsp files:**

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

<!-- c:out ; c:forEach etc. -->

<%@ taglib prefix = "c" uri = "<http://java.sun.com/jsp/jstl/core>" %>

<!-- Formatting (dates) -->

<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>

<!-- form:form -->

<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>

<!-- for rendering errors on PUT routes -->

<%@ page isErrorPage="true" %>

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Tacos</title>

<link rel="stylesheet" href="/webjars/bootstrap/css/bootstrap.min.css">

<link rel="stylesheet" href="/css/main.css"> <!-- change to match your file/naming structure -->

<script src="/webjars/bootstrap/js/bootstrap.min.js"></script>

<script type="text/javascript" src="/js/app.js"></script><!-- change to match your file/naming structure -->

</head>

<body>

</body>

</html>

================================================================================================

**FORM:FORM**

<form:form action="(the route it will go to)" method="(type of method/Get or POST)" modelAttribute="(name of the variable you gave MA in your route controller)">

<form:label path="(must be the same name as the attribute in your model you are referencing)"></form:label>

<form:input type="(type of input, like text" path="(must be the same name as the attribute in your model you are referencing)"></>

</form:form>

**C:FOREACH**

<c:forEach var="(references the single item in list)" items="${(name of what is being passed over from the controller Model model)}">

</c:forEach>

================================================================================================

**MODELS**

@Entity

@Table(name="(name of table)" ///plural

public class (Modelname) { ///singular

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

///other types of attributes. Use @NotNull and @NotEmpty above appropriate types

///You can use @Size(min= , max= , message="") for validations and requirements

@Column(updatable=false)

@DateTimeFormat(pattern="yyyy-MM-dd")

private Date createdAt;

@DateTimeFormat(pattern="yyyy-MM-dd")

private Date updatedAt;

public (Modelname) { }

@PrePersist

protected void onCreate(){

this.createdAt = new Date();

}

@PreUpdate

protected void onUpdate(){

this.updatedAt = new Date();

///make sure to add getters & setters as well

}

***\*\*RELATIONSHIP ADDITIONS\*\****

***(IN THE ONE SIDE MODEL)***

@OneToMany(mappedBy="(model that is the ONE, singular and lowercase)", fetch = FetchType.LAZY)

private List<(the model that is MANY, capitalized & singular to reference the model)> (pluralized & lowercase);

***(IN THE MANY SIDE MODEL)***

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name="(the model that is the ONE, lowercase and singular\_id)")

private Dojo dojo;

================================================================================================

**REPOSITORIES (interface)**

@Repository

public interface (ModelnameRepository) extends CrudRepository<Modelname, Long>{

List<Modelname> findAll();

}

================================================================================================

**SERVICES**

@Service

public class ModelnameService {

@Autowired

private ModelnameRepository modelnameRepo;

///query functions next

///////////////////////////EXAMPLES////////////////////////////////////

// returns all the books

public List<Book> allBooks() {

return bookRepo.findAll();

}

// creates a book

public Book createBook(Book b) {

return bookRepo.save(b);

}

// retrieves a book

public Book findBook(Long id) {

Optional<Book> optionalBook = bookRepo.findById(id);

if(optionalBook.isPresent()) {

return optionalBook.get();

} else {

return null;

}

}

}