Link for all spring boot:

https://github.com/in28minutes/spring-boot-master-class

1. @SpringBootApplication annotation : it initializes Spring ( Component scanning) and Spring Boot (Auto Configuration)
2. If we give “ApplicationName.class” to SpringApplication.run it will run that specific application.

SpringApplication.run launches a Spring Boot application.

1. Application.properties file is the configuration files eg we can change the port no in it.
2. We started a spring application using Spring initializer. We create a web application. Then we import this project in to eclipse as a maven project and final we ran the project.
3. We get “**Whiteable error page**” error if we search for the url which is not there is our application. This is default error page for spring boot.
4. If we are including devops tools dependency in our project, there while saving any changes the server is restarted automatically in spring boot.

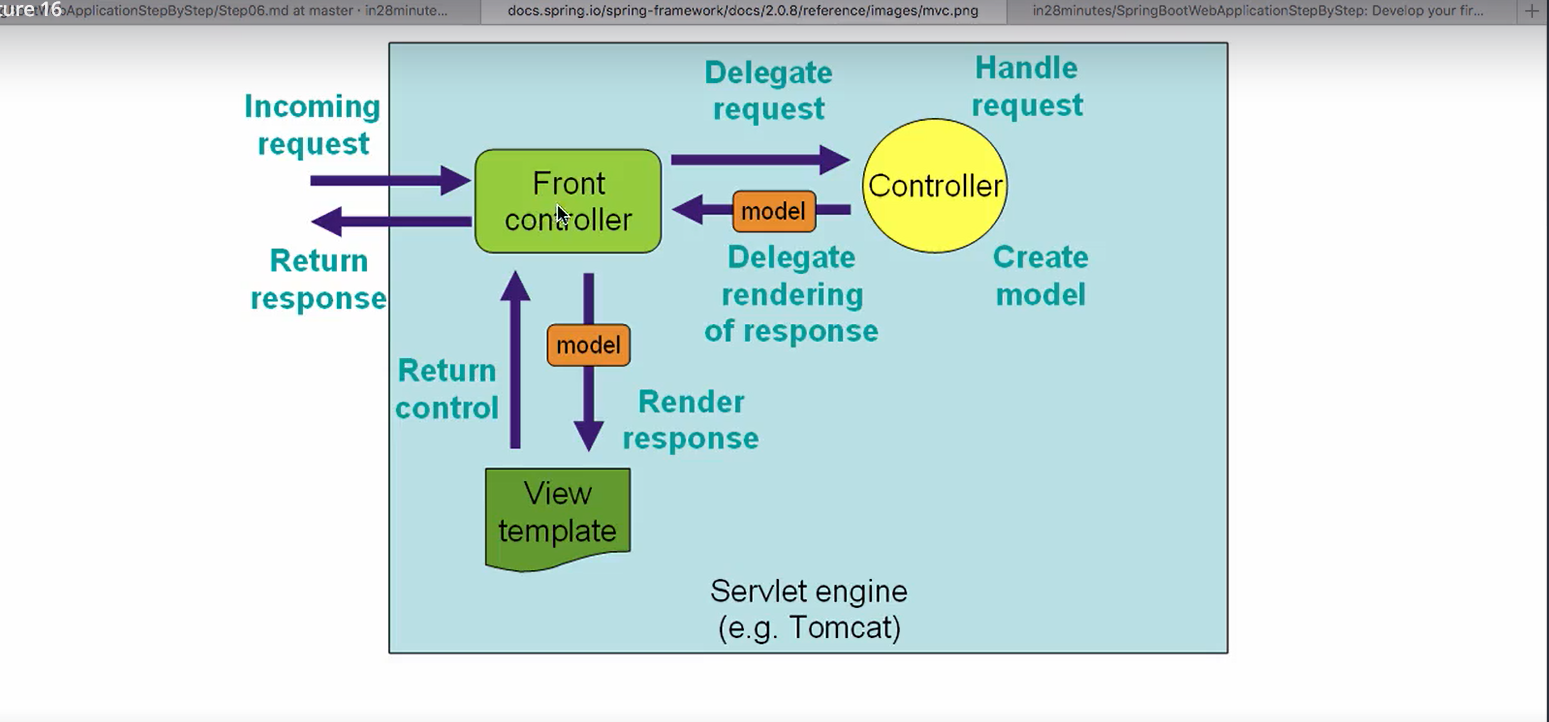
We have below points that we have used in spring boot:

1. **Spring-boot-starter-parent** : All the dependencies needed by spring application
2. **Spring-boot-starter-web :** all the core, web, web-mvc and embedded tomcat dependencies. This “Spring-boot-starter-web” dependency **autoconfigures** the Dispatcher Servlet in spring boot application if present in pom.xml
3. **Dev tools :** If we change the dynamic contents in project the project will be automatically restarted in spring boot. eg it will check the java resource, java main not the static contents like jars if we add some more depeneceny for eg for supporting embedded tomcat for jsp we need to include the “tomcat-embed-jasper” artifact in pom.xml and then restart the server as it wont take automatically.
4. **Embedded tomcat** : it is included when we inclu de starter-web dependency in pom.xml. so the application will be deploy along with the server and can be used on other machines as well.

**Spring MVC Request Flow:**

1. **Dispatcher servlet receives HTTP request.**
2. **Dispatcher servlet identifies the right Controller based on the URL.**
3. **Controller executes Business Logic.**
4. **Controller returns a) model b) View Name back to Dispatcher Servlet.**
5. **Dispatcher servlet identifies the correct view (viewResolver) .**
6. **Dispatcher Servlet makes the model available to view and executes it.**
7. **Dispatcher servlet return HTTP Response back.**

Go to docs.spring.io/spring-framewok/docs/2.0.8/reference/images/mvc.png



**Session from Manoj Behl**

Spring boot resolves all the dependency issues. We can added the parent-starter tag. It has embedded tomcat and jetty web servers. We can create the jar file directly and run it using command prompt. There is no need for xml files anymore all it use is java configuration. There is no web.xml file. Spring boot has innovative idea to include the single version dependency no need to worry about the compatibility of the dependency and jars.

Everything will be deployed as Jars in spring boot.

Everything to learn from :

**Spring doc : spring reference**

Spring boot has **application.properties** file for any configuration related to database, JMS, logs level any message anything

Contuning spring boot from udemy :

Various annotations :

1. Component,service,repository and controller
2. Autowired
3. componentScan
4. When we put @Component, @service, (@controller used on web controller) annotation on any bean, it means we are requesting spring to manager it.

@controller, @service and @Repository are the specialization of the @component which is genric annotation

in spring

@controller is specialization of @Component used in the context of web MVC.

@Service is used on the bean having business logic.

@Repository is mainly used in case of data store. In terms of Jpa storing the data in database.

1. If we create the instance of loginService in controller and have not used @Autowired it will throw null pointer exception. If we put this annotation it is managed by spring.
2. @SpringBootApplication has inbuilt @ComponentScan which will scan the package in which Spring boot starter class is present. If we want to scan other packages as well then we need to add @componentScan(“”packagename”) annotation below @SpringBootApplication otherwise will get application failed to start error.

**Spring framework is all about finding Beans using various annotations like @component, @service etc and manage their lifecycle, inject them in another beans using @Autowired.**

**==================================================================================**

**Session, Request and model**

**If we have some model attribute it is not available to subsequent request by default. For the model attribute to be available for the other request i.e conversational stage we need to used**

**@SessionAttribute(“name”) annotation on all the controller class which needs that model data to store the model for subsequent request.**

**HTTP is a stateless protocol and one request values are not available for other request. so to store the state we need to store the model in the session at server side.**

**We need to be cautious to add the amount of data in the session.**

**==================================================================================**

**JSTL in JSP page :**

**If we want to use the JSTL in JSP page for listing in table will include the below dependency in pom.xml and then include the jstl directive as below**

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

</dependency>

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

**==================================================================================**

**BootStrap for Page formatting using webJars:**

**WebJars upgraded the lastest version for static files that are css files and js files.**

**To be able to used Bootstrap jquery is needed. That’s why we need to include both the jars as below.**

<dependency>

<groupId>org.webjars</groupId>

<artifactId>bootstrap</artifactId>

<version>3.3.6</version>

</dependency>

<dependency>

<groupId>org.webjars</groupId>

<artifactId>jquery</artifactId>

<version>1.9.1</version>

</dependency>

**It’s a good practice to include the js file just above the body end tag </body> and css file below the title in the <head> in JSP page tag as mentioned below.**

<script src="webjars/jquery/1.9.1/jquery.min.js"></script>

<script src="webjars/bootstrap/3.3.6/js/bootstrap.min.js"></script>

<link href="webjars/bootstrap/3.3.6/css/bootstrap.min.css"

rel="stylesheet">

**To use bootstrap we need one ccs file which is bootstrap.css and two js file which are boostrap.js and jquery.js**

**So, finally we have include dependency for jquery and bootstrap. We use webjars for them.**

**We made use of some of the below classes in boostrap for formatting**

**Class=”container”**

**Class=”table table-striped”**

**Class=”button”**

**<button lass=”btn btn-success”>**

**Class=”btn btn-warn”**

<fieldset class=*"from-group"*> **–inside the form tag**

**Calss=”form-control” in each html element**

**BootStrap is basically a css framework which helps us to format our stuff very very well.**

**VALIDATION :**

**We also added html 5 validation at client side i.e. required=”required” to show the error if the input is not available.**

**It’s a good practice to put server side validation always. We will use hibernate validator for this.**

**Steps to Implementing server side validation:**

1. **Command bean or form backing bean**
2. **Add validation**
3. **Use Valiadtion on controller**
4. **Display errors in view**

**Creation of Command Bean :**

1. **On controller we include todo in place of desc**
2. **View – spring form tags we used and include modelAttribute=”todo” in <form:form>**

**Advantage of Form backing bean or Command Bean**

1. **Double Binding.**
2. **Bean 🡪 Form**
3. **Form -> Bean**

**@valid annotation we need to provide on the Todo Class for validation and provide the annotation based validation on bean class. The BindingResult is used to validate if the validation is fine or failed.**

**And will user <form:errors path=”desc” />**

**For the jsp side**

**If we want to add Target date field will get InvalidArgumentException as the application is trying to convert string to date. So ill use @InitBinder which will initializes the web databinder. This binding from string to date is happened through web databinder.**

**Below code will include in controller class:**

@InitBinder

//This annotation is used to bind the string value to the date and initializes the web databinder

**public** **void** InitBinder(WebDataBinder binder) {

//user will input the format dd/MM/yyyy

//we are creating a simple date fomat as dd/MM/yyyy here

SimpleDateFormat dateFormat = **new** SimpleDateFormat("dd/MM/yyyy");

//whenever we need to input date will input in this format and bind with Date.class

binder.registerCustomEditor(Date.**class**, **new** CustomDateEditor(dateFormat, **false**));

}

Using below tag library we can include format date in jsp side also:

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

<td><fmt:formatDate value=*"*${todo.targetDate}*"* pattern=*"dd/MM/yyyy"* /></td>

If we want to add the datepicker we need to include below dependency in pom.xml and add the script in the body

<dependency>

<groupId>org.webjars</groupId>

<artifactId>bootstrap-datepicker</artifactId>

<version>1.0.1</version>

</dependency>

Set the path for the added jars in the jsp file body tag

<script

src=*"webjars/bootstrap-datepicker/1.0.1/js/bootstrap-datepicker.js"*></script>

script to add date picker

<script>

$('#targetDate').datepicker({

format : 'dd/mm/yyyy'

});

</script>

Same format as above with explanation

<script>

$('#targetDate').datepicker({ //jquery syntax to take the targetDateField id using #

format : 'dd/MM/yyyy' // initialize the datepicker with the mentioned format

});

</script>

**SPRING BOOT END TO END APPLICATION**

**To create the persistent layer(data access layer)**

1. **We need to write the model class/entity class using JPA annotations to map the database table;**
2. **Create the table in database;**
3. **Create the Repository interface by extend the JPA/CRUD repository in spring boot that will provide all the crud operations .**
4. **Configure the datasource in the application.properties file i.e for mysql/oracle database whatever we are using.**
5. **Write the unit test for all the 4 operations to test the operations are fine with database or not.**

**……………………………………………………………………………………………..**

**To Code the Utility layer for Email Functionality in our application.**

**HOW TO CONFIGURE THE SPRING SECURITY USING SPRING-SECURITY**

1. **WE inject the Authenticationmanager to authenticationServiceImpl, which has the authenticate method for authentication and providing token using username and password. The isAuthenticated() of it will return true or false if the user is authenticated or not. We will store that token for subsequent requests in SecurityContextHolder.getContext().setautherntication(token).**
2. **Using the UserDetails Class we will loadUserName from database and then pass the UserDetails in spring into class UserNamePasswordAuthenticationToken(userDetails, password, userDetails.)**
3. **This authenticationManager bean is created by spring in some versions automatically. In spring bot 2.0.0 and 3.0.0 versions this is not created automatically that’s why we are creating it manually using @Bean.**

**TO Deploy the Spring Boot application as JAR/WAR file.**

**To deploy it as jar, when need to deploy on web server like Tomcat/Jetty**

1. **Run maven clean**
2. **Run maven install/build/package.**
3. **Check the .jar is built under / target directory and we can deploy this .jar using**
4. **Java -jar applicationname.jar (as tomcat is in built here)**

**To deploy it as WAR file i.e. when we need to provide the war file for our application so that it can be deployed on application servers like websphere application server or weblogic server.**

1. **Go to pom.xml, make** <packaging>war</packaging> as WAR
2. **Then Go to the Boot class, i.e. FlightreservationApplication. It should extends the SpringBootServletInitializer class to support the servlet 3 filter and override the configure() that takes SpringApplicationbuilder as the input parameter and returns the class that annotated with @SpringBootApplication and that will be the starting point of application.**

**Update the maven project.**

**Do maven clean**

**Run as maven install.**

**Now our project is bundle as the war file this time.**

**WHEN TO USE MICROSERVICE :**

1. **Organizational alignment**
2. **Release functionality faster**
3. **Independent scaling**
4. **Easier to focus on security concerns**
5. **Adopt technology faster**
6. **Embrace Uncertainty In digital**