**Spring Boot**

***Class: 1***

Questions:

1. What is the advantage of Spring-boot-starter dependency?
2. What are the embedded servers & databases given by spring boot?
3. What is spring boot autoconfiguration?
4. What is the advantage of Spring boot Actuators?

**Limitation of Spring:**

1. we need to hunt for all the compactable spring version jar

we need to add all the jars in build-path / class-path

1. we need Download & install external the database & server.

we need to deploy the application into external servers.

1. In spring we need to configure, DispatherServLet , ViewResolver, DataSoruce

JDBCTemplate, peroperties

1. In spring we have the multiple beans, it is difficult to monitor

the complete application.

**Advantages of Spring boot:**

1. Spring-boot starters

starter dependency will aggregate the multiple dependencies into single dependency. in build path/class-path it is good to add only one jar

Ex: spring-boot-starter-web: MVC tomcat REST

Spring-boot-starter-test: junit, Hamcrest, mockito, Spring test

1. Spring boot Embedded databases & servers

Embedded servers: Tomcat(default), jettey, undertow

Embedded databases: H2, HSQL, Apache Derby

1. Spring boot Autoconfigurations

The beans are autoconfigured DispatherServLet, ViewResoLver,

JDBCTempLate... etc

1. Spring boot Actuators

We need to monitor health, shutdown, the application in production environment.

Logger, database, Data Source, end-points, Auto wired.

***Class: 2***

Artifact ID: Project name

Group iD : package name

version : 1.0

Type of project: Maven, gradle

**Maven:**

pom.xml: XML Configuration

<!-- https://mvnrepository.com/artifact/org.springframework/spring-core -->

<dependency>

<groupId>org.springframework</groupId> <artifactId>spring-corek/artifactId> <version>5.3.6</version>

</dependency>

**Gradle:** build.gradle : JSON configuration

// https://mvnrepository.com/artifact/org.springframework/spring-core implementation group: 'org.springframework', ,

name: 'spring-core'

version: '5.3.6'

**packaging:**

Jar (java archive file)

war (web archive file)

**Java version:** LTS (Long term Support)

java8 java11 java17 java20

**Language:** java, Groovy, Kotlin

**Spring Boot Version:**

Spring boot 1.x

Spring boot 2.x

Spring boot 3.x

**Add the dependencies:**

Spring-boot-starter-web

Spring-boot-starter-test

Spring-boot-starter-data-jpa

Q. What are the build tools?

Ans: Maven: XML Configuration 1: pom.xml

Gradle: JSON configuration: build.gradle

Q. How many types of packaging?

Ans: Jar (java archive files)

War (web archive file)

Q. Explaining maven terms

Ans: Group ID: package-name

Artifact ID: project name

Version: version-info

Q. What are the languages spring boot support?

Ans: java, Kotlin, groovy

Q. What are the LTS java version support by spring boot

Ans: java8 java11 java17 java20

Q. how to create the project in spring website

Ans: by using spring Initializer

Start.spring.io

***Class: 3***

**Spring Boot folder structure:**

1. src/main/java ----> create the java classes Here

Controller, Services, Repository, entity, Exception, model

2. src/main/resources ---> configurations

src/main/resources/template

Here we can the HTML files

src/main/resources/static

here CSS, image and

application.properties: Here we will take the configurations port, databaseinfo, security, log....

3. src/test/java

to write the unit test cases

ex: Junit, mockito, Hamcrest, SpringTest

4. JRE system Library: java version 17

5. Maven dependency: all the jar files display here

6. target:

Once the project build completed the classes, jar and war files comes here

7. POM.xml (Project Object Model)

Which contains the project meta data

Version, dependency, java version details.

8. View Layer:

1. JSP View Layer:

Src/webapp/WEB-INF/myviews/first.jsp

2. Thymeleaf

HTML files under the templates

***Class: 4***

**Controller:**

@controller annotation indicates that an annotated class is a Controller in MVC architecture.

@Controller

@ResponseBody // (instead of using this annotation to every method simply we can use to class level.

**public** **class** TestController {

@RequestMapping("/test") // url name can be anything ex: test

//@ResponseBody // --> it helps to make the return string visible on browser

**public** String greet() // method name can be anything ex: greet()

{

**return** "Good Morning Rohith, like comment and subscribe";

}

@RequestMapping("/info")

//@ResponseBody

**public** String info() {

**return** "2nd text view to using responsebody annotation.";

}

}

Note: In MVC the controller class request handler methods return value is by default view name.

But in the above example just we need to return the test response, for that we need to use @ResponseBody annotation for that method or if we have all the same methods required this annotation in this Test Controller class we can add the @ResponseBody annotation to the class level instead of adding annotation at method level for each method in class.

***Class: 5***

**Create the JAR:**

Create the jar file for the above application.

Now run as 7 maven install

Now it will download some files automatically, you can see those in consol.

Once we see: **BUILD SUCCESS** on the console we need to refresh the project explorer by right click on the project name then choose refresh, or simply use F5 button to refresh.

Now some jar files are created in target folder of the project.

File: Spring-MVC-yt-Ex-1-0.0.1-SNAPSHOT.jar

Folder Location: "E:\Sathya Technologies\Ratan IT SpringBoot yt\Springboot CODE prac yt\Spring-MVC-yt-Ex-1\target\Spring-MVC-yt-Ex-1-0.0.1-SNAPSHOT.jar"

And .m2 folder is also created in user folder in c:/ drive

Ex Location: "C:\Users\rohit\.m2"

**Delete the JAR:**

If we want to delete the jar files, we need to delete the .m2 folder and the jar files manually by going to their location

Once we deleted those files and folder, we need to run the application with 5 maven clean

Then we can re start the process to create the jar file again.

Close the IDE, we don’t need to run the IDE for server conection.

**Run the jar in CMD:**

Go to the target folder location in file explorer type cmd in location navigation bar then you will get the command prompt open with the target folder location by default.

Now enter the code like this

**Java -jar jar\_file\_name.jar**

Java – jar Spring-MVC-yt-Ex-1-0.0.1-SNAPSHOT.jar

Now the same way how console give the details same way command prompt will also give the details

Go to the browser and search

<http://localhost:1234/test>

this is the localhost server where our project display

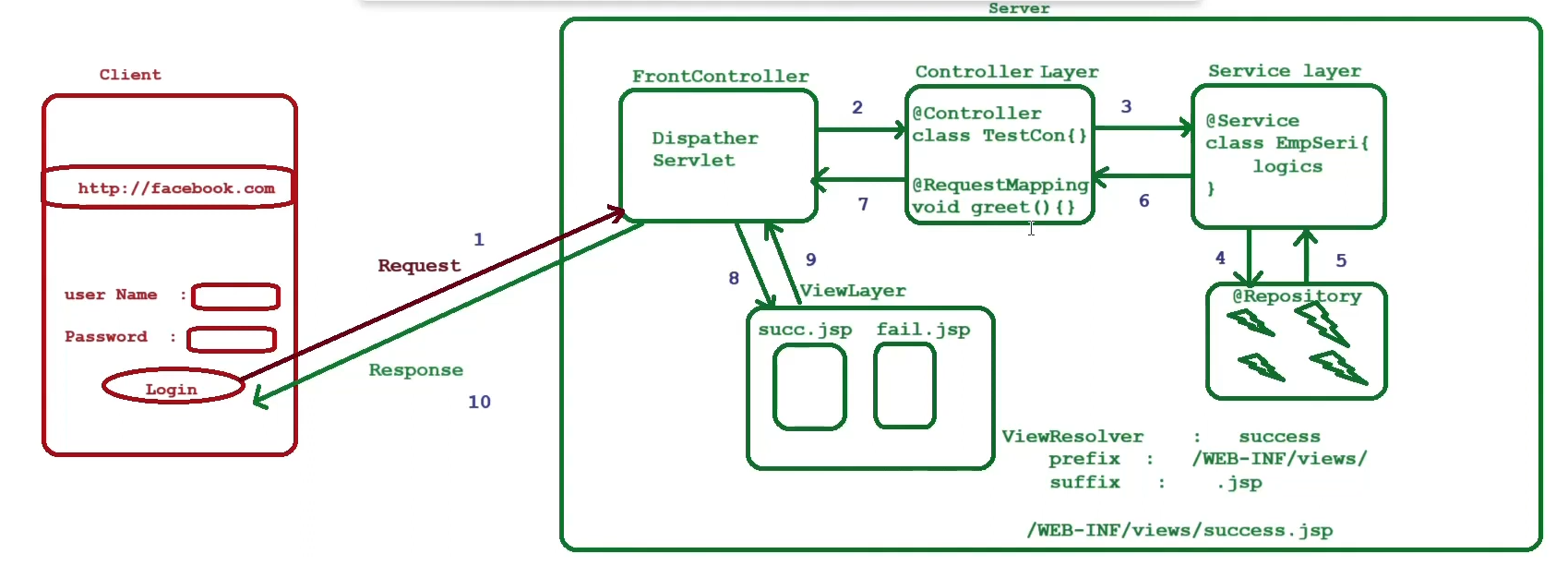
1234 is the port number of our project

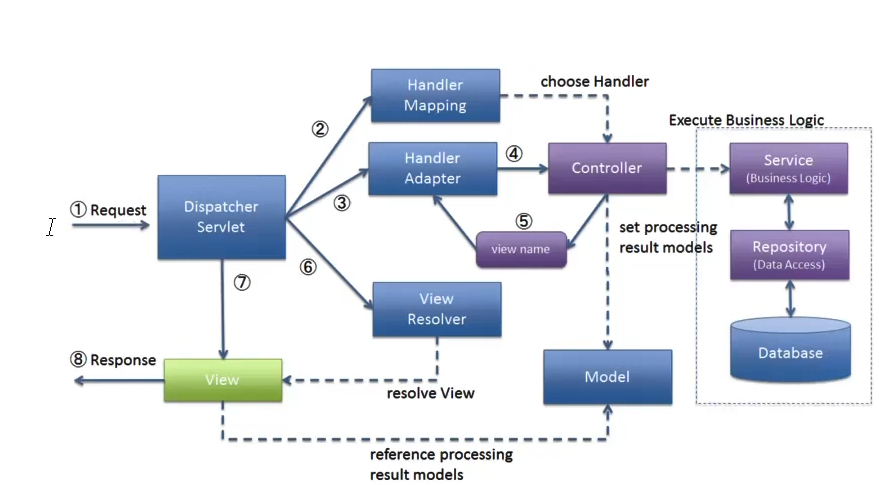
Test is the URL of the String value to display on the screen.

***Class: 6***

**MVC Architecture**

**MVC: Model View Controller**





The controller return value is by default view name.

How the controller layer sends the data to view layer to display:

Model

ModelAndView

View Layer:

1. JSP views
2. Thymeleaf

Q. what is front controller?

Ans: the request & response pass through the front controller.

DispatcherServlet will act as FrontController.

In Spring we should configure the FrontController, but in spring boot it is autoconfigured.

Q. What is controller & service & repository?

Ans: The controller layer contains request handler methods to handle the request flow.

@Controller annotation represents the controller.

The Service layer contains the business logics. @service annotation represent the service layer.

The Repository Layer contains the logics interacting with the database.

Q. what is the return value of the controller?

Ans: the controller return value is by default view name. (JSP file name).

The FrontController will take the view-name, with the help of ViewResolver (prefix and suffix) it will identify the complete view path and render the view to browser.

Q. what are the view layer technologies?

Ans: JSP view layer

Thymeleaf view layer

Note: spring boot the default template engine is thymeleaf.

Q. how the controller gives the result to view layer to display?

Ans: In two ways

1. Model: we can have only data
2. ModelAndView : we can have data & view name

Q. if the Application contains multiple controllers. How it identifies the request URL belongs to specific controller?

Ans: The Primary purpose of Handler Mapping is to determine which controller should process for incoming http request. It defines the mapping between incoming request and specific handler methods.

The handlerAdaptor Invoking (call) the handler method that has been selected by the handler mapping. It deals with the execution of the handler methods.

***Class: 7***

Boot:

* JSP views
* Thymeleaf
* Freemarker
* Velocity

Spring boot default template engine: Thymeleaf

Q. How the JSP file path identified by springboot?

Ans: to the view name it will add suffix & prefix to identify the complete path in application.properties for “greeting” view

Spring.mvc.view.prefix=/WEB-INF/MyViews/

Spring.mvc.view.suffix=.jsp

Ex: this will add the like like this

/WEB-INF/MuViews/greeting.jsp

**Test Controller code:**

@Controller

**public** **class** TestController {

@RequestMapping("/greet")

**public** String greetMsg()

{

**return** "greeting";

}

}

So, this controller class/ requestMapping method will return “greeting” this is the name of JSP file.

And that how JSP files is called and displayed on the web when we use this link

<http://localhost:9999/greet>

**JSP File code:**

<%@ page language=*"java"* contentType=*"text/html"*%>

<!DOCTYPE html>

<html>

<head>

<title>Greeting Page</title>

</head>

<body>

<h1> Hi students how are you</h1>

<h2>Welcome to ratan classes</h2>

<h3> like comment and subscribe </h3>

</body>

</html>

This page is displayed when we use local host URL with /greet.

***Class: 8***

Just Installing Spring Suit Tool

***Class: 9***

**Model & ModelView:**

Q. how the controller sends the data to view layer?

Or

What is the purpose of Model & ModelView object?

Ans: there are two ways the controller can send the data view layer,

1. Model: here we can have only data
2. ModelAndView: here we can have data & viewname.

Q. what is the purpose of springboot dev tools

Ans: it will auto reload the application

Provides fast application restarts.

Q. what is @RequestMapping?

Ans: this annotation can be used both at the class and method level.

In most cases, at the method level applications will prefer to use one of the HTTP method specific variants @GetMapping, @PostMappling, @PutMapping, @DeleteMapping, or @PatchMapping.

Q. what is the difference between get & post?

Ans: @GetMapping: used to get the data from server.

@Postmapping: used to send the data to server.

**Model:**

@Controller

**public** **class** TestController<modelAndView> {

// 1. model

@GetMapping("/data")

**public** String getMessage(Model model) {

model.addAttribute("message1","Good morning");

model.addAttribute("message2","like comment");

**return** "data";

}

**JSP file: Model**

<%@ page language=*"java"* contentType=*"text/html"*%>

<!DOCTYPE html>

<html>

<head>

<title> Model Example </title>

</head>

<body>

<h1>your Message1... ${message1}</h1>

<h2>your Message2... ${message2}</h2>

</body>

</html>

**Modelview:**

@GetMapping("/studentinfo")

**public** ModelAndView info () **throws** Exception

// instead of creating an object you can give parameter to the method

Ex: **public** ModelAndView info (ModelAndView modelAndView)

// anything can work choose any one of two types.

{

ModelAndView modelAndView = **new** ModelAndView();

modelAndView.addObject("name", "Rohith");

modelAndView.addObject("age", 24);

modelAndView.addObject("email", "rohithsomella@gmail.com");

modelAndView.addObject("mobile",9948617330L);

List<Integer> marks = List.*of*(98,67,45,98,47,36);

modelAndView.addObject("marks",marks);

modelAndView.setViewName("info");

**return** modelAndView;

}

**JSP file: ModelView**

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

pageEncoding=*"UTF-8"*%>

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

This above line we need to write for ModelView

<!DOCTYPE html>

<html>

<head>

<title>ModelAndView Example</title>

</head>

<body>

Student Name: ${name} <br>

Student Age: ${age}<br>

Student Email: ${email} <br>

Student Mobile: ${mobile}

<!-- To iterate the list of marks use for-each loop in JSTL -->

<c:forEach items=*"*${marks}*"* var=*"marks"*>

<h2>Your marks data: ${marks}</h2>

</c:forEach>

</body>

</html>

In for each loop items= attribute name of ModelAndView Object var= list name

${list name} to print in h2 tag.

***Class: 10***

Lombok dependency used to reduce the boiler code and make us archive loose coupling

It helps us to reduce the code of constructor, toString, getters and setters…

**The important annotation from Lombok:**

* @Data
* @AllArgsConstructor
* @NoArgsConstructor

**Persons class from model package:**

@Data

@AllArgsConstructor

@NoArgsConstructor

**public** **class** Person {

**private** **int** id;

**private** String name;

**private** String email;

}

**Controller class from controller package:**

@Controller

**public** **class** PersonController {

@GetMapping("/persons")

**public** String perList(Model model)

{

List<Person> persons = **new** ArrayList<Person>();

persons.add(**new** Person(100,"Rohith","rohithsomella@gmail.com"));

persons.add(**new** Person(101,"Charan","charansomella@gmail.com"));

persons.add(**new** Person(102,"Sachin","sachintakur@gmail.com"));

model.addAttribute("persons",persons);

**return** "persons-list";

**JSP file:**

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

pageEncoding=*"UTF-8"*%>

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

<!DOCTYPE html>

<html>

<head>

<meta charset=*"UTF-8"*>

<title>Lombok example with Model object</title>

</head>

<body>

<h1> Persons List...</h1>

<c:forEach items=*"*${persons}*"* var =*"person"*>

Person ID: ${person.id} <br>

Person Name: ${person.name} <br>

Person Email: ${person.email} <br>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* <br>

</c:forEach>

</body>

</html>

**Application properties:**

spring.application.name=Spring-MVC-yt-Ex-4

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

spring.mvc.view.suffix=.jsp

server.port=1111