SPRING BOOT MVC

Spring MVC

- Web application framework that takes advantage of design principles of Spring framework
- Flexible and extensible via component’s
- Simplified form handling through its parameter binding
- Can do validation and error handling

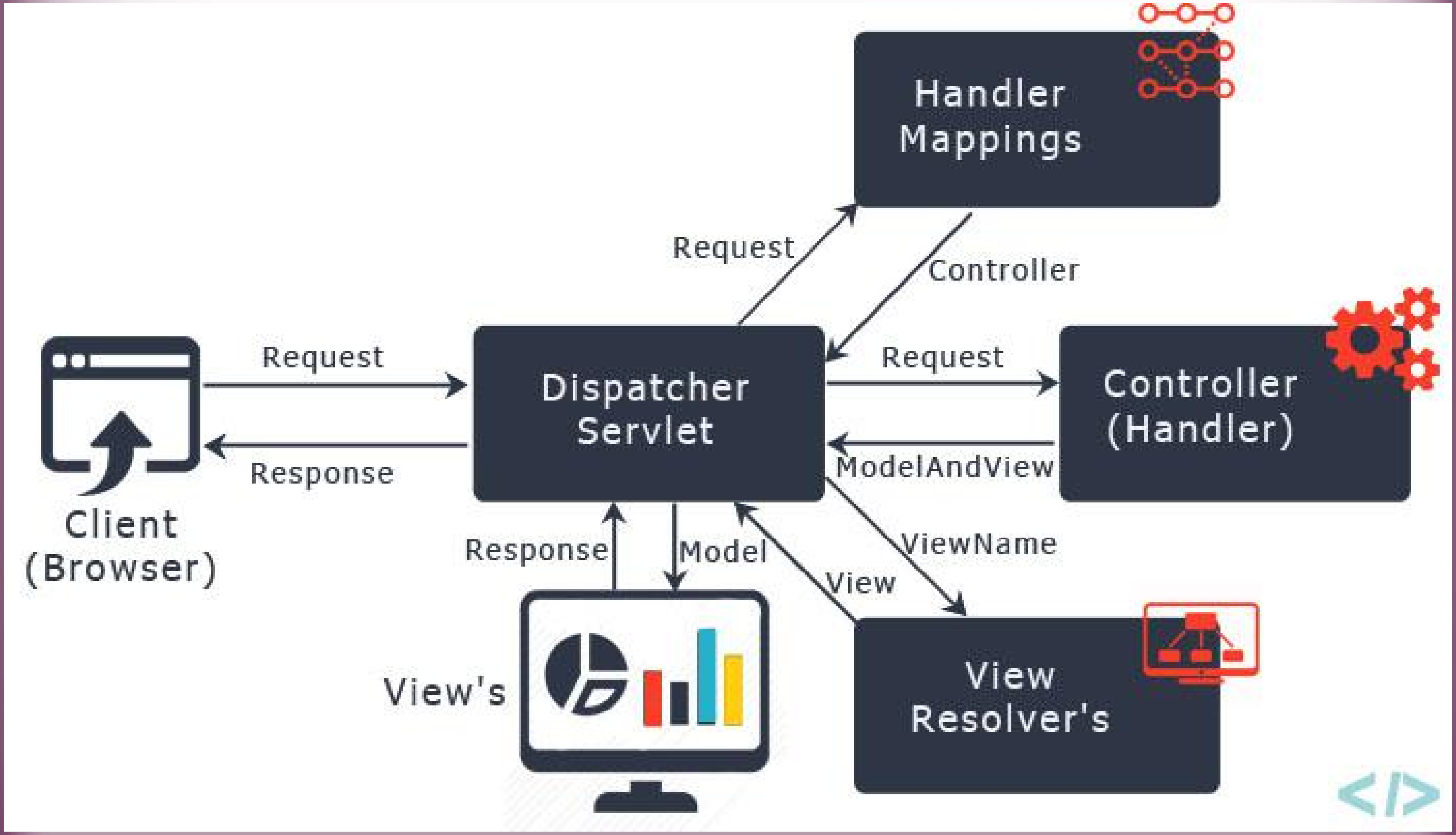
MVC Architecture

- **Controller**
  - Designed around a DispatcherServlet

- **Model**
  - Can use any object as a command or form object.
  - Data binding is highly flexible.

- **View**
  - View resolution is extremely flexible.
  - View name resolution is configurable
  - Can use properties file to configure views

The requesting processing workflow



@RequestMapping

- Used to map URLs onto a class or a particular method.
- **Class-level annotation** maps a specific request path or pattern to a controller
- Method-level annotations are used to narrow the primary mapping
  - HTTP "GET"/"POST“ methods.

```
@Controller
@RequestMapping("/api")
public class SecondController {

}
```

@RequestMapping

```
@Controller
@RequestMapping("/api")
public class SecondController {

@RequestMapping(value="/hello" method = RequestMethod.GET)

public String greet()
 {
    return "Hello User";
 }
}
```

http://localhost:8080/api/hello

Model

- A map object used to store attribute value pairs
- Its created before invoking a handler method if the method has an argument type Model.
  - Stores attribute values to render dynamic views such as JSP.
- **addAttribute**(String name, Object obj)
  - Used to map attribute names to object as attribute vales.

```
public String init(Model model) {
    model.addAttribute("majHeading", "Jeevan Blood Bank");
    return "index";
}
```

Controller

```
@Controller
public class WelcomeController {


@RequestMapping("/welcome")
   public String showLoginPage() {
      return "welcome";
   }
  }
```

View Resolver

- **InternalResourceViewResolver**
  - Used to map the logical view names to view files

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

spring.mvc.view.suffix=.jsp
```

Welcome.jsp

– **src/main/webapp/WEB-INF/pages**

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
</head>
<body>
<h2>MVC Configured Successfully</h2>
</body>
</html>
```

Dependency Required

- Update the pom.xml with following Dependencies
- The Entries can be picked from the Effective Pom.

```
<dependency>
      <groupId>org.apache.tomcat.embed</groupId>
      <artifactId>tomcat-embed-jasper</artifactId>
</dependency>

<dependency>
       <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
</dependency>
```

`ModelAndView` object

- Encapsulates both model and view that is to be used to render model
- Model is represented as a **java.util.Map**
- Objects can be added to without name:
  - **addObject(String, Object)** – added with explicit name
  - **addObject(Object)** – added using name generation (*Convention over Configuration*)
- View is represented by **String** or **View** object
- Analogous to Struts **Action**

Controller

```
@Controller
public class FirstController {



@RequestMapping("/first")
public ModelAndView execute()
{
 String message = "Welcome to Spring!";

      return new ModelAndView("Success", “msg",message);
}


}
```

Controller-Model and View

```
@Controller
public class SecondController {

@RequestMapping("/second")

public ModelAndView getMessage()    {

      ModelAndView mdl=new ModelAndView();

      mdl.setViewName("Second");

      mdl.addObject("msg","Hello India");

      return mdl;
                              }   }
```

@RequestParam

- Use to bind request parameters to a method parameter in the controller.

```
@RequestMapping ("/find")
public String get ( @RequestParam ("custId") int id,  Model model) {
    Customer cust =dao.findByCustomerId(id);
    System.out.println("Inside GET Method"+ cust);

    model.addAttribute("foundCustomer",cust);
   return "Display";
}
```

- Can also do **return new** ModelAndView("redirect:RedirectPage.htm");

@ModelAttribute

- When Used as **method parameter**, maps a model attribute to the specific, annotated method parameter

- The controller gets a reference to the object holding the data entered in the form.

- When used at **method level** provides *reference data* for the model

- The @ModelAttribute annotated methods are executed *before* the chosen @RequestMapping annotated handler method.

- This helps in pre-populating the implicit model with specific attributes,

Model

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@ToString
public class TripDetail {

private long tripId;
private String source;
private String destination;
private double amount;



}
```

Controller

```
@Controller
public class TripController {

@Autowired
TripDetail detail;

@GetMapping("/")
public String init() {
     return "index";
}
@ModelAttribute("location")
public String[] loadPlaces() {
   return new String[]{"Bessy Beach","Mahabs","Mayajal"};
}
```

Controller

```
@GetMapping("/addTrip")
public String initForm(Model model) {

model.addAttribute("command",detail);
return "addTripDetails";
}

@PostMapping("/addTrip")
public String onSubmit(@ModelAttribute("data") TripDetail
   details) {
            System.out.println(details);
return "success";
    }
}
```

Using Spring's form tag library

- Spring's form tag library gives the tags access to the command object and reference data of the controller
- The form tag library comes bundled in spring-webmvc.jar.
- The library descriptor is called spring-form.tld.
- **<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>**
- A Form tag puts the command object in the PageContext

View

```
<form:form action="/addTrip" method="post">
<form:input path="tripId"/>
<form:select path="source" items="${location}">

</form:select>
<form:input path="destination"/>
<form:input path="amount"/>

<input type="submit" value="Add">
</form:form>
```

Spring Validation

- JSR-303 Bean Validation API is used by Spring.
- The standardized validation constraint declaration and metadata
- Annotate domain model properties with declarative validation constraints and the runtime enforces them.
- Can define own custom constraints.
- To trigger validation of a @Controller input, Input arguments are annotated with @Valid
- Using Hibernate Validator in the classpath, Spring will detect it and automatically support across all Controllers

Maven Dependency

```
<dependency>
   <groupId>javax.validation</groupId>
   <artifactId>validation-api</artifactId>
   <version>2.0.0.Final</version>
</dependency>

<dependency>
   <groupId>org.hibernate</groupId>
   <artifactId>hibernate-validator</artifactId>
   <version>5.4.0.Final</version>
</dependency>
```

Hibernate inbuilt Validation

- @AssertFalse
- @AssertTrue
- @Email
- @Length
- @Range

Model

```
public class BloodDonor {



private int id;

   @Length(min = 3,max = 8)

private String name;

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

private LocalDate dateOfBirth;



private String bloodGroup;

}
```

Controller - @Valid annotation

```
@RequestMapping(“/donors")

public String initAddDonorForm(Model model) {

model.addAttribute("command",donor);

return "addDonor";
}

 @PostMapping(path="/donors")
  public String greetingSubmit(@Valid @ModelAttribute("command")
     BloodDonor donor, BindingResult result) {

 if(result.hasErrors()) {

 return "addDonor";
 } else {
    return "result";
 }
  }
```

Validation Input Jsp Page

```
<form:form method="post" action=" donors" >

ID:  <form:input path="id" />
                    <form:errors  path="id" />



Name : <form:input path="name" />
    <form:errors path="name" />

<input type="submit" value="Submit" />
```

JPA ASSOCIATION

OneToMany

- **@OneToMany**
  - In a relational database system, a *one-to-many* association links two tables based on a Foreign Key column so that the child table record references the Primary Key of the parent table row.
  - Represented either through a @ManyToOne or a @OneToMany
  - Association can be either unidirectional or bidirectional.

- **@ManyToOne:**
  - Used to map the relationship of entities
  - To map the Foreign Key column in the child entity mapping
    - so that the child has an entity object reference to its parent entity.

One to Many Mapping

- **@JoinColumn**
  - Foreign Key reference
  - Use to join the patient entity with doctor entity
  - The Patient entity is considered as owning side of the mapping , as the foreign key reference in this class
  - The Doctor entity is considered as Inverse side of the relationship
- **mappedBy**
  - Set with the value “doctor”
  - This is the previously declared Many to one Mapping field name of the Patient entity

JPA – One To Many

```
@Entity
@Table(name = "hateoas_doctor")
public class Doctor {

@Id
private int id;
private String name;
private String speciality;

   public Doctor(int id, String name, String speciality) {
      super();
      this.id = id;
      this.name = name;
      this.speciality = speciality;
}
@OneToMany(mappedBy = "doctor", cascade =
    CascadeType.ALL,fetch=FetchType.EAGER)
   private List<Patient> patientList= new ArrayList<Patient>();
}
```

JPA – One To Many

```
@Entity
@Table(name = "hateoas_patient")

public class Patient {

    @Id
    private int id;
    private String name;


    @ManyToOne(fetch=FetchType.LAZY)
    @JoinColumn(name="doctor_ref" ,referencedColumnName = "id")
    @JsonIgnore
    private Doctor doctor;

}
```

CORS configuration

- Annotation enables cross-origin requests
- Can be added at the Method or controller Class level
- @CrossOrigin(origins = "http://localhost:9000")
- By default, allows all origins, all headers, the HTTP methods specified in the @RequestMapping annotation
- It can be customized specifying the value of one of the annotation attributes:
  - origins,
  - methods,
  - allowedHeaders,

SPRING ACTUATOR

Spring Boot Actuator

- Exposes REST endpoints that can be consumed to manage and monitor application.

- **Monitor application health**, application **bean details**, **version details**, **thread dumps**, **logger details** etc

- Can restrict these endpoints to be consumed by authorized users only and Spring provides easy way to secure your REST endpoints.

actuator End Point

- **actuator**
  - It provides a hypermedia-based discovery page for the other endpoints.
  - By default it is sensitive and hence requires username/password for access or may be disabled if web security is not enabled.
- **Beans**
  - It displays complete beans configured in the app.
- **Configprops**
  - It displays a collated list of all @ConfigurationProperties.

actuator End Point

- **Health**
    - It shows application health information
  - management.endpoint.health.show-details:=**always**

- **Info**
  - It displays arbitrary application info.
- **Loggers**
  - It shows and modifies the configuration of loggers in the application.
- **metrics**:
  - It shows metrics information for the current application.
- **Mappings**
  - It displays a collated list of all @RequestMapping paths.

Actuator Endpoints

- health and info can be accessed
  - **Other endpoints are disabled by default**

- ***management.endpoints.web.exposure.include=*.***
  - Will enable all of them
  - Can also list endpoints which should be enabled

- To expose all enabled endpoints except one (for example */loggers*), we use:

```
management.endpoints.web.exposure.include=*
management.endpoints.web.exposure.exclude=loggers
management.security.enabled=false
```

Info and Health Endpoint

- Reads the Information from the pom.xml

```
info:
  build:
    artificatId: '@project.artifact@'
    groupId: '@project.groupId@'
    version: '@project.version@'
  java:
      version:@java.version@


management:
  endpoint:
    health:
      show-details: always
```

Logger Endpoint

- http://localhost:8787/actuator/metrics
- http://localhost:8787/actuator/loggers
- To Change the Log level make a **post** request

  http://localhost:8787/actuator/loggers/org.springframework.boot.SpringApplication

  ```
  { "configuredLevel": "trace“ }
  ```
- To reset the logging level back to the original value

  ```
  {  "configuredLevel": null }
  ```

Shut Down End Point

- Endpoint is used to shut down the Spring Boot application
- Need to enable by adding a property in the application.yml
- Done By Making a POST call.
  - http://localhost:7070/actuator/shutdown

```
management:
   endpoint:
    shutdown:
      enabled: true
```