Spring MVC

Trainer: Ly Quy Duong

Introduction

- Your role
- Your background and experience in the subject
- What do you want from this course

Course Objectives

- At the end of the course, you will have acquired sufficient knowledge to:
  - perform objective 1
  - perform objective 2

Agenda

- Section One
- Section Two
- Section Three
- Section Four
- Section Five
- Section Six
- Section Seven

Course Audience and Prerequisite

- The course is for <whom>
- The following are prerequisites to <course>:
  - <knowledge>
  - <experiences>
  - <course>
  - …

Assessment Disciplines

- Class Participation: <%>
- Assignment: <%>
- Final Exam: <%>
- Passing Scores: <%>

Duration and Course Timetable

- Course Duration: <hrs>
- Course Timetable:
  - From <time> to <time>
  - Break <x> minutes from <time> to <time>

Further References

- <Source 1>
- <Source 2>
- …

Set Up Environment

- To complete the course, your PC must install:
  - Software 1
  - Software 2
  - …

Course Administration

- In order to complete the course you must:
  - Sign in the Class Attendance List
  - Participate in the course
  - Provide your feedback in the End of Course Evaluation

August 16, 2017

Spring MVC

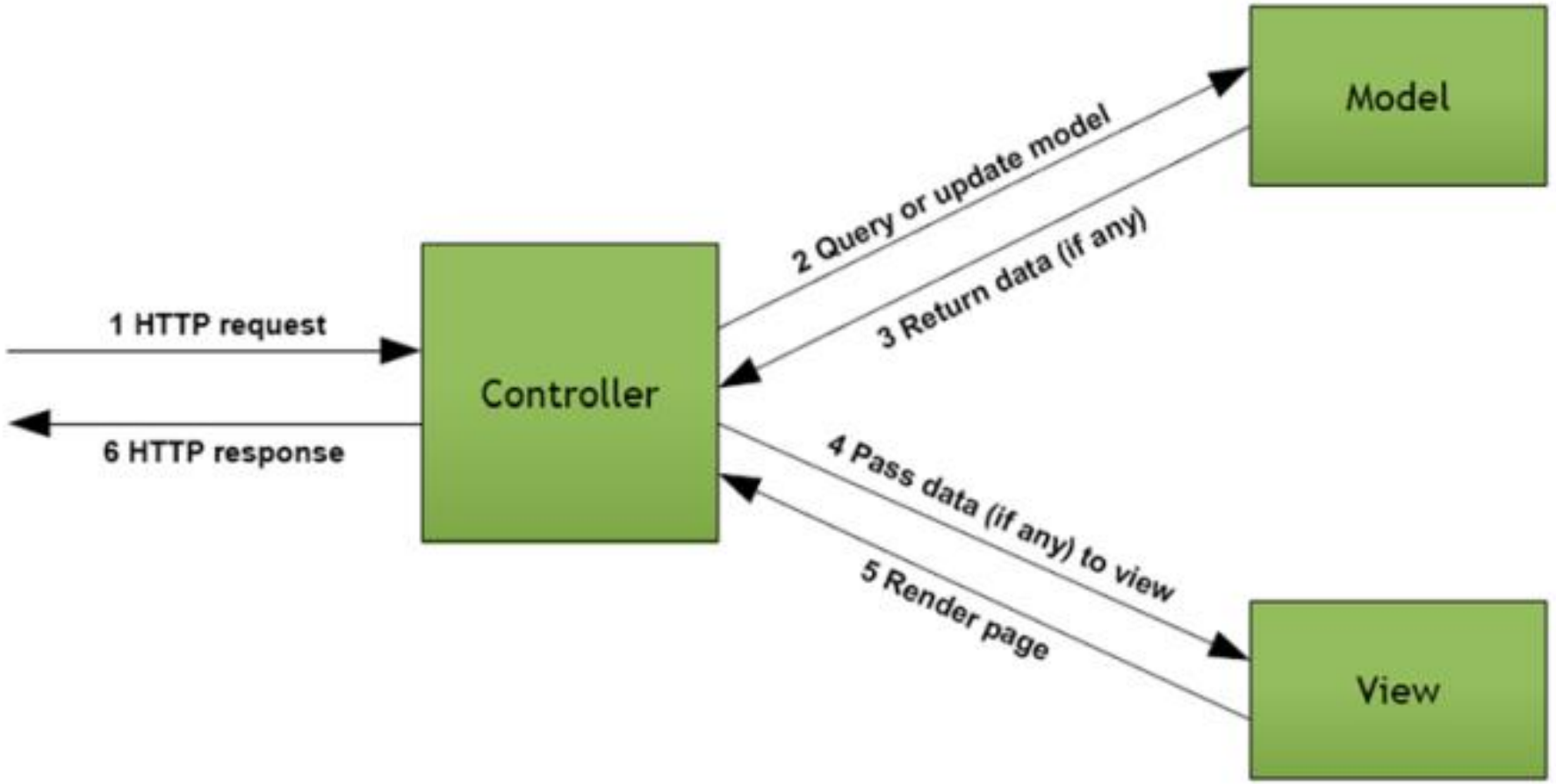
DXC Proprietary and Confidential

DXC.technology

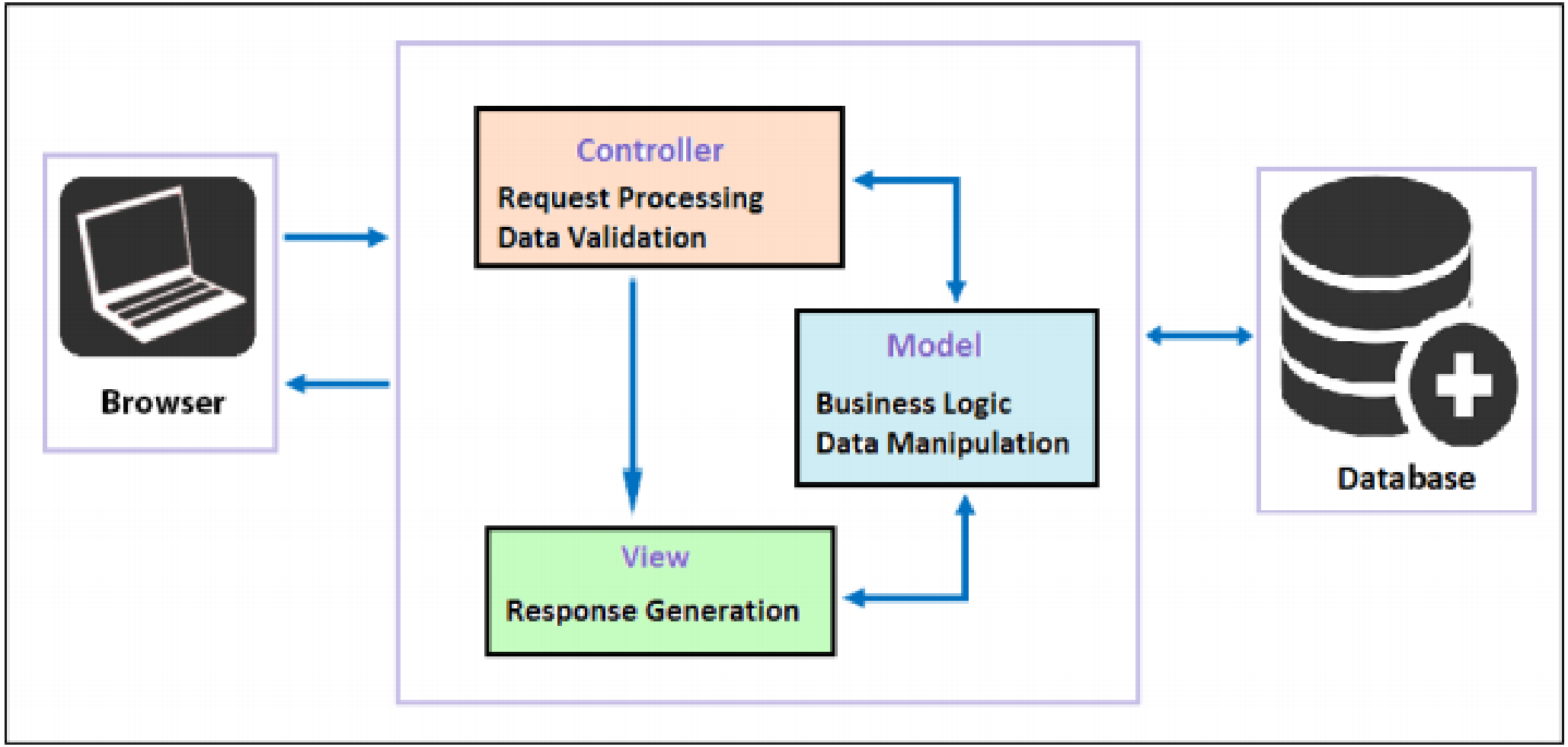
Model-View-Controller (MVC) Design Pattern

- MVC is an architectural pattern used in the development of web applications
- Separate your business services and domain objects (the model) from the UI (the view) and mediate their interaction through one or more controllers.
- To be able to modify your UI without having to change your business logic and domain objects

Model-View-Controller (MVC) Design Pattern



Model-View-Controller (MVC) Design Pattern



Model

- The Model represents the business entity on which the application's data is stored
- It is the conceptualization of the objects that the user works with and the mapping of those concepts into data structures: the user model and data model.

View

- The View is responsible for preparing the presentation for the client based on the outcome of the request processing, without including any business logic
- It renders the model data into the client's user interface type

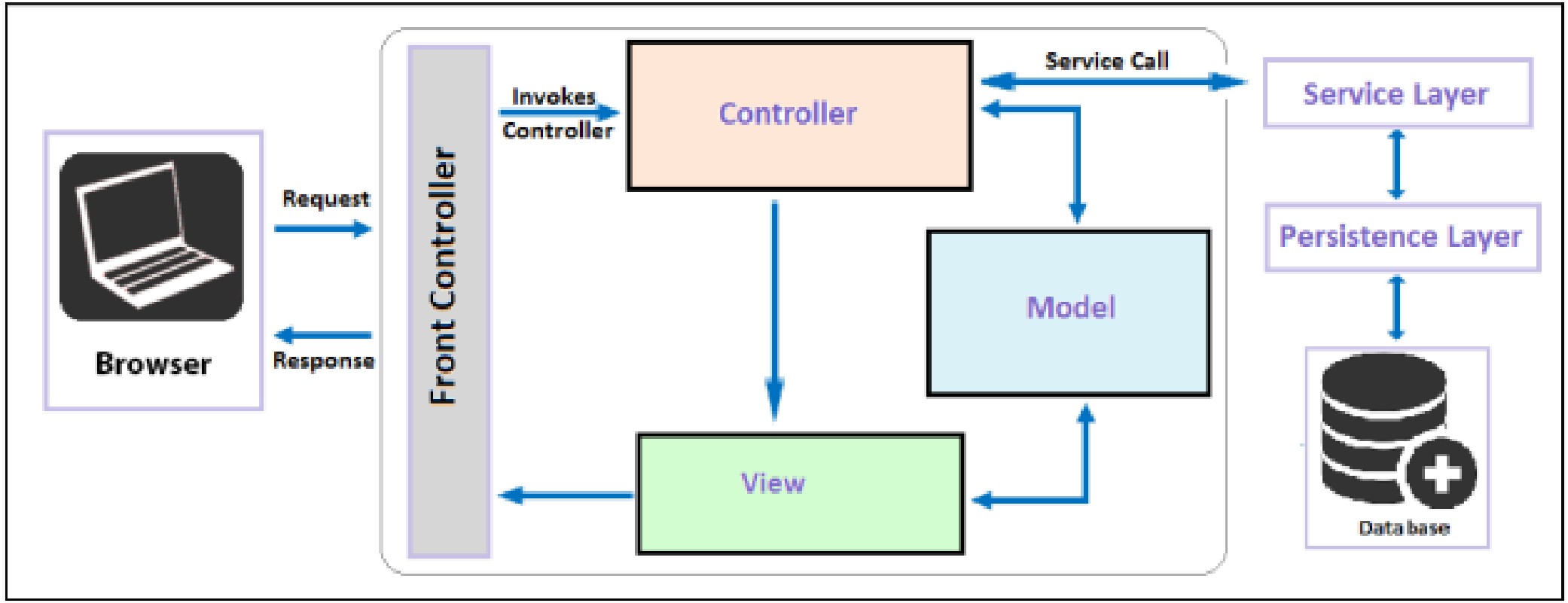
Controller

- The Controller is responsible for controlling the flow request to response flow in the middleware
- It invokes backend services for businesses after receiving a request from the user, and updates the model
- It prepares models for the View to present.
- It is also responsible for determining which view should be rendered.

Front Controller Design Pattern

- The Front Controller is used at the initial point of contact to handle all Hyper Text Transfer Protocol (HTTP) requests
- It enables us to centralize logic to avoid duplicate code, and manages the key HTTP request-handling activities
- The Front Controller design pattern enables centralizing the handling of all HTTP requests without limiting the number of handlers in the system.

Front Controller Design Pattern



Spring MVC

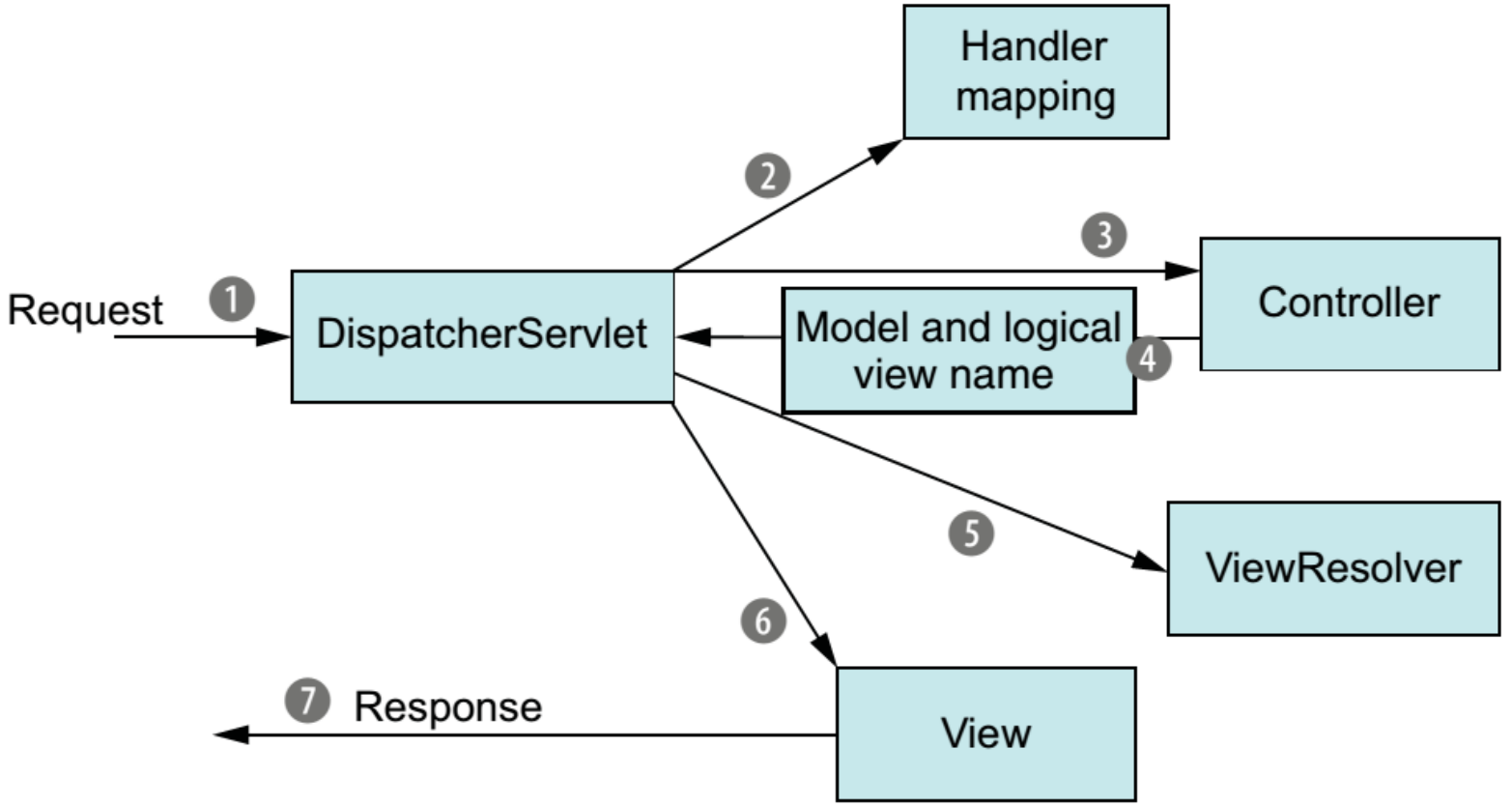
- A web framework built on the principles of the Spring Framework
- Spring’s web framework is designed to address these concerns (state management, workflow, and validation)
- The Spring MVC framework is implemented using standard Java technologies such as Java, Servlet, and JSP

Features of the Spring MVC framework

- Powerful configuration of framework and application classes
- It allows easier testing
- It allows separation of roles. Each component of a Spring MVC framework performs a different role during request handling (Controller, Validator, Model Object, View Resolver, and HandlerMapping interfaces)
- No need for the duplication of code
- It allows specific validation and binding

Flow of request handling in Spring MVC

- Spring moves requests between a dispatcher servlet, handler mappings, controllers, and view resolvers.



DispatcherServlet

- A single front controller servlet.
- The Servlet intercepts and analyzes the incoming HTTP request and dispatches them to the appropriate controller to be processed.
- It is configured in the web.xml file of any web application

Handler mapping

- This maps the HTTP request to the handler, that is, a method within a Spring MVC controller class, based on the HTTP paths expressed through the @RequestMapping annotation at the method or type level within the controller class

Controller

- A **controller** is a Spring component that processes the request
- The Controller in Spring MVC receives requests from the DispatcherServlet class and performs some business logic in accordance with the client.
- Package up the model data and identify the name of a view that should render the output

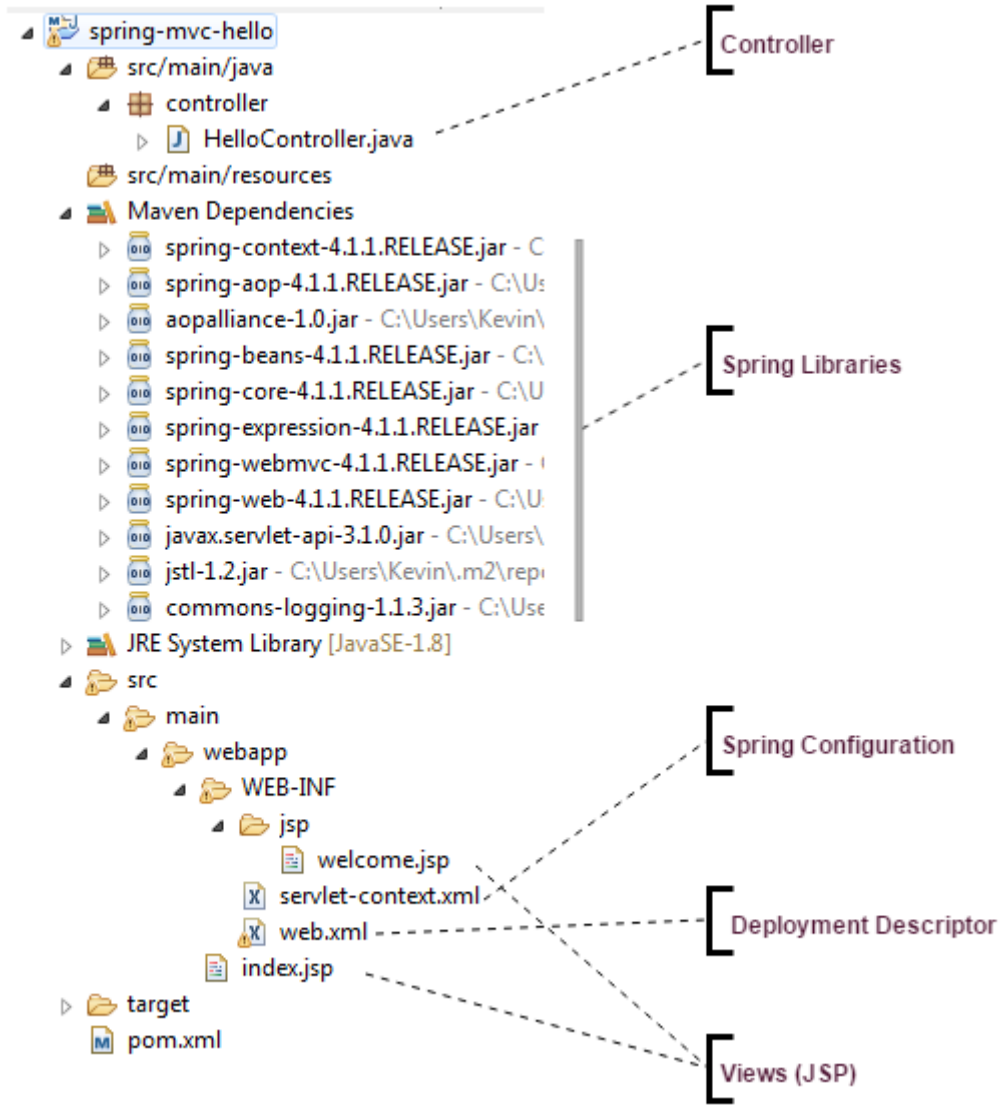
ViewResolver

- The ViewResolver interface of Spring MVC supports view resolution based on the view name returned by controller
  - The URLBasedViewResolver class supports the direct resolution of view name to URLs.
  - The ContentNegotiatingViewResolver class supports the dynamic resolution of views based on the media type supported by the client, such as PDF, XML, JSON , and so on

View

- The View components are user-interface elements which are responsible for displaying the output of a Spring MVC application.

Spring MVC Sample



Controller

```java
@Controller
public class HelloController {

    @RequestMapping("/welcome")
    public String sayHello(ModelMap model){
        String message = "Welcome to Spring MVC.!";

        model.addAttribute("message", message);

        return "welcome";
    }
}
```

Spring Configuration

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">

    <context:component-scan base-package="controller" />

    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix">
            <value>/WEB-INF/jsp/</value>
        </property>
        <property name="suffix">
            <value>.jsp</value>
        </property>
    </bean>

</beans>
```

Deployment Descriptor

```
<servlet>
    <servlet-name>appServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/servlet-context.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>appServlet</servlet-name>
    <url-pattern>/welcome.jsp</url-pattern>
    <url-pattern>/welcome.html</url-pattern>
    <url-pattern>*.html</url-pattern>
</servlet-mapping>
```

@Controller annotation

- The @Controller annotation is used to define a class as a controller class without inheriting any interface or class

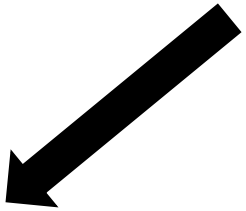
```
@Controller
public class HelloController {

    //....
```

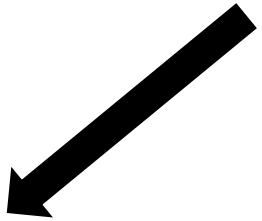
```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">

    <context:component-scan base-package="controller" />
```

@RequestMapping annotation

- The web request in Spring MVC is mapped to handlers by one or more @RequestMapping annotations declared in the controller class

http://localhost:8080/spring-mvc-hello/welcome.html





/spring-mvc-hello

/welcome.html

```
@RequestMapping("/welcome")
public String sayHello(ModelMap model){
```

@RequestMapping - Mapping requests at the class level

```
@Controller
@RequestMapping(value = "/employee")
public class EmployeeController {

    @RequestMapping("/add")
    public String addEmployee (Model model) {
       model.addAttribute("employee", new Employee());
       model.addAttribute("empList", employeeService.list());

       return "employeeList";
    }
...
}
```

http://localhost:8080/spring-mvc/employee/add

@RequestMapping - Mapping requests at the class level

```
@Controller
@RequestMapping(value = "/employee")
public class EmployeeController {

    @RequestMapping(value = {"/remove", "/delete"}",
                            method = RequestMethod.GET)
     public String removeEmployee (
                (@RequestParam("employeeId") int employeeId){

        employeeService.removeEmployee(employeeId);
        return "redirect:";
     }
...
}
```

http://localhost:8080/spring-mvc/employee/remove

http://localhost:8080/spring-mvc/employee/delete

@RequestMapping - Mapping requests at the class level

```
@Controller
@RequestMapping(value = "/employee")
public class EmployeeController {

    @RequestMapping(value = "/{employeeId}",
                                    method = RequestMethod.GET)
    public String getEmployee (
                @PathVariable("employeeId")Integer employeeId,
                Model model){
        //...
        return "employeeList";
    }

...
}
```

http://localhost:8080/spring-mvc/employee/10121

@RequestParam

- It can be used to bind the HTTP request parameter to the argument of the controller method
- Its functionality is similar to ServletRequest.getParameter(java.lang.String)

```
@RequestMapping(value = {"/remove", "/delete"}",
                                method = RequestMethod.GET)
 public String removeEmployee (
                (@RequestParam("employeeId") int employeeId){

    employeeService.removeEmployee(employeeId);
    return "redirect:";
 }
```

Return values in @RequestMapping annotated methods

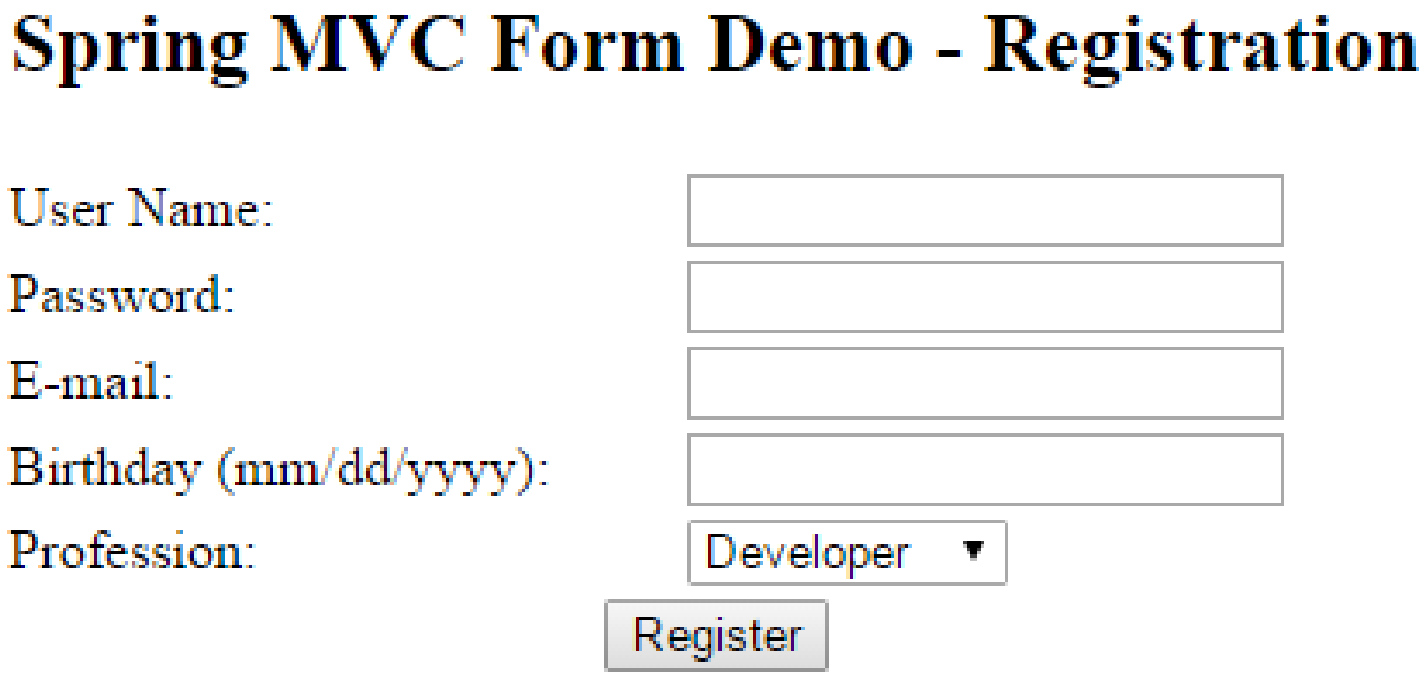
Return type	Description
`ModelAndView`	This holds Model and View information
`String`	This represents the View name
`View`	This represents the View object
`Model/Map`	This contains data exposed by a view; view is determined implicitly by the `RequestToViewNameTranslator` class
`Void`	This specifies that a view can be handled by the invoked method internally or can be determined implicitly by the `RequestToViewNameTranslator` class

ViewResolver in Spring MVC

- Spring provides a number of ViewResolver classes that are configured in the XML files

ViewResolver	Description
`org.springframework.web.servlet.view.ResourceBundleViewResolver`	This configures view names in property files; the default resource bundle is properties
`org.springframework.web.servlet.view.InternalResourceViewResolver`	This refers to a convenient `ViewResolver` class that uses suffix and prefix properties for the view name and `RequestDispatcher` to transfer the control
`org.springframework.web.servlet.view.Freemarker.FreeMarkerViewResolver`	This maps the view name with the `FreemarkerView` class, which is used for the FreeMarker template engine
`org.springframework.web.servlet.view.velocity.VelocityViewResolver`	This maps the view name with the `VelocityView` class, which is used for the Velocity template engine

Spring MVC Form



Spring MVC Form

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


<form:form action="register" method="post" commandName="userForm">
    <table border="0">
        <tr>..
        <tr>
            <td>User Name:</td>
            <td><form:input path="username" /></td>
        </tr>
        <tr>
            <td>Password:</td>
            <td><form:password path="password" /></td>
        </tr>
        <tr>..
        <tr>
            <td>Birthday (mm/dd/yyyy):</td>
            <td><form:input path="birthDate" /></td>
        </tr>
        <tr>
            <td>Profession:</td>
            <td><form:select path="profession" items="${professionList}" /></td>
        </tr>
        <tr>
            <td colspan="2" align="center"><input type="submit" value="Register" /></td>
        </tr>
    </table>
</form:form>
```

Spring MVC Form

```java
@Controller
@RequestMapping(value = "/register")
public class RegisterController {

    @RequestMapping(method = RequestMethod.GET)
    public String viewRegistration(Map<String, Object> model) {
        User userForm = new User();
        model.put("userForm", userForm);

        List<String> professionList = new ArrayList<>();
        professionList.add("Developer");
        professionList.add("Designer");
        professionList.add("IT Manager");
        model.put("professionList", professionList);

        return "Registration";
    }

    @RequestMapping(method = RequestMethod.POST)
    public String processRegistration(@ModelAttribute("userForm") User user, Map<String, Object> model) {

        System.out.println(user.toString());

        return "RegistrationSuccess";
    }
}
```

@ModelAttribute in the controller class

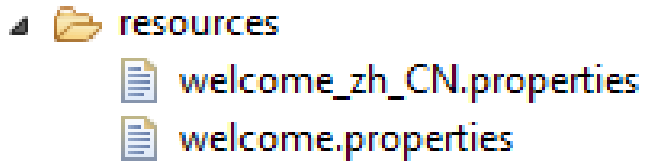
- The org.springframework.web.bind.annotation.ModelAttribute in Spring MVC is used to an annotation for the handler method or method arguments in the controller class
- The @ModelAttribute annotation binds a named model attribute to any arguments in a method or to the method itself

Spring MVC internationalization (i18n)

- Spring Configuration

```
<bean id="messageSource"
    class="org.springframework.context.support.ResourceBundleMessageSource">
    <property name="basename" value="welcome" />
</bean>
```

- Resource Bundle Files



- Using on JSP

```
<spring:message code="welcome.springmvc" text="default text" />
```

Spring MVC internationalization (i18n)

- ReloadableResourceBundleMessageSource
  - Reloading properties file without restarting the JVM

```
<bean id="messageSource"
    class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
    <property name="basename" value="welcome" />
    <property name="defaultEncoding" value="UTF-8" />
</bean>
```

Spring MVC internationalization (i18n)

- SessionLocaleResolver
  - SessionLocaleResolver resolves locales by inspecting a predefined attribute in a user's session.
  - If the session attribute doesn't exist, this locale resolver determines the default locale from the accept-language HTTP header

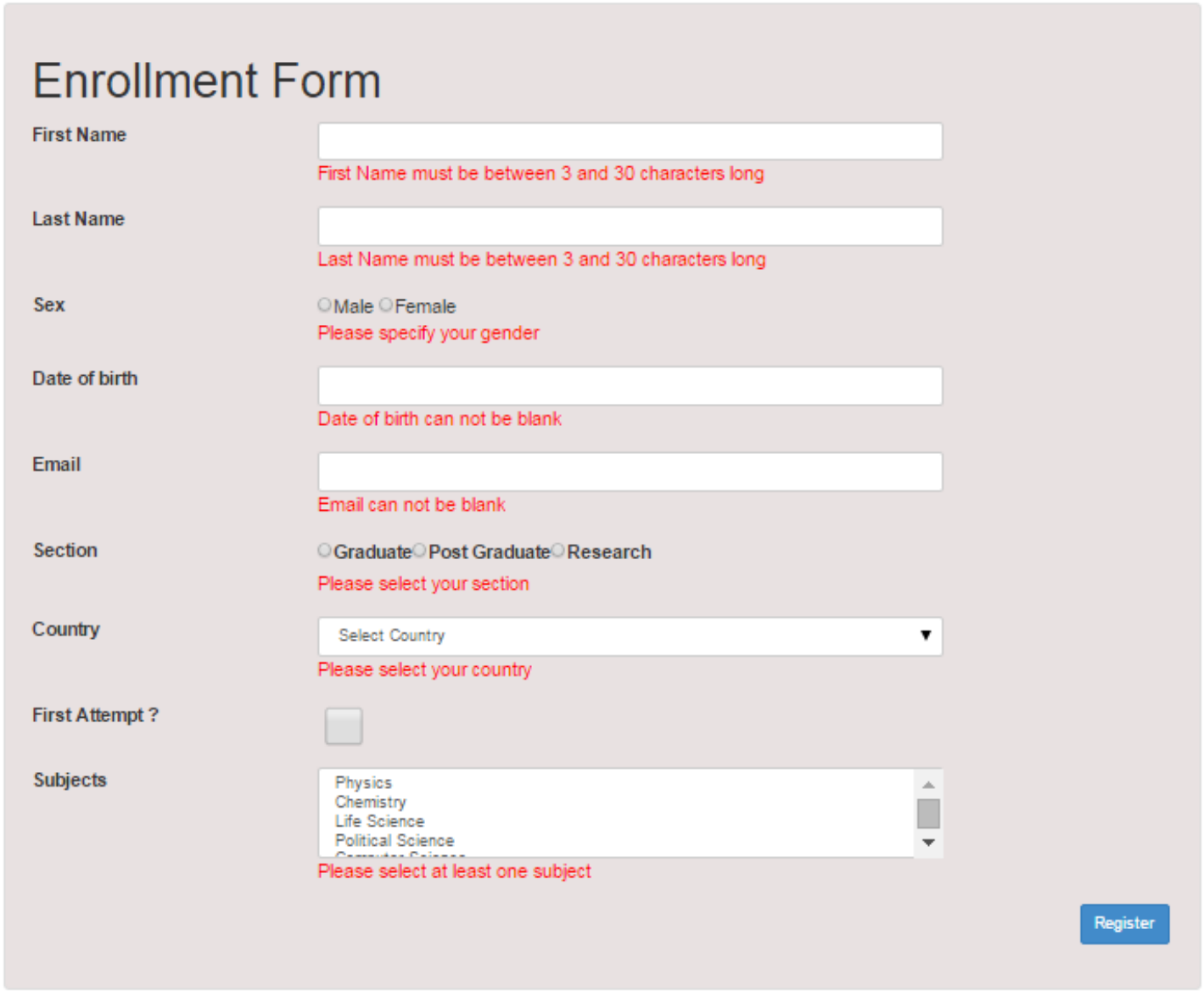
```
<bean id="localeResolver" class="org.springframework.web.servlet.i18n.SessionLocaleResolver">
    <property name="defaultLocale" value="en" />
</bean>
```

Spring MVC internationalization (i18n)

- LocaleChangeInterceptor
  - LocaleChangeInterceptor interceptor detects if a special parameter is present in the current HTTP request.
  - The parameter name can be customized with the paramName property of this interceptor.
  - If such a parameter is present in the current request, this interceptor changes the user's locale according to the parameter value.

```
<bean id="localeChangeInterceptor" class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
    <property name="paramName" value="lang" />
</bean>

<bean class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping">
    <property name="interceptors">
        <list>
            <ref bean="localeChangeInterceptor" />
        </list>
    </property>
</bean>
```

Spring MVC Form Validation Using Hibernate Validators



Spring MVC Form Validation Using Hibernate Validators

```
public class Student implements Serializable {

    @Size(min=3, max=30)
    private String firstName;

    @Size(min=3, max=30)
    private String lastName;

    @NotEmpty
    private String sex;

    @DateTimeFormat(pattern="dd/MM/yyyy")
    @Past @NotNull
    private Date dob;

    @Email @NotEmpty
    private String email;

    @NotEmpty
    private String section;

    @NotEmpty
    private String country;

    private boolean firstAttempt;

    @NotEmpty
    private List<String> subjects = new ArrayList<String>();
```

Spring MVC Form Validation Using Hibernate Validators

- Controller

```
/*
 * This method will be called on form submission, handling POST request It
 * also validates the user input
 */
@RequestMapping(method = RequestMethod.POST)
public String saveRegistration(@Valid Student student, BindingResult result, ModelMap model) {

    if (result.hasErrors()) {
        return "enroll";
    }
```

Spring MVC Form Validation Using Hibernate Validators

```
Size.student.firstName=First Name must be between {2} and {1} characters long
Size.student.lastName=Last Name must be between {2} and {1} characters long
NotEmpty.student.sex=Please specify your gender
NotNull.student.dob=Date of birth can not be blank
Past.student.dob=Date of birth must be in the past
Email.student.email=Please provide a valid Email address
NotEmpty.student.email=Email can not be blank
NotEmpty.student.country=Please select your country
NotEmpty.student.section=Please select your section
NotEmpty.student.subjects=Please select at least one subject
```

@SessionAttributes

- It’s a way to add objects to Session
- @SessionAttributes is used in conjunction with @ModelAttribute

```
@Controller
@RequestMapping("/")
@SessionAttributes({"cart"})
public class ProductController {

    @RequestMapping(method = RequestMethod.GET)
    public String get(Model model) {
        if (!model.containsAttribute("cart")) {
            model.addAttribute("cart", new ArrayList<Product>());
        }
```

August 16, 2017

RESTful Web Service - Spring

DXC Proprietary and Confidential

DXC.technology

RESTful Web Service - Spring

- What is REST?
- REST Concepts
- RESTful Architecture Design
- Spring @MVC
- Implement REST api using Spring @MVC

What is REST?

- Representational State Transfer
- Term coined up by Roy Fielding
  - Author of HTTP spec
- Architectural style
- Architectural basis for HTTP
  - Defined a posteriori

Core REST Concepts

- Identifiable Resources
- Uniform interface
- Stateless conversation
- Resource representations
- Hypermedia

Identifiable Resources

- Everything is a resource
  - Customer
  - Order
  - Catalog Item
- Resources are accessed by URIs

Uniform interface

- Interact with Resource using single, fixed
- interface
- **GET /orders** - fetch list of orders
- **GET /orders/1** - fetch order with id 1
- **POST /orders** – create new order
- **PUT /orders/1** – update order with id 1
- **DELETE /orders/1** – delete order with id 1
- **GET /customers/1/order** – all orders for customer with id 1

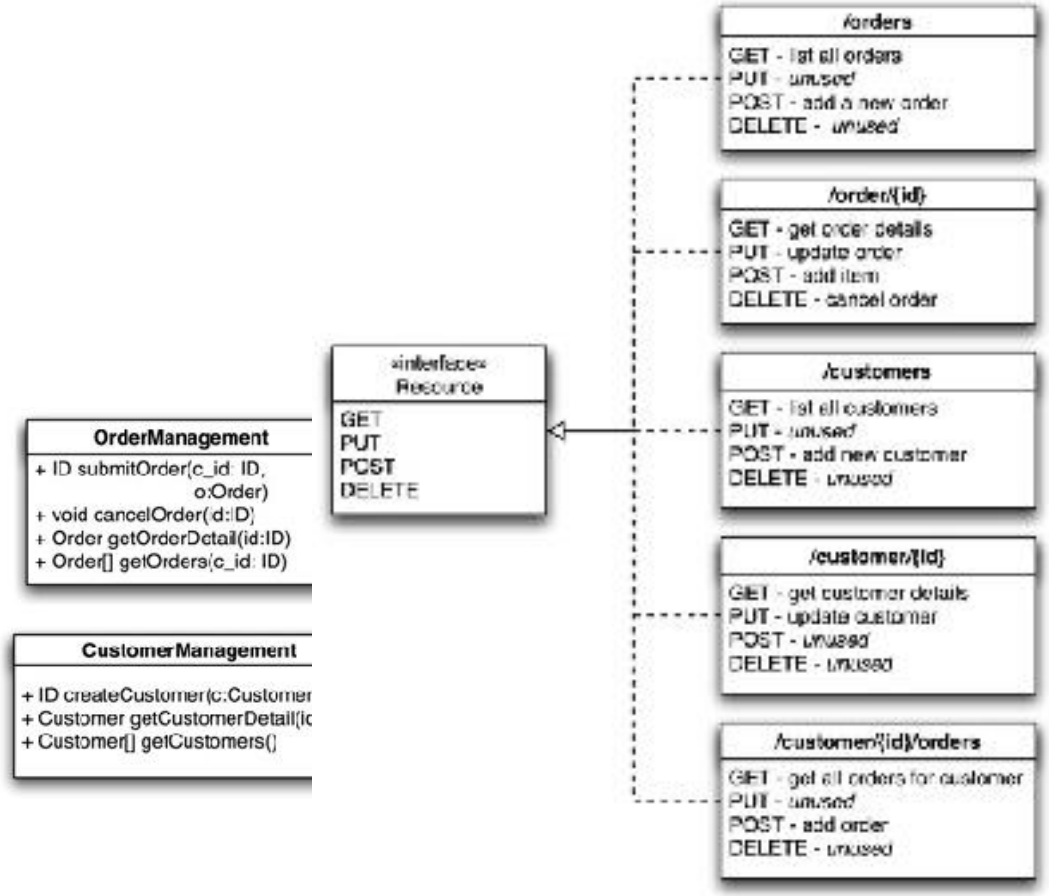
Resource representations

- More that one representation possible
  - text/html, image/gif, application/pdf
- Desired representation set in Accept HTTP header
  - Or file extension
- Delivered representation show in Content-Type
- Access resources through representation
- Prefer well-known media types

Stateless conversation

- Server does not maintain state
  - Don’t use the Session!
- Client maintains state through links
- Very scalable
- Enforces loose coupling (no shared session knowledge)

RESTful Architecture



RESTful application design

- Identify resources
  - Design URIs
- Select representations
  - Or create new ones
- Identify method semantics
- Select response codes

Spring @MVC

```
@Controller
public class AccountController {
        @Autowired AccountService accountService;

        @RequestMapping("/details")
        public String details(@RequestParam("id")Long id,
                              Model model) {
                Account account = accountService.findAccount(id);
                model.addAttribute(account);
                return "account/details";
        }
}

account/details.jsp
        <h1>Account Details</h1>
        <p>Name :  ${account.name}</p>
        <p>Balance :  ${account.balance}</p>

../details?id=1
```

RESTful support in Spring 3.0

- Extends Spring @MVC to handle URL templates - @PathVariable
- Leverages Spring OXM module to provide marshalling / unmarshalling (castor, xstream, jaxb etc)
- @RequestBody – extract request body to method parameter
- @ResponseBody – render return value to response body using converter – no views required

REST controller

```
@Controller public class AccountController {

        @RequestMapping(value="/accounts",method=RequestMethod.GET)
        @ResponseBody  public AccountList getAllAcccount() {
        }

        @RequestMapping(value="/accounts/${id}",method=RequestMethod.GET)
        @ResponseBody public Account getAcccount(@PathVariable("id")Long id) {
        }

        @RequestMapping(value="/accounts/",method=RequestMethod.POST)
        @ResponseBody public Long createAcccount(@RequestBody Account account) {
        }

        @RequestMapping(value="/accounts/${id}",method=RequestMethod.PUT)
        @ResponseBody public Account updateAcccount(@PathVariable("id")Long id, @RequestBody Account account) {
        }

        @RequestMapping(value="/accounts/${id}",method=RequestMethod.DELETE)
        @ResponseBody public void deleteAcccount(@PathVariable("id")Long id) {
        }
}
```

Invoke REST services

- The new RestTemplate class provides clientside invocation of a RESTful web-service

HTTP Method	RestTemplate Method
DELETE	delete(String url, String... urlVariables)
GET	getForObject(String url, Class<t> responseType, String … urlVariables)
HEAD	headForHeaders(String url, String… urlVariables)
OPTIONS	optionsForAllow(String url, String… urlVariables)
POST	postForLocation(String url, Object request, String… urlVariables)
PUT	put(String url, Object request, String…urlVariables)

Points to Remember

DXC Proprietary and Confidential

Questions & Answer

Thank You !

Revision History

Date	Version	Description	Updated by	Reviewed and Approved By
12/13/2015	1.0	Initial Document • Front Controller Design Pattern • Spring MVC • Spring MVC Form	Kien Tran	
Dec 2016	1.1	Add • RESTful Web Service - Spring	Duong Ly	
Aug 2017	2.0	Retheme with DXC template		Quang Tran