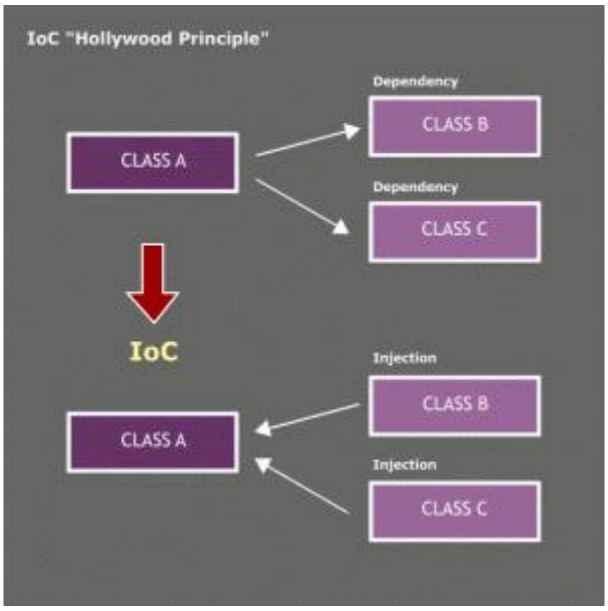
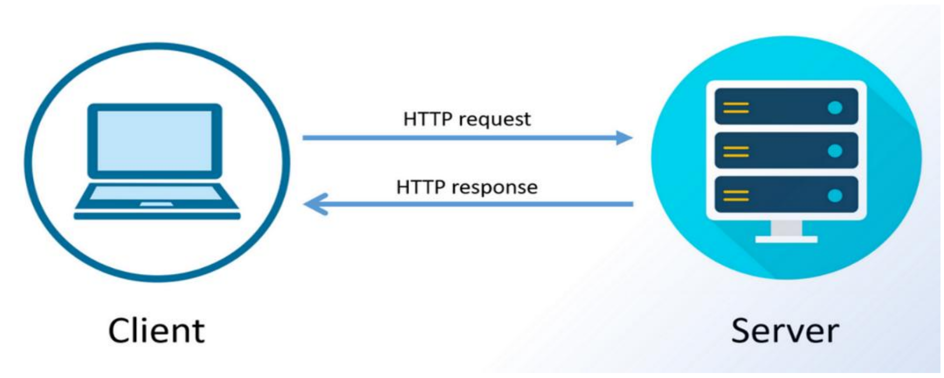
Spring Intro and IOC



Request-Response Programming Model



Introduction to Java Servlets

Servlets are the Java programs that run on the Java-enabled web server or application server. They are used to handle the request obtained from the webserver, process the request, produce the response, then send a response back to the webserver.

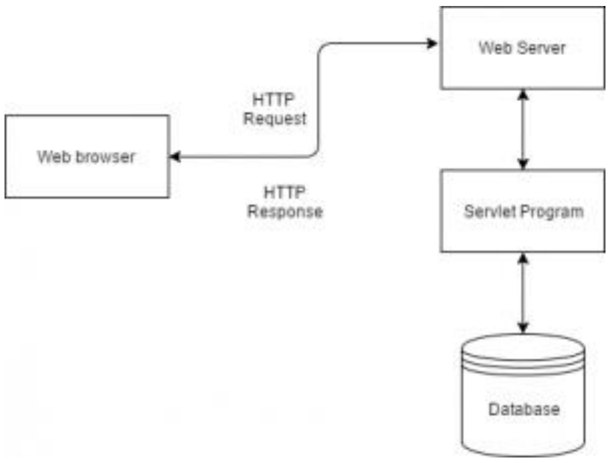
History of Web application, initial days we had to install – client program application to connect to server. With browser-based technology browser act as client which connect to server – sends request and processes response for client.

JSP – provide way to embed java code in html.

Client Side vs Server Side Rendering for Web Apps.

A servlet is a Java programming language class that is used to extend the capabilities of servers that host applications accessed by means of a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers.

Servlet Architecture:



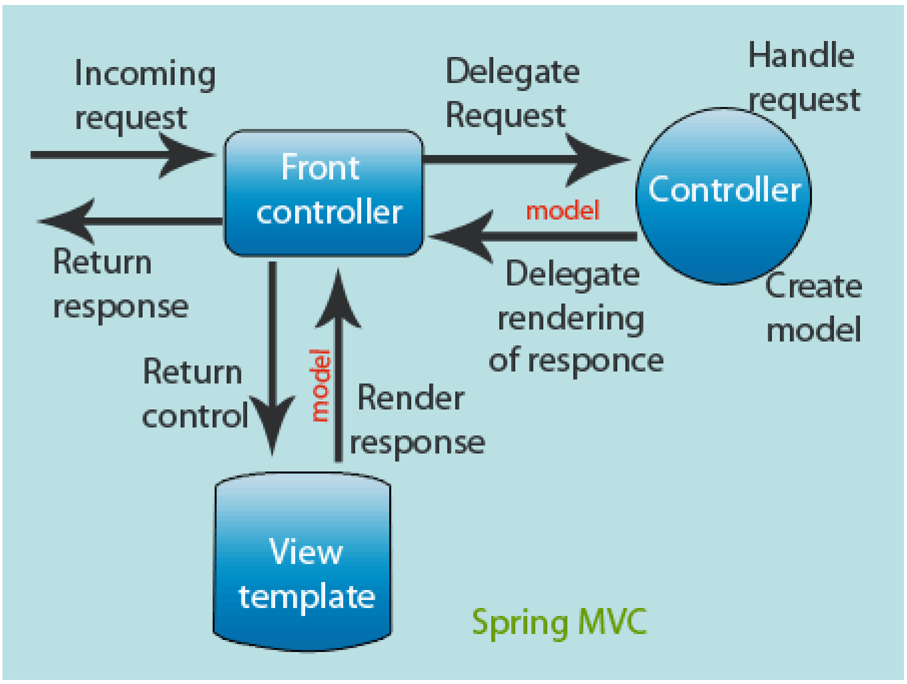
Spring MVC framework Introduction

Spring MVC framework and it's lifecycle.

Spring MVC is model-view-controller(MVC) based web framework. **Spring MVC is the web component of the Spring framework**. Spring MVC framework is highly configurable in terms of functionality and logic. Spring MVC framework integration with other framework like Struts, JSF, Tapestry etc is very easy. For rendering view JSP and Servlet is not the only option in Spring MVC framework, you can use other view technologies like Freemarker, Velocity, Excel or Pdf for providing view to the clients.

Spring MVC framework is request driven and designed around a central Servlet. This central Servlet sends requests, for processing, to suitable controller and it also render the render the view to the client with the help of *View Resolver* object. The central Servlet in Spring MVC is *DispatcherServlet* which is fully integrated with Loc container, which give us freedom to use other features of Spring.

Given below the complete life cycle of Spring MVC (step by step):



- First client made a request in form of http request.
- This request is caught by the Front controller (i.e. *DispatcherServlet*).
- *DispatcherServlet* consults with suitable Handler Mapping to dispatch the request to suitable Controller.

- Controller process the request, and returns model and view to the Front controller (i.e. *DispatcherServlet*).

- The Front controller afterwards resolves the View with the help of View Resolver object and rendered back to the client.

https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html

Life Cycle of Web MVC

1. **Request Handling:**
   - The client sends a request to the web server.
   - The web server forwards the request to the appropriate controller.
2. **Controller Processing:**
   - The controller processes the request, extracting data from the request if needed.
   - It interacts with the model to perform business logic.
   - The controller determines the appropriate view to render.
3. **Model Interaction:**
   - The controller interacts with the model to retrieve or update data.
   - Business logic and data manipulation occur in this phase.
4. **View Resolution:**
   - The controller selects the appropriate view to render based on the request and model data.
   - The view is responsible for presenting the data to the user.
5. **View Rendering:**
   - The selected view is rendered, combining the model data with the view template.
   - The resulting HTML or other content is sent back to the client.
6. **Response Sent:**
   - The server sends the response back to the client, completing the request-response cycle.
7. **User Interaction:**
   - The user interacts with the rendered view, triggering new requests.

DispatcherServlet

DispatcherServlet acts as the **Front Controller** for Spring-based web applications.

What is Front Controller?

Any request is going to come into our website the front controller is going to stand in front and is going to accept all the requests and once the front controller accepts that request then this is the job of the front controller that it will make a decision that who is the right controller to handle that request.

DispatcherServlet handles an incoming HttpRequest, delegates the request, and processes that request according to the configured HandlerAdapter interfaces that have been implemented within the Spring application along with accompanying annotations specifying handlers, controller endpoints, and response objects.

***DispatcherServlet** is example of front controller design pattern.*

Spring MVC framework Features

This section contains the unique features of Spring MVC web framework.

Given below the unique features of the Spring MVC framework :

- **Separate roles** - The Spring MVC separates each role, where the model object, controller, command object, view resolver, DispatcherServlet, validator, etc. can be fulfilled by a specialized object.
- **Light-weight** - It uses light-weight servlet container to develop and deploy your application.
- **Powerful Configuration** - It provides a robust configuration for both framework and application classes that includes easy referencing across contexts, such as from web controllers to business objects and validators.
- **Rapid development** - The Spring MVC facilitates fast and parallel development.
- **Reusable business code** - Instead of creating new objects, it allows us to use the existing business objects.
- **Easy to test** - In Spring, generally we create JavaBeans classes that enable you to inject test data using the setter methods.
- **Flexible Mapping** - It provides the specific annotations that easily redirect the page.

Stereotype Annotations in Spring

Stereotype annotations in the context of Spring Framework carry a specific meaning and are used to categorize and define the roles of classes within an application.

These annotations provide metadata that helps the Spring container understand how to treat and manage these classes. Here's a breakdown of the meaning behind stereotype annotations:

1. **@Component:**
    - **Meaning:** Indicates that a class is a general-purpose Spring component.
    - **Usage:** Used for any plain Java class that needs to be managed by the Spring IoC (Inversion of Control) container. These classes are typically used for various purposes in the application.
2. **@Controller:**
    - **Meaning:** Identifies a class as a controller in a Spring MVC (Model-View-Controller) architecture.
    - **Usage:** Used for classes that handle user requests, interact with the business logic, and return views. Controllers are a key part of the web layer in a Spring MVC application.
3. **@Service:**
    - **Meaning:** Marks a class as a service component in the business/service layer.
    - **Usage:** Used for classes that contain business logic, perform operations, and are part of the service layer. Services often encapsulate the application's core functionality.
4. **@Repository:**
    - **Meaning:** Identifies a class as a data repository, typically for database operations.
    - **Usage:** Used for classes that interact with a database, providing data access logic. Repositories are commonly used in the persistence layer and work with Spring Data for simplified data access.

Spring MVC In Practice:

Deployment Descriptor:

Maven Archtype:

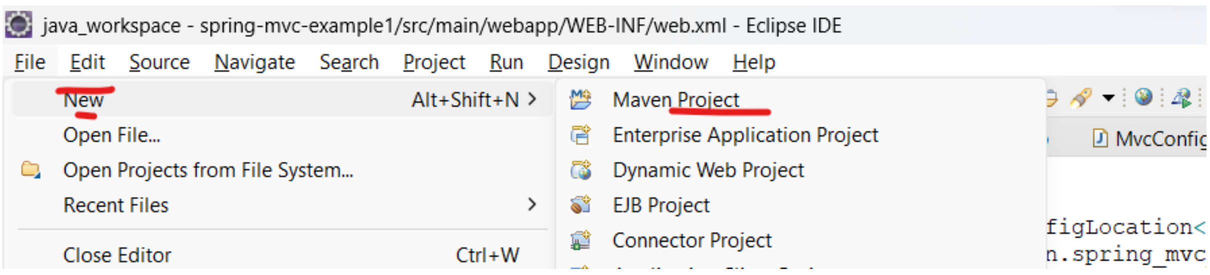
https://maven.apache.org/archetypes/maven-archetype-webapp/

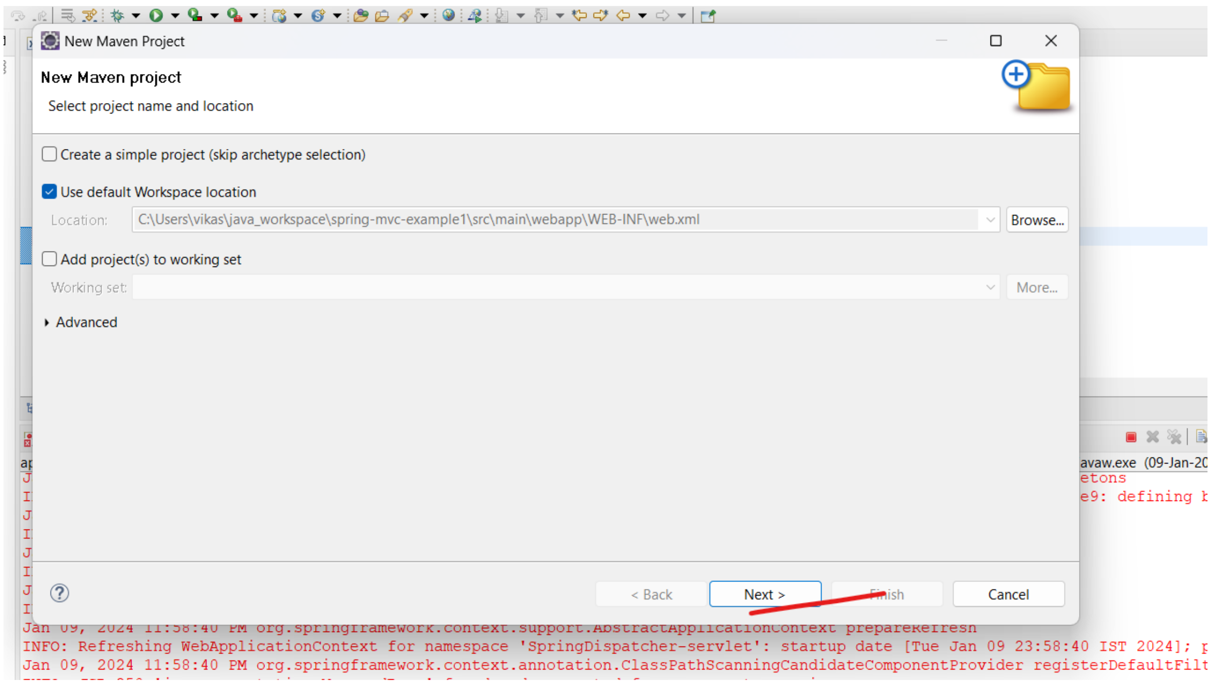
Troubleshooting:

If you get error – The default superclass, "javax.servlet.http.HttpServlet", according to the project's Dynamic Web Module facet version (2.3), was not found on the Java Build Path. Follow steps – in Eclipse -> Project → *Properties* → *Target Runtimes* → *Apache Tomcat*
To change deployment descriptor in eclipse edit file to change specific versions: org.eclipse.wst.common.project.facet.core.xml

If JSP displays ${} then add below dependency to solve problem and add line to jsp page.
 	
`<dependency>`
`    <groupId>javax.servlet</groupId>`
`    <artifactId>jstl</artifactId>`
`    <version>1.2</version>`
`</dependency>`
 	
Add jsp page with below
`<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>`

Create Simple non Spring-MVC App.


java_workspace - spring-mvc-example1/src/main/webapp/WEB-INF/web.xml - Eclipse IDE
File Edit Source Navigate Search Project Run Design Window Help
New Alt+Shift+N
Open File...
Open Projects from File System...
Recent Files
Close Editor Ctrl+W
Maven Project
Enterprise Application Project
Dynamic Web Project
EJB Project
Connector Project
MvcConfig
figLocation<
n.spring_mvc



New Maven Project

New Maven project

Select project name and location

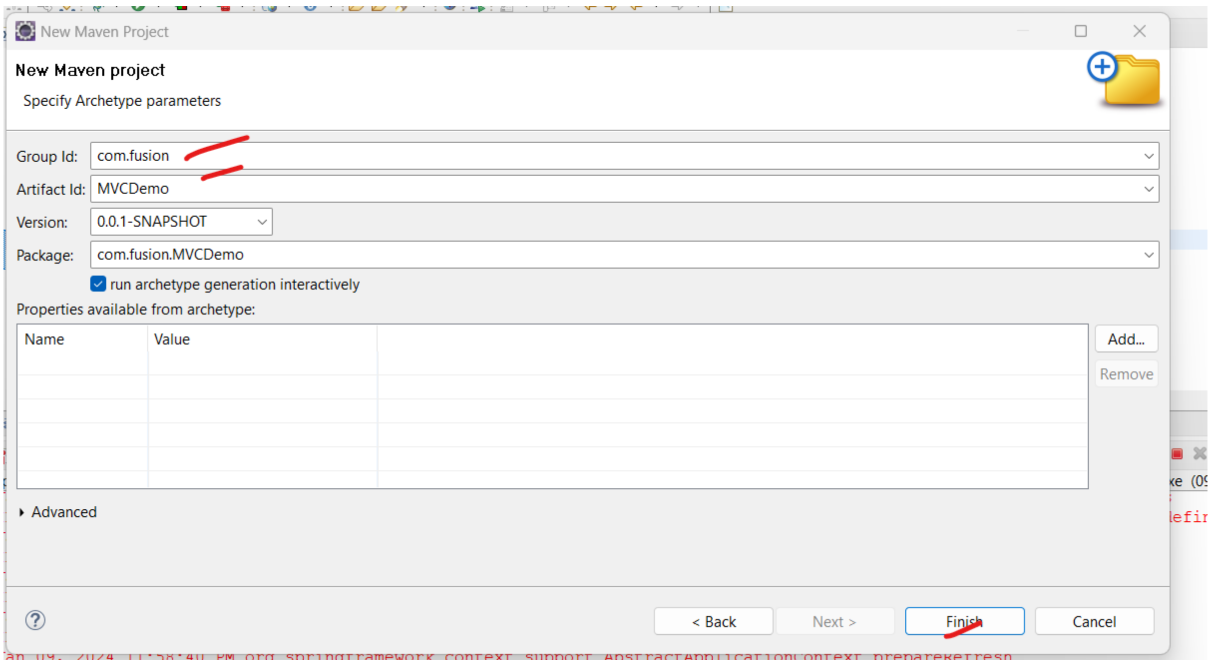
- [ ] Create a simple project (skip archetype selection)
- [x] Use default Workspace location

Location: C:\Users\vikas\java_workspace\spring-mvc-example1\src\main\webapp\WEB-INF\web.xml

- [ ] Add project(s) to working set

Working set:

```
Jan 09, 2024 11:58:40 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh
INFO: Refreshing WebApplicationContext for namespace 'SpringDispatcher-servlet': startup date [Tue Jan 09 23:58:40 IST 2024]; p
Jan 09, 2024 11:58:40 PM org.springframework.context.annotation.ClassPathScanningCandidateComponentProvider registerDefaultFilt
```



New Maven Project

New Maven project

Specify Archetype parameters

Group Id: com.fusion

Artifact Id: MVCDemo

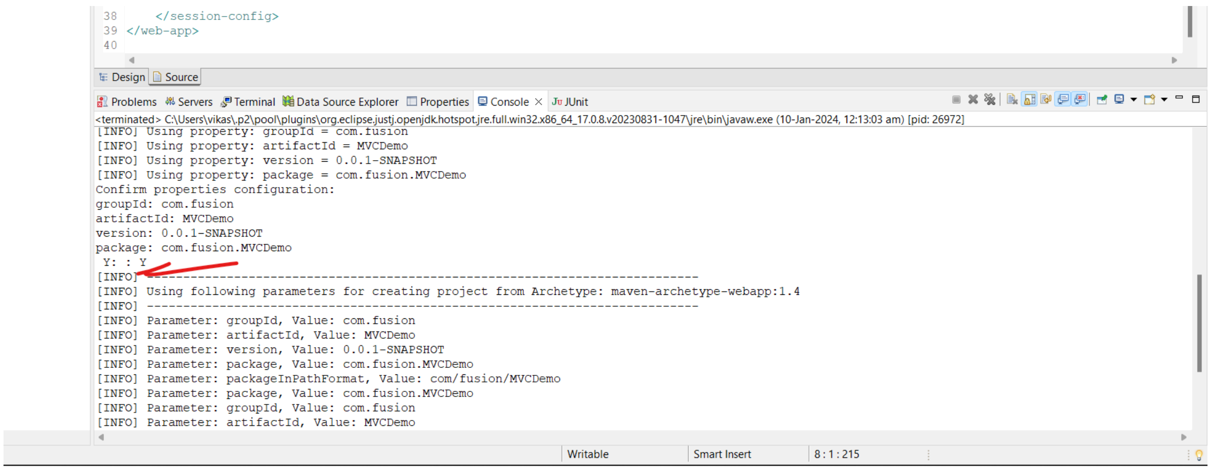
Version: 0.0.1-SNAPSHOT

Package: com.fusion.MVCDemo

- [x] run archetype generation interactively

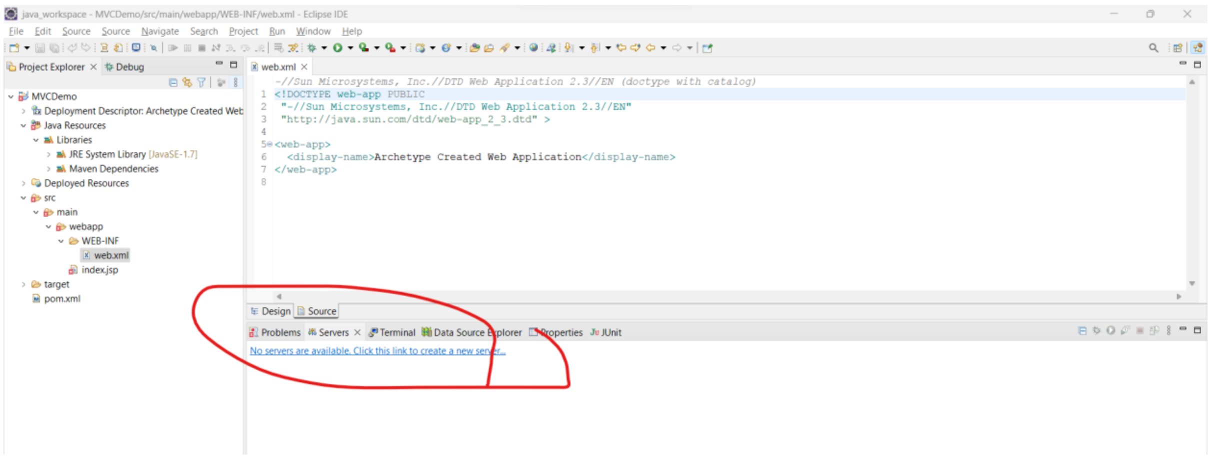
Properties available from archetype:

Name	Value



```
    </session-config>
</web-app>
```

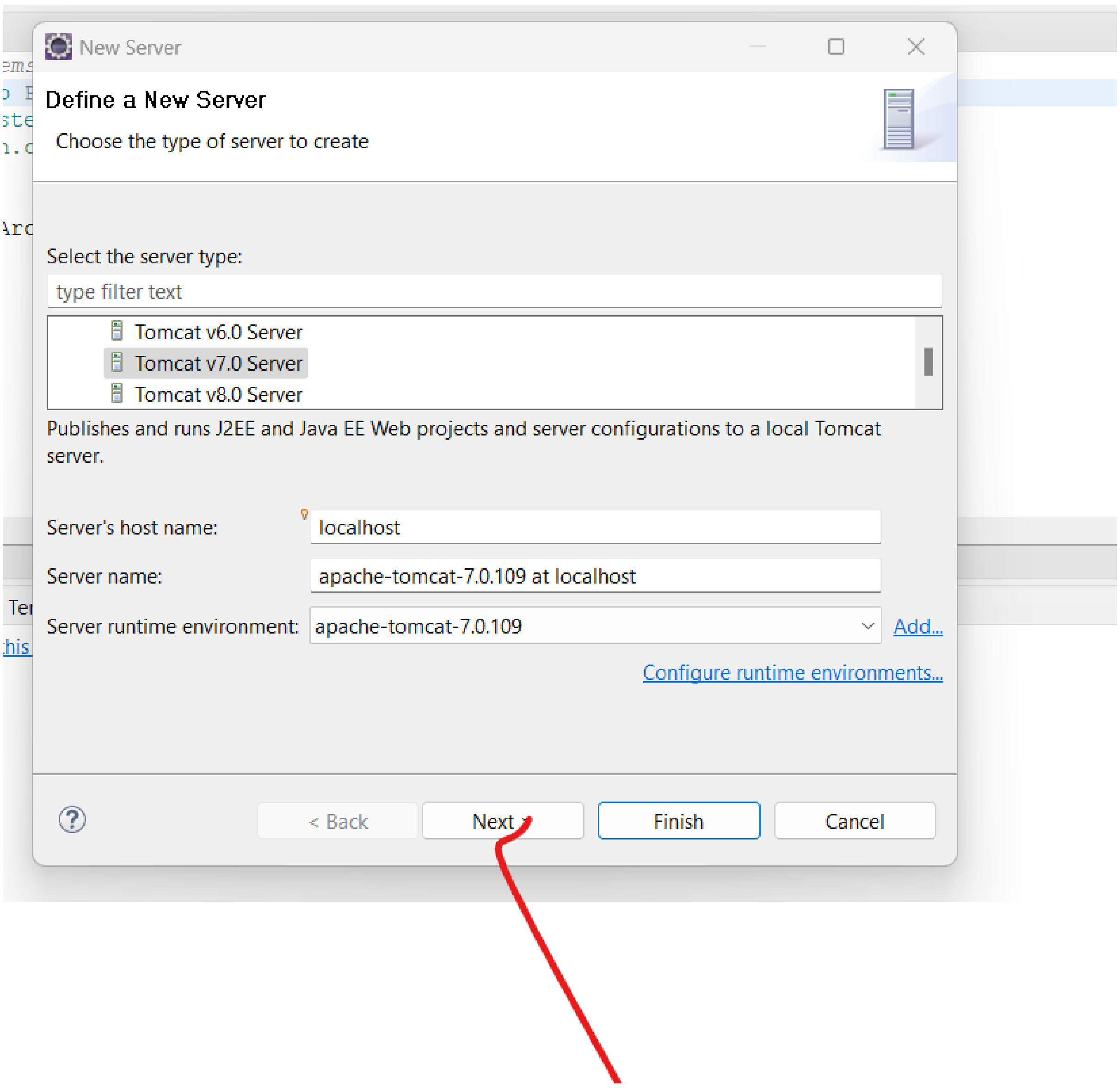
```
<terminated> C:\Users\vikas\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.8.v20230831-1047\jre\bin\javaw.exe (10-Jan-2024, 12:13:03 am) [pid: 26972]
[INFO] Using property: groupId = com.fusion
[INFO] Using property: artifactId = MVCDemo
[INFO] Using property: version = 0.0.1-SNAPSHOT
[INFO] Using property: package = com.fusion.MVCDemo
Confirm properties configuration:
groupId: com.fusion
artifactId: MVCDemo
version: 0.0.1-SNAPSHOT
package: com.fusion.MVCDemo
 Y: : Y
[INFO] ----------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Archetype: maven-archetype-webapp:1.4
[INFO] ----------------------------------------------------------------------------
[INFO] Parameter: groupId, Value: com.fusion
[INFO] Parameter: artifactId, Value: MVCDemo
[INFO] Parameter: version, Value: 0.0.1-SNAPSHOT
[INFO] Parameter: package, Value: com.fusion.MVCDemo
[INFO] Parameter: packageInPathFormat, Value: com/fusion/MVCDemo
[INFO] Parameter: package, Value: com.fusion.MVCDemo
[INFO] Parameter: groupId, Value: com.fusion
[INFO] Parameter: artifactId, Value: MVCDemo
```



```
-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN (doctype with catalog)
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>Archetype Created Web Application</display-name>
</web-app>
```

No servers are available. Click this link to create a new server...



New Server

Define a New Server

Choose the type of server to create

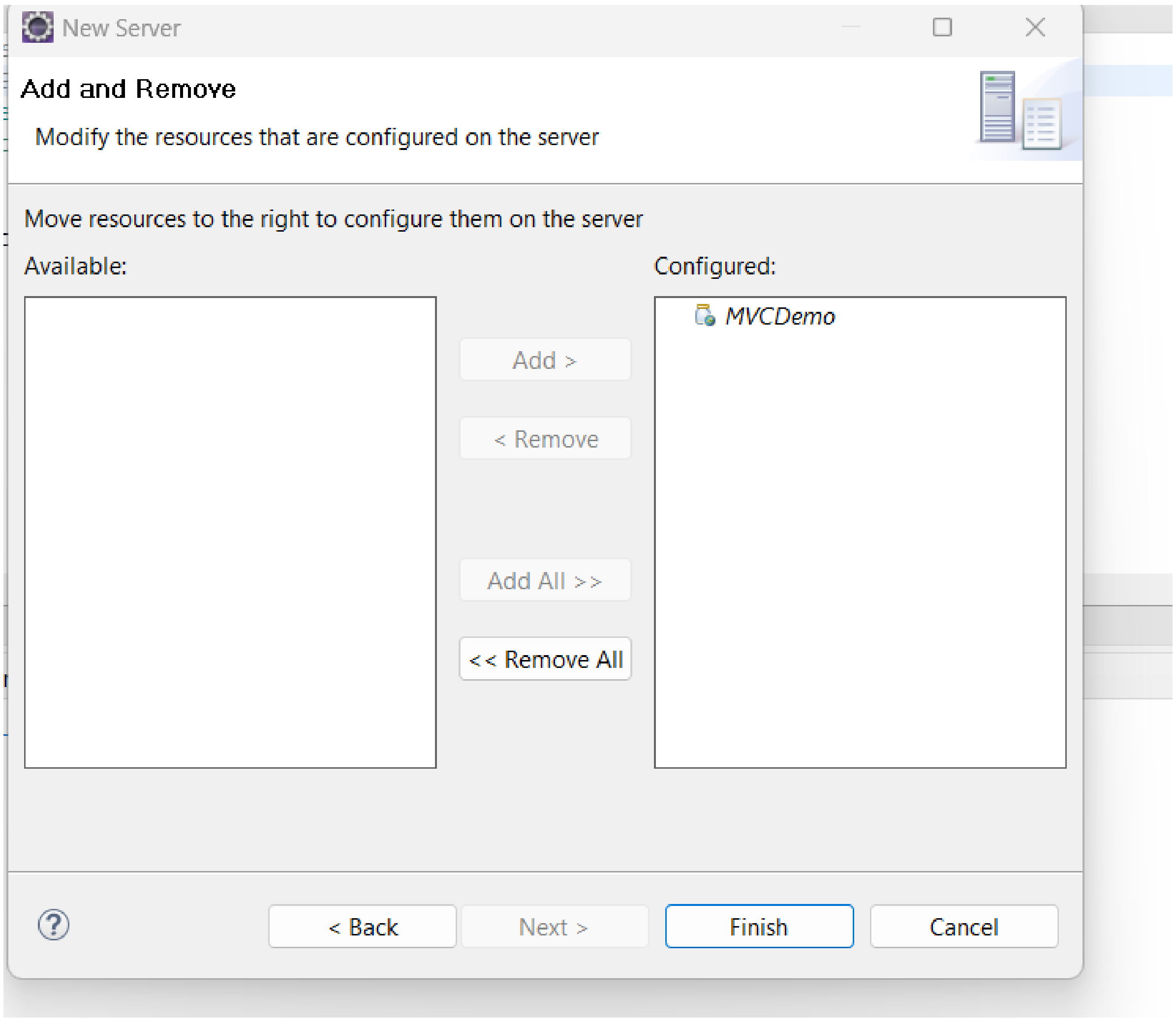
Select the server type:

type filter text

- Tomcat v6.0 Server
- Tomcat v7.0 Server
- Tomcat v8.0 Server

Publishes and runs J2EE and Java EE Web projects and server configurations to a local Tomcat server.

Server's host name:	localhost
Server name:	apache-tomcat-7.0.109 at localhost
Server runtime environment:	apache-tomcat-7.0.109



New Server

Add and Remove

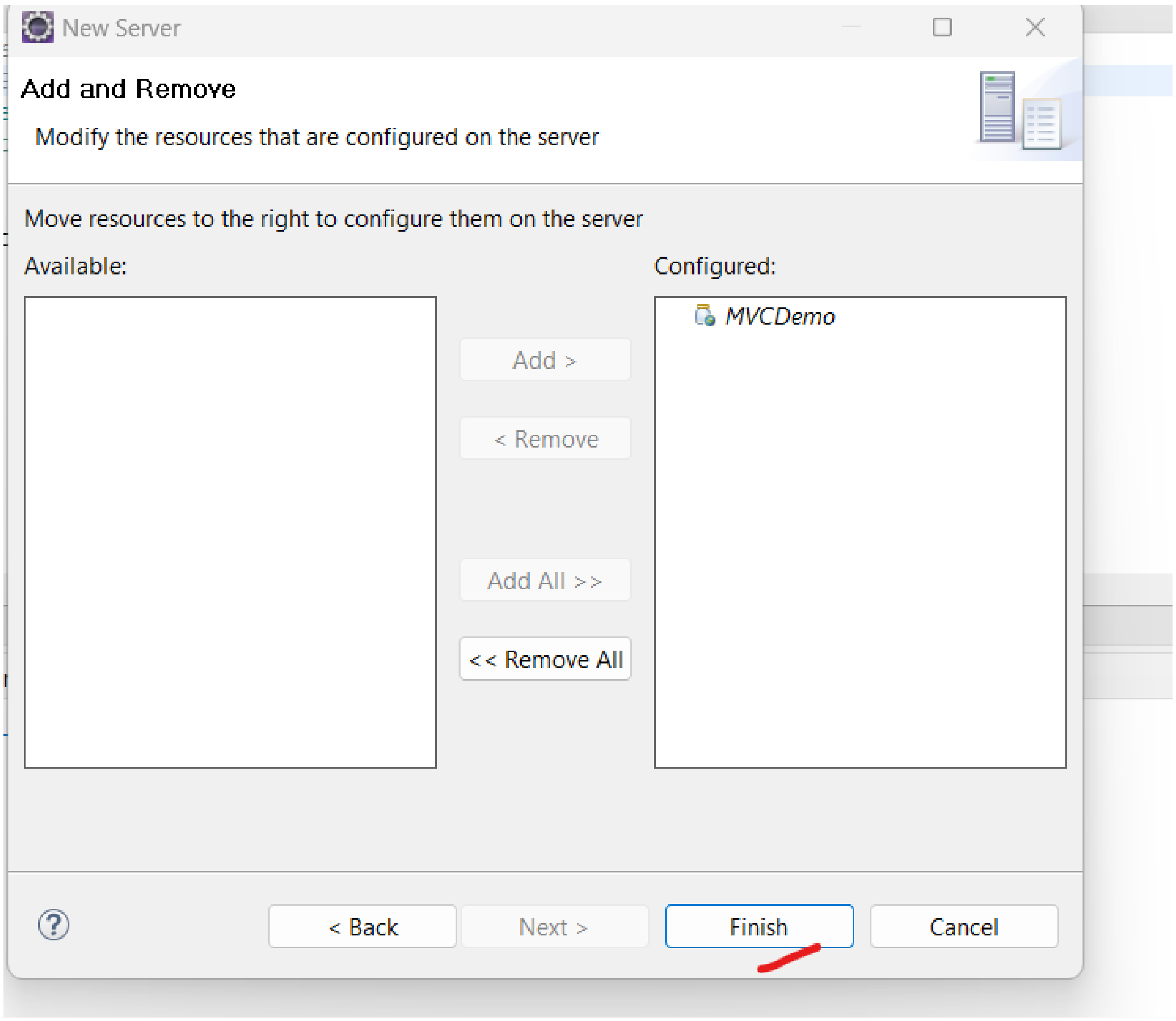
Modify the resources that are configured on the server

Move resources to the right to configure them on the server

Available:

Configured:

MVCDemo


New Server
Add and Remove
Modify the resources that are configured on the server
Move resources to the right to configure them on the server
Available:
Configured:
MVCDemo
Add >
< Remove
Add All >>
<< Remove All
< Back
Next >
Finish
Cancel

Spring MVC

S.No.	SPRING MVC	SPRING BOOT
1.	Spring MVC is a Model View, and Controller based web framework widely used to develop web applications.	Spring Boot is built on top of the conventional spring framework, widely used to develop REST APIs.
2.	If we are using Spring MVC, we need to build the configuration manually.	If we are using Spring Boot, there is no need to build the configuration manually.
3.	In the Spring MVC, a deployment descriptor is required.	In the Spring Boot, there is no need for a deployment descriptor.
4.	Spring MVC specifies each dependency separately.	It wraps the dependencies together in a single unit.

5.	Spring MVC framework consists of four components : Model, View, Controller, and Front Controller.	There are four main layers in Spring Boot: Presentation Layer, Data Access Layer, Service Layer, and Integration Layer.
6.	It takes more time in development.	It reduces development time and increases productivity.
7.	Spring MVC do not provide powerful batch processing.	Powerful batch processing is provided by Spring Boot.
8.	Ready to use feature are provided by it for building web applications.	Default configurations are provided by it for building a Spring powered framework.