🌐Spring Boot Web Application (JSP + MVC + Request Mapping + Request Object)

A complete, beginner-friendly **SPRING BOOT WEB APP** guide explaining:

✅How Spring Boot Web MVC works
✅What DispatcherServlet does
✅How `@Controller`, `@GetMapping`, `@PostMapping` work
✅How to use `HttpServletRequest`
✅How to pass data from Controller → JSP
✅How to configure JSP in Spring Boot (because it's NOT supported by default)
✅How to build a mini working web app (form → controller → JSP)

☑️1. Difference between Spring Boot Web vs Spring Boot REST

Spring Boot has two major ways to create web apps:

Feature	Spring Boot Web (MVC + JSP/HTML)	Spring Boot REST API
Annotation	`@Controller`	`@RestController`
Response Type	JSP/HTML (View)	JSON
Use Case	Websites, Forms	APIs, Mobile apps

☸️For JSP, we MUST use `@Controller`.

🚳2. How DispatcherServlet Works (Very Important)

Spring Boot automatically creates **DispatcherServlet**, which handles *all incoming HTTP requests*.

🛑Request Journey

```
Browser → Embedded Tomcat → DispatcherServlet
        → HandlerMapping → Controller Method
        → returns view name → ViewResolver → JSP
```

☸️DispatcherServlet = the core of Spring MVC.

✡️3. Project Setup — pom.xml (JSP Support)

Spring Boot does NOT support JSP out of the box.
Add these dependencies:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

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

<dependency>
    <groupId>jakarta.servlet</groupId>
    <artifactId>jakarta.servlet-api</artifactId>
    <scope>provided</scope>
</dependency>

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

✔️ `tomcat-embed-jasper` → enables JSP engine

⤴️4. Folder Structure

```
src/main/java/com.example.demo/controller/HomeController.java
src/main/webapp/WEB-INF/views/home.jsp
src/main/webapp/WEB-INF/views/result.jsp
src/main/resources/application.properties
```

🛑5. Configure JSP in application.properties

```
spring.mvc.view.prefix=/WEB-INF/views/
spring.mvc.view.suffix=.jsp
```

Now:

```
return "home"; → loads /WEB-INF/views/home.jsp
```

🧳6. Creating Your First Controller

```
@Controller
public class HomeController {

    @GetMapping("/")
    public String home() {
        return "home";   // home.jsp
    }
}
```

☸️MUST use `@Controller`.
`@RestController` will return **plain text**, not JSP.

◀️7. Reading Request Parameters

Let's build a simple addition form.

🔱home.jsp

```
<form action="add" method="get">
    Enter number 1: <input type="text" name="num1"><br>
    Enter number 2: <input type="text" name="num2"><br>
    <button type="submit">Add</button>
</form>
```

🔱Controller Method (using HttpServletRequest)

```
@GetMapping("/add")
public String add(HttpServletRequest req, Model model) {
    int a = Integer.parseInt(req.getParameter("num1"));
    int b = Integer.parseInt(req.getParameter("num2"));

    int result = a + b;
    model.addAttribute("result", result);

    return "result";   // result.jsp
}
```

🛞**Yes, HttpServletRequest is also available in Spring Boot**, because it still runs on Servlet API.

🌠8. Sending Data from Controller → JSP

We use the **Model** object.

```
model.addAttribute("result", result);
```

In JSP we access:

```
Result is: ${result}
```

🕉9. result.jsp

```
<h2>Addition Result:</h2>
<p>The answer is: ${result}</p>
```

JSP uses **Expression Language (EL)** to display model data.

🅱10. Using @RequestParam (Cleaner Version)

```
@GetMapping("/add2")
public String add2(@RequestParam int num1,
                   @RequestParam int num2,
                   Model model) {
    model.addAttribute("result", num1 + num2);
    return "result";
}
```

🔄11. Redirect vs Forward

✔️Forward (default)

```
return "home";
```

Does NOT change URL.

✔️Redirect

```
return "redirect:/";
```

Changes URL in browser.

---

12. Full Working Example — Complete Web App

✔️Controller

```
@Controller
public class WebController {

    @GetMapping("/")
    public String home() {
        return "home";
    }

    @PostMapping("/submit")
    public String submit(@RequestParam String username, Model model) {
        model.addAttribute("name", username);
        return "result";
    }
}
```

✔️home.jsp

```
<form action="submit" method="post">
    Enter name: <input type="text" name="username">
    <button type="submit">Submit</button>
</form>
```

✔️result.jsp

```
<h1>Hello, ${name}</h1>
```

---

13. How Request Object Works in Spring Boot MVC

Even though Spring Boot controllers aren't raw servlets, the system underneath still is.

Behind the scenes:

```
Browser → Tomcat → DispatcherServlet → HandlerMethod → Controller
```

So Spring can inject: - `HttpServletRequest` - `HttpServletResponse` - `HttpSession` - `Cookie[]` - `ServletContext`

Example:

```
@GetMapping("/agent")
public String getAgent(HttpServletRequest req, Model m) {
    String ua = req.getHeader("User-Agent");
    m.addAttribute("ua", ua);
    return "agent";
}
```

🧳14. Summary

You learned how to:

✔️Create a Spring Boot Web MVC project
✔️Configure JSP correctly
✔️Understand DispatcherServlet
✔️Use @Controller
✔️Map GET and POST requests
✔️Read form data
✔️Inject HttpServletRequest
✔️Send data to JSP using Model
✔️Build a fully working web flow

If you want, I can add: ✅Form validation (BindingResult)
✅Sessions & Cookies
✅JSTL tutorial
✅A full CRUD Student App with JSP + DB

🔵WHY “MODEL” IN MVC IS NOT DATABASE MODEL (IMPORTANT CONCEPT)

Many beginners get confused because:

✔️We call database entity classes = **Model**
✔️MVC also has something called **Model**

But they are **NOT the same**.

✅1. In MVC (Spring MVC + JSP)

Model = data you want to display on the view.

Examples: - Strings - Integers - Lists - Java Objects (DTOs) - Actual database entities

You send this data to JSP using:

```
model.addAttribute("students", list);
```

This is **purely for the view**.

⬜It has **nothing to do with the database layer**.

✅2. In Database Layer (JPA / Hibernate)

"Model" usually refers to: - `@Entity` - Represents a **table schema** - Maps columns → Java fields

Example:

```
@Entity
public class Student {
    @Id
    private int id;
    private String name;
}
```

This is **Database Model**, not MVC model.

✅3. How They Work Together

Database Model (Entity) → loaded by Service → given to Controller → added as MVC Model → shown on JSP.

Flow:

```
Database → Entity → Service → Controller → Model → JSP View
```

🔵WHY DO WE USE HttpServletRequest IN SPRING MVC?

Even in Spring Boot, you can do:

```
public String add(HttpServletRequest req)
```

Because:

✔️Spring MVC is built on top of Servlets

Every request passes through:

```
Tomcat → DispatcherServlet → Controller
```

So you still get: - HttpServletRequest - HttpServletResponse

✔️When do we use HttpServletRequest?

- To read raw parameters manually
- To access session
- To access cookies
- To read headers manually
- When working with JSP forms

Example:

```
String name = req.getParameter("name");
```

🔵WHY NOT @RequestBody OR @PathVariable IN JSP MVC?

Because JSP forms DO NOT send JSON.

They send:

```
application/x-www-form-urlencoded
```

Example form:

```
<form action="/add" method="post">
   <input name="num1" />
   <input name="num2" />
</form>
```

So Spring MVC uses: ✔️@RequestParam
❌@RequestBody (for JSON)
❌@PathVariable (for RESTful URLs)

REST API (JSON-based)

Use: - @RequestBody - @PathVariable - @ResponseBody

Web MVC (JSP + forms)

Use: - @RequestParam - Model - HttpServletRequest
