JSP Files in Spring Boot

give me the deep and detailed explaination of JSP Files in spring boot so that i can able to understand it well and do not have any type of doubt in this.

so give me the deep and detailed explaination

Alright Piyush 👍
Since you already have **strong backend fundamentals and Spring MVC knowledge**, I'll explain **JSP files in Spring Boot** from **absolute basics → internal flow → real-world usage**, so that **no conceptual gaps remain**.

I'll keep the language **simple, structured, and practical**, exactly the way you prefer.

JSP Files in Spring Boot — Deep & Complete Explanation

1️⃣ What is JSP (JavaServer Pages)?

JSP is a **server-side view technology** used to create **dynamic HTML pages** using **Java code**.

👉 JSP allows you to:

- Write **HTML + Java**
- Display **dynamic data** coming from the backend
- Act as the **View layer** in **MVC architecture**

JSP is part of **Java EE (Jakarta EE)** and is built on top of **Servlets**.

📌 Important fact:

> **Every JSP file is internally converted into a Servlet by the server**

2️⃣ Why JSP in Spring Boot?

Spring Boot is mostly used for **REST APIs**, but it also supports **server-side rendering** using JSP.

JSP is used when:

- You want **traditional MVC web apps**
- You want **HTML rendered on server**
- You don't want frontend frameworks (React/Angular)
- You want **tight integration with Java**

3️⃣ JSP in MVC Architecture (Big Picture)

Spring Boot follows **MVC** using **Spring MVC**

```scss
Client (Browser)
      ↓
Controller (Java)
      ↓
Model (Data)
      ↓
View (JSP)
      ↓
HTML Response
```

Roles:

Layer	Responsibility
Controller	Handles request
Model	Holds data
View (JSP)	Displays data

4️⃣ JSP vs HTML (Very Important Difference)

HTML	JSP
Static	Dynamic
No Java	Java supported
Runs on browser	Runs on server
Cannot access backend data	Can access model data

5️⃣ JSP Execution Flow (Internal Working)

This is **CRITICAL** to understand.

Step-by-step flow:

1. Browser requests `/login`
2. Request goes to **DispatcherServlet**
3. DispatcherServlet calls Controller
4. Controller returns **view name**
5. ViewResolver finds `.jsp`
6. JSP is converted to **Servlet**
7. Servlet generates HTML
8. HTML sent to browser

📌 JSP **never goes directly to browser**
📌 Only **generated HTML** goes

6️⃣ JSP Needs Servlet Container

JSP cannot run alone.

It requires a **Servlet Container** like:

- Apache Tomcat
- Jetty
- Undertow

Spring Boot **embeds Tomcat**, so you don't need external setup.

7️⃣ Why JSP is NOT Default in Spring Boot?

Spring Boot prefers:

- Thymeleaf
- REST APIs

JSP limitations:

- No hot reload
- Tight coupling with server
- Not cloud-native friendly

⚠ JSP **works best with WAR deployment**, not JAR.

8️⃣ JSP Project Structure in Spring Boot

This is **very important**.

```
css


src
 └── main
     ├── java
     ├── resources
     └── webapp
         └── WEB-INF
             └── jsp
                 ├── home.jsp
                 └── login.jsp
```

📌 JSP **must be inside** `WEB-INF`

📌 Otherwise, it becomes publicly accessible (security issue)

9️⃣ JSP Configuration in Spring Boot

Spring Boot needs **explicit configuration**.

application.properties

```
properties
```

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

📌 This tells Spring:

- Where JSP files are
- What extension to use

🔟 How Controller Returns JSP

Controller Code

```java
@Controller
public class HomeController {

    @GetMapping("/home")
    public String home(Model model) {
        model.addAttribute("name", "Piyush");
        return "home"; // resolves to home.jsp
    }
}
```

JSP Code (home.jsp)

```jsp
<h1>Welcome ${name}</h1>
```

📌 `${name}` comes from **Model**

1️⃣1️⃣ JSP Expression Language (EL)

EL makes JSP **clean and readable.**

Syntax

```jsp
```

```
${variable}
```

Examples

```jsp
${user.name}
${user.email}
${list[0]}
```

📌 EL replaces ugly Java scriptlets

1️⃣2️⃣ JSP Scriptlets (❌ Avoid This)

```jsp
<%
   int x = 10;
   out.println(x);
%>
```

❌ Bad practice
❌ Breaks MVC
❌ Hard to maintain

✅ Use **EL + JSTL**

1️⃣3️⃣ JSTL (JavaServer Pages Standard Tag Library)

JSTL removes Java code from JSP.

Common JSTL Tags

Tag	Purpose
`<c:if>`	condition
`<c:forEach>`	loop
`<c:choose>`	switch

Tag	Purpose
`<c:out>`	print safely

Example

```jsp
<c:forEach items="${users}" var="u">
  <p>${u.name}</p>
</c:forEach>
```

📌 JSTL keeps JSP **pure view**

1️⃣4️⃣ JSP Lifecycle (Very Important)

Every JSP follows this lifecycle:

1. Translation → JSP → Servlet
2. Compilation → `.java` → `.class`
3. Initialization → `jspInit()`
4. Request Handling → `_jspService()`
5. Destruction → `jspDestroy()`

📌 Happens automatically

1️⃣5️⃣ JSP vs Thymeleaf (Interview Level)

JSP	Thymeleaf
Old	Modern
Servlet-based	Template-based
Needs container	Container independent
Hard to test	Easy to test
Not cloud friendly	Cloud friendly

📌 JSP still used in **legacy enterprise apps**

1️⃣6️⃣ When SHOULD You Use JSP?

Use JSP when:

- Maintaining **legacy projects**
- Traditional **MVC apps**
- No SPA frontend
- Learning **Servlet + MVC deeply**

1️⃣7️⃣ When NOT to Use JSP?

Avoid JSP when:

- Microservices
- REST APIs
- Cloud-native apps
- React/Angular frontend

1️⃣8️⃣ Real-World Analogy (Easy)

Think of JSP as:

> **HTML template with placeholders**

Controller:

> **Chef preparing data**

JSP:

> **Plate arranging food**

Browser:

> **Customer receiving dish**

1️⃣9️⃣ Common JSP Mistakes (Avoid These)

❌ Business logic in JSP
❌ Scriptlets
❌ Direct DB access
❌ JSP outside `WEB-INF`
❌ Returning `.jsp` in controller

2️⃣0️⃣ Final Summary (One Shot)

✔️ JSP is **view layer**
✔️ JSP runs on **server**
✔️ Converted to **Servlet internally**
✔️ Uses **Model data**
✔️ Works with **Spring MVC**
✔️ Needs **ViewResolver**
✔️ Best for **legacy MVC apps**

If you want, next I can explain:

- JSP + Spring Security flow
- JSP vs REST architecture
- Complete JSP project (login + form)
- How JSP internally becomes Servlet (code-level)

Just tell me 👍