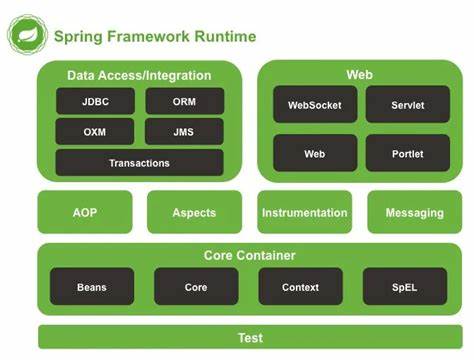
23/07/24

* @Component
* @Configuration
* @ComponentScan
* @SpringBoot
* @PostInitiate
* @PreDestroy
* @RequestMapping
* @PostMapping
* @GetMapping
* @RequestHeader
* @RequestParam
* @RequestBody
* @Configuration

Hava -j terminal/(jar name)🡪 used to run the jar file that we have created.

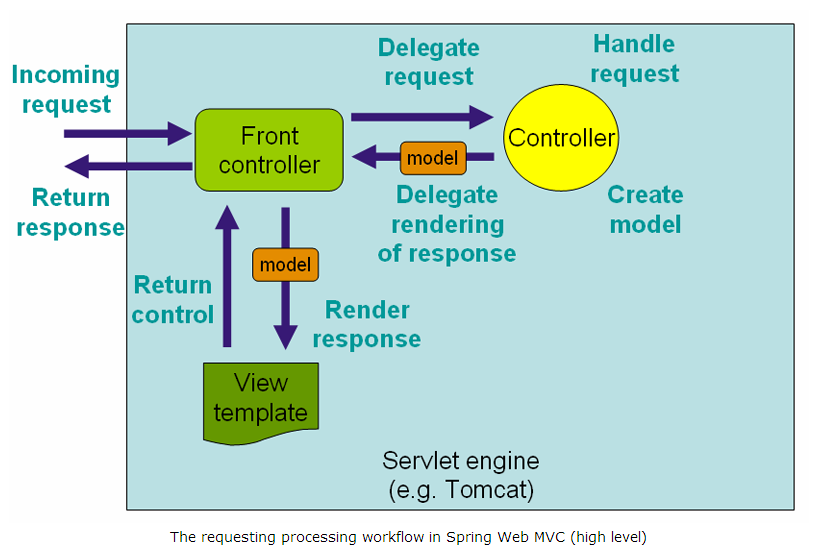


Portlets🡪are creating website in a website.

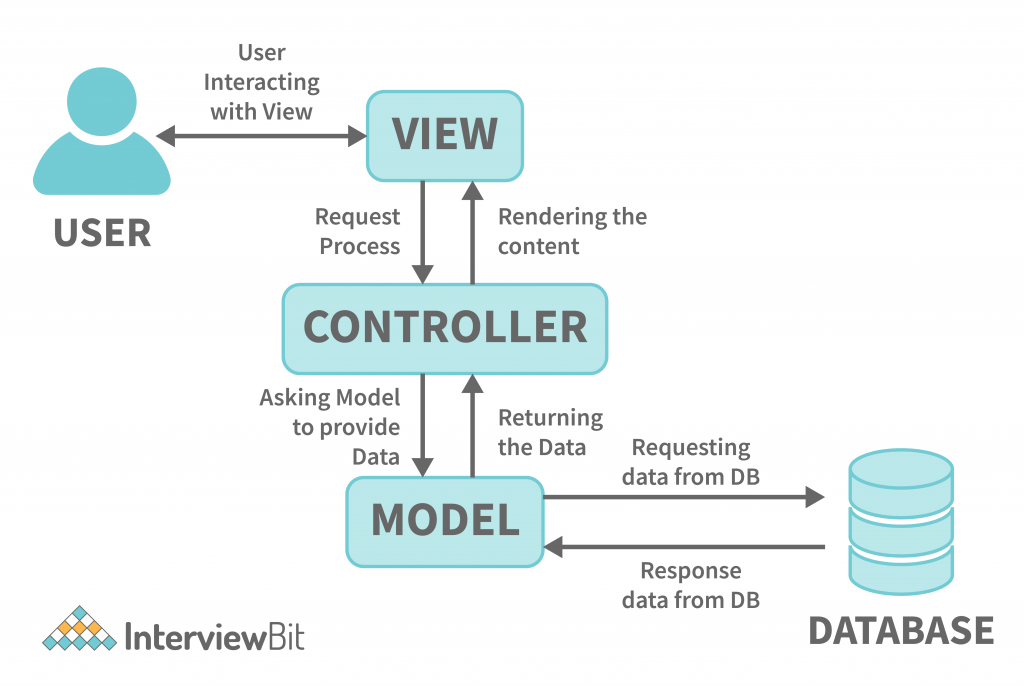
Spring OXM stands **for Spring Object XML Mappers**and it is a module available in Spring to ease the mapping between java objects and XML documents.

[Spring Web MVC :: Spring Framework](https://docs.spring.io/spring-framework/reference/web/webmvc.html)

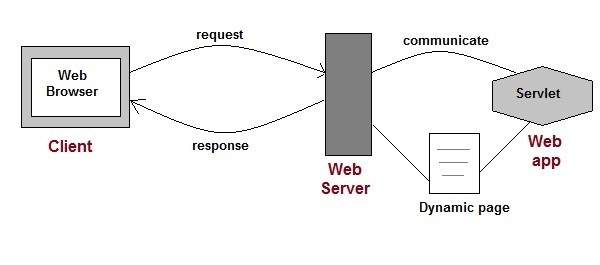
Spring mv architecture🡪



**MVC:**



**SERVLETS:**



Simply put, a Servlet is a class that handles requests, processes them and reply back with a response.

For example, we can use a Servlet to collect input from a user through an HTML form, query records from a database, and create web pages dynamically.

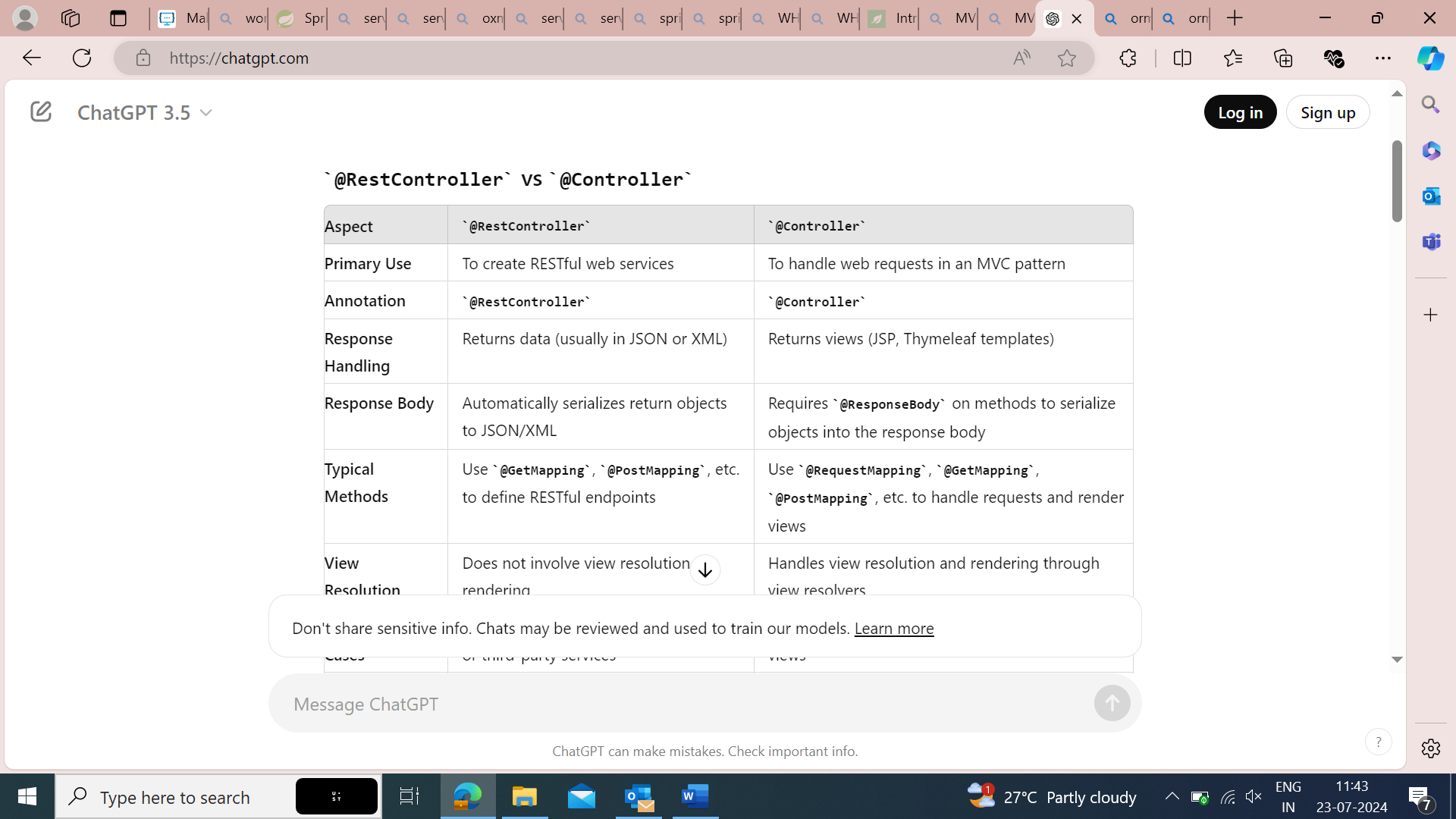
Servlets are under the control of another Java application called a **Servlet Container.**When an application running in a web server receives a request, the Server hands the request to the Servlet Container – which in turn passes it to the target Servlet

[Introduction to Java Servlets | Baeldung](https://www.baeldung.com/intro-to-servlets)

* In mvc the view responses; in rest we directly write into views

In the Spring Framework, \*\*`@RestController`\*\* and \*\*`@Controller`\*\* (typically used in the context of MVC) are two different annotations that serve distinct purposes for handling HTTP requests. Let’s break down the differences and clarify your points:

### `@RestController` vs `@Controller`



|  |  |  |
| --- | --- | --- |
| **Typical Use Cases** | APIs for web applications, mobile apps, or third-party services | Web applications where the server generates HTML views |

|  |  |  |
| --- | --- | --- |
| **Data Serialization** | Yes, the data is automatically serialized and returned | No, typically deals with HTML templates or redirects |

`@**RestController**`

- \*\*Purpose\*\*: Designed for REST APIs. It simplifies the creation of RESTful web services by eliminating the need for explicit `@ResponseBody` annotations on methods.

- \*\*Response\*\*: Methods return domain objects or collections, which are automatically serialized to JSON or XML based on the content type requested by the client.

example:

```java

@RestController

public class MyRestController {

@GetMapping("/api/data")

public Data getData() {

return new Data("example");

}

}

```

In this example, `getData()` returns a `Data` object which is automatically serialized into JSON.

**`@Controller`**

- \*\*Purpose\*\*: Primarily used for MVC web applications where the server-side application logic interacts with user interfaces through views.

- \*\*Response\*\*: Methods typically return view names that are resolved to actual HTML pages or templates. If you need to return JSON or XML, you must use `@ResponseBody` explicitly.

- \*\*Example\*\*:

```java

@Controller

public class MyMvcController {

@GetMapping("/home")

public String home(Model model) {

model.addAttribute("message", "Hello World");

return "home"; // Name of the view template (like home.jsp or home.html)

}

}

```

In this example, `home()` returns `"home"` which is resolved to a view template, and the `Model` object holds the data for that view.

**View** **Resolution**

- \*\*`@RestController`\*\*: Directly returns data objects (JSON/XML). There is no view resolution.

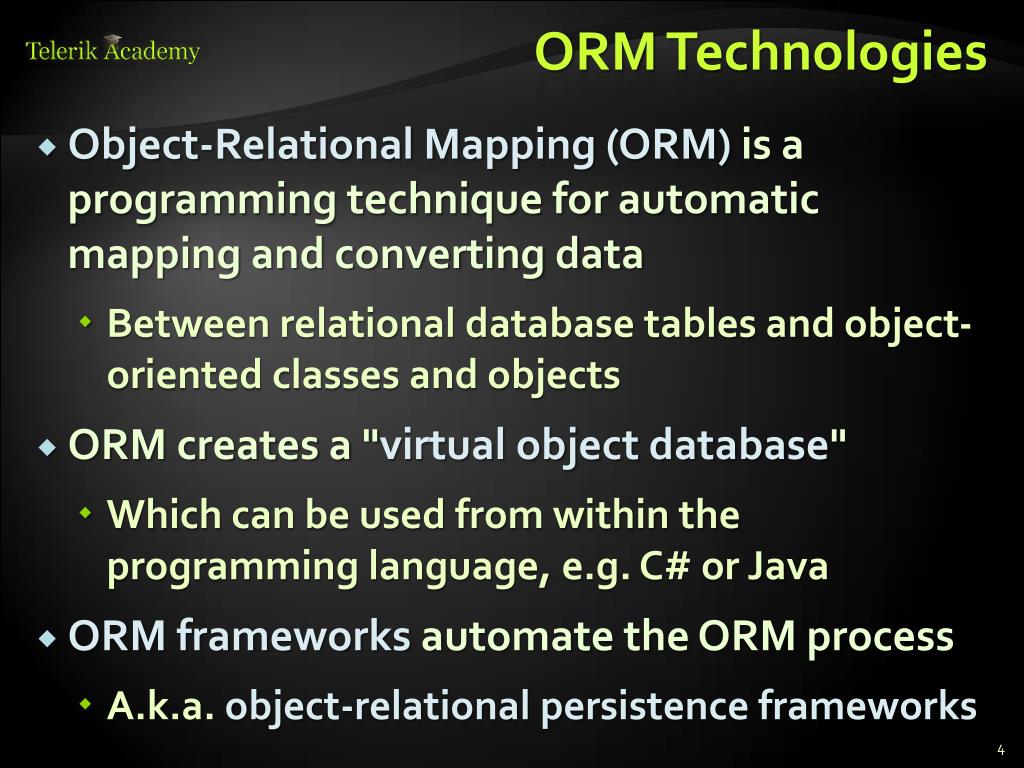
- \*\*`@Controller`\*\*: Returns view names which are resolved into HTML pages or templates by a `ViewResolver`.

**Summary**

- \*\*`@RestController`\*\* is for creating REST APIs and directly returns data as JSON/XML.

- \*\*`@Controller`\*\* is for handling web requests in an MVC pattern, typically returning view names for rendering HTML.

Both annotations can coexist in the same application but are used in different contexts based on whether you're building a RESTful API or an MVC-based web application.



* Impedance mismatch is a term used in computer science to describe the problem that arises when two systems or components that are supposed to work together have different data models, structures, or interfaces that make communication difficult or inefficient.
* [What is Object/Relational Mapping? - Hibernate ORM](https://hibernate.org/orm/what-is-an-orm/)

|  |  |  |
| --- | --- | --- |
| * **Feature** | **JSP (JavaServer Pages)** | **EJB (Enterprise JavaBeans)** |

|  |  |  |
| --- | --- | --- |
| **Purpose** | View layer for generating dynamic web content | Business logic layer for enterprise applications |

|  |  |  |
| --- | --- | --- |
| **Type** | Presentation technology | Component technology for server-side business logic |

|  |  |  |
| --- | --- | --- |
| **Role in MVC** | View component | Model component for business logic |

|  |  |  |
| --- | --- | --- |
| **Lifecycle** | Translated into servlets, then executed | Managed by the EJB container (injection, transactions) |

|  |  |  |
| --- | --- | --- |
| **Common Use** | Creating HTML/JSP pages | Handling complex business logic, transactions |

|  |  |  |
| --- | --- | --- |
| **Components** | Pages (JSP files) | Beans (Session Beans, Message-Driven Beans, etc.) |

|  |  |  |
| --- | --- | --- |
| **APIs** | Java EE JSP API, JSTL | Java EE EJB API |

|  |  |  |
| --- | --- | --- |
| **Access** | Directly interacts with HTTP requests and responses | Interacts through interfaces, remote or local calls |

