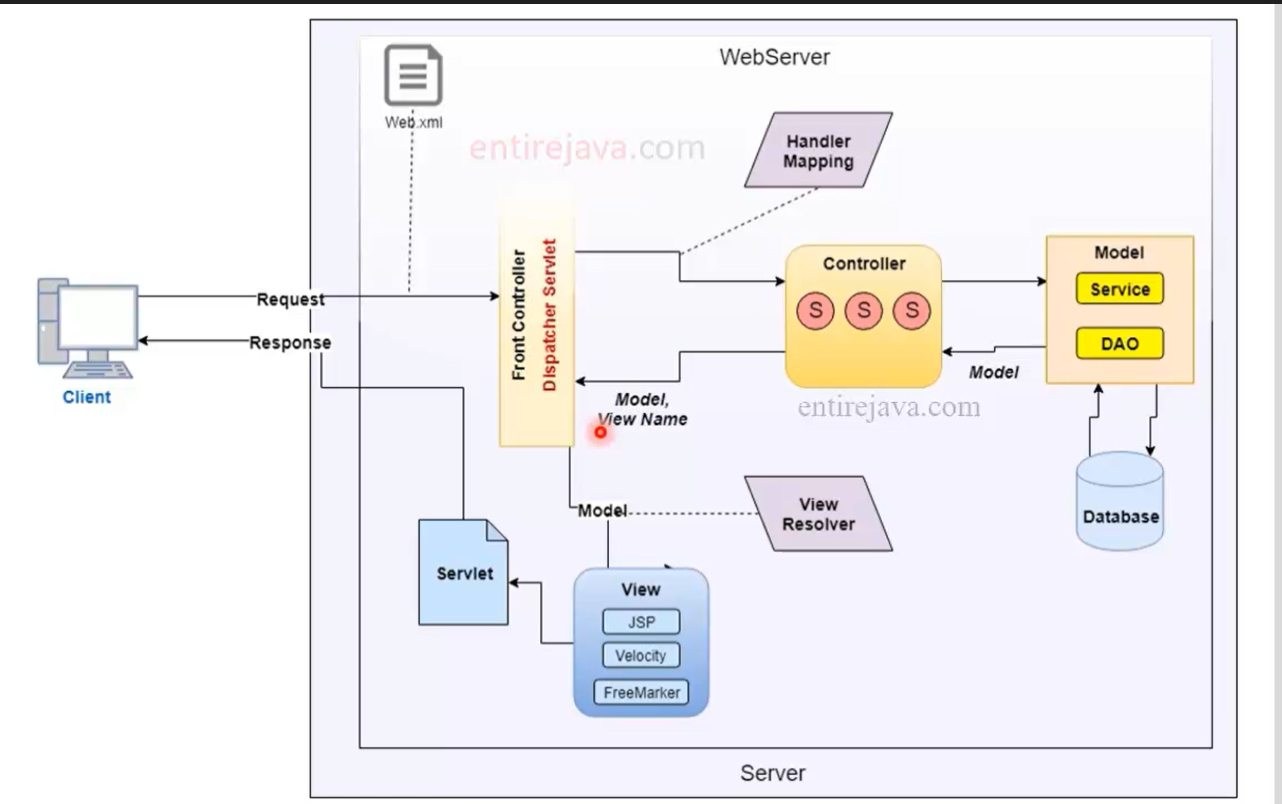
**Spring MVC**

The Spring Web MVC framework provides Model-View-Controller (MVC) architecture and ready components that can be used to develop flexible and loosely coupled web applications.

* The **Model** encapsulates the application data and in general, they will consist of POJO.
* The **View** is responsible for rendering the model data and in general it generates HTML output that the client's browser can interpret.
* The **Controller** is responsible for processing user requests and building an appropriate model and passes it to the view for rendering.

A *DispatcherServlet* that handles all the HTTP requests and responses.





The difference between traditional MVC and Spring MVC is Spring I.e., Dispatcher Servlet. When an HTTP request is received from the Client. It will go to the Web.XML file and tries to look for a mapping. Here in Spring MVC, we will map to Front Controller not directly to the servlets. *DispatcherServlet* consults the *HandlerMapping* to call the appropriate *Controller*. We need to explicitly specify the URL mapping inside our config file. But in with annotations we don’t need to specify URL. The *Controller* takes the request and calls the appropriate service methods based on the used GET or POST method. The service method will set model data based on defined business logic. The Service layer communicates with DAO Objects or Data Access Object to able to communicate with the Database. Then we will get the Result, In Spring terms results are called as Model. The Model is known as an Object that holds the data. The servlet will send the model to the end-user along with the View name( doesn’t send any data like where it is residing or file extension) to the Front Controller. The Front Controller will send the model object to the View with the help of *ViewResolver* to pick up the defined view for the request which is configured in the Spring Config file. Then it will convert the file into servlets and sends the response to the Client.

The **web.xml** file will be kept in the WebContent/WEB-INF directory of your web application

* The *[servlet-name]-servlet.xml* file will be used to create the beans defined, overriding the definitions of any beans defined with the same name in the global scope.
* The *<context:component-scan...>* tag will be used to activate Spring MVC annotation scanning capability which allows to make use of annotations like @Controller and @RequestMapping etc.
* The *InternalResourceViewResolver* will have rules defined to resolve the view names. As per the above-defined rule, a logical view named **hello** is delegated to a view implementation located at */WEB-INF/jsp/hello.jsp* .

The **@Controller** annotation indicates that a particular class serves the role of a controller.

The **@RequestMapping** annotation is used to map a URL to either an entire class or a particular handler method.

REST:

**RESTful web services** are built to work best on the Web. Representational State Transfer (REST) is an architectural style that specifies constraints, such as the uniform interface, that if applied to a web service induce desirable properties, such as performance, scalability, and modifiability, that enable services to work best on the Web. In the REST architectural style, data and functionality are considered resources and are accessed using **Uniform Resource Identifiers (URIs)**, typically links on the Web. The resources are acted upon by using a set of simple, well-defined operations. The REST architectural style constrains an architecture to a client/server architecture and is designed to use a stateless communication protocol, typically HTTP. In the REST architecture style, clients and servers exchange representations of resources by using a standardized interface and protocol.

The **@RestController** annotation is used to define the RESTful web services. It serves JSON, XML, and custom response.

The **@RequestMapping** annotation is used to define the Request URI to access the REST Endpoints. We can define the Request method to consume and produce an object. The default request method is GET.

The **@RequestBody** annotation is used to define the request body content type.

The **@PathVariable** annotation is used to define the custom or dynamic request URI.

The **@RequestParam** annotation is used to read the request parameters from the Request URL. By default, it is a required parameter.

The default HTTP request method is GET. This method does not require any Request Body. You can send request parameters and path variables to define the custom or dynamic URL.

The HTTP POST request is used to create a resource. This method contains the Request Body. We can send request parameters and path variables to define the custom or dynamic URL.

The HTTP PUT request is used to update the existing resource. This method contains a Request Body. We can send request parameters and path variables to define the custom or dynamic URL.

The HTTP Delete request is used to delete the existing resource. This method does not contain any Request Body. We can send request parameters and path variables to define the custom or dynamic URL.

A postman is a tool that allows you to test your APIs - send requests, retrieve responses. **Postman**is a scalable API testing tool that quickly integrates into CI/CD pipeline. It started in 2012 as a side project by Abhinav Asthana to simplify API workflow in testing and development. API stands for Application Programming Interface which allows software applications to communicate with each other via API calls.

Advantages:

1. Accessibility - To use the Postman tool, one would just need to log in to their accounts making it easy to access files anytime, anywhere as long as a Postman application is installed on the computer.
2. Use of Collections - Postman lets users create collections for their Postman API calls. Each collection can create subfolders and multiple requests. This helps in organizing your test suites.
3. Collaboration - Collections and environments can be imported or exported making it easy to share files. A direct link can also be used to share collections.
4. Creating Environments - Having multiple environments aids in less repetition of tests as one can use the same collection but for a different environment. This is where parameterization will take place which we will discuss in further lessons.
5. Creation of Tests - Test checkpoints such as verifying for successful HTTP response status can be added to each Postman API call which helps ensure [test coverage](https://www.guru99.com/test-coverage-in-software-testing.html).
6. Automation Testing - Through the use of the Collection Runner or Newman, tests can be run in multiple iterations saving time for repetitive tests.
7. Debugging - Postman console helps to check what data has been retrieved making it easy to debug tests.
8. Continuous Integration - With its ability to support continuous integration, development practices are maintained.