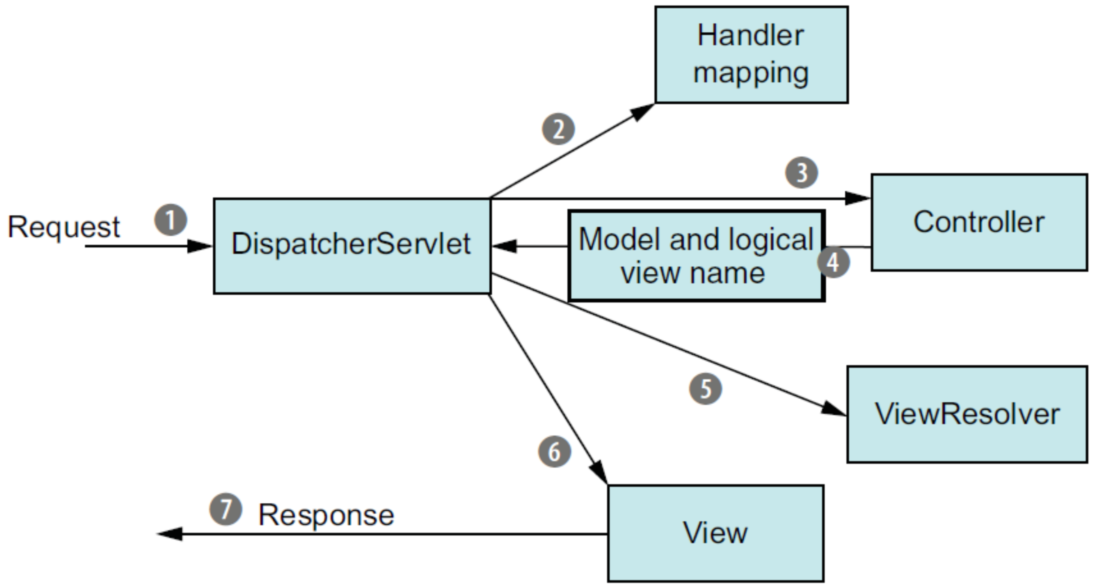
Building Spring web applications:

As an enterprise Java developer, you've likely developed a web-based application or two. For many Java developers, web-based applications are their primary focus. If this is your experience, then you're well aware of the challenges that come with these systems. Specifically, state management, workflow, and validation are all important features that need to be addressed. None of these is made any easier given the HTTP protocol's stateless nature. Spring's web framework is designed to help you address these concerns. Based on the Model-View-Controller (MVC) pattern, Spring MVC helps you build webbased applications that are as flexible and as loosely coupled as the Spring Framework itself.

Spring moves requests between a **dispatcher servlet, handler mappings, controllers, and view resolvers**.
Each of the components in Spring MVC performs a specific purpose. And it's really not that complex.

Following the life of a request

- Every time a user clicks a link or submits a form in their web browser, a request goes to work. A request's job description is that of a courier. Just like a postal carrier or a FedEx delivery person, a request lives to carry information from one place to another:



1. When the request leaves the browser, it carries information about what the user is asking for. At the least, the request will be carrying the requested URL. But it may also carry additional data, such as the information submitted in a form by the user. The first stop in the request's travels is at Spring's DispatcherServlet. Like most Java based web frameworks, Spring MVC funnels requests through a single front controller servlet. A *front controller* is a common web application pattern where a single servlet delegates responsibility for a request to other components of an application to perform actual processing. In the case of Spring MVC, **DispatcherServlet is the front controller**.

2. The DispatcherServlet’s job is to send the request on to a Spring MVC controller. ***A controller* is a Spring component that processes the request**. But a typical application may have several controllers, and DispatcherServlet needs some help deciding which controller to send the request to. So the DispatcherServlet consults one or more handler mappings to figure out where the request’s next stop will be. **The handler mapping pays particular attention to the URL carried by the request when making its decision.**
3. Once an appropriate controller has been chosen, DispatcherServlet sends the request on its merry way to the chosen controller. At the controller, the request drops off its payload (the information submitted by the user) and patiently waits while the controller processes that information. (Actually, a well-designed controller performs little or no processing itself and instead delegates responsibility for the business logic to one or more service objects.)
4. The logic performed by a controller often results in some information that needs to be carried back to the user and displayed in the browser. This information is referred to as the *model*. But sending raw information back to the user isn’t sufficient—it needs to be formatted in a user-friendly format, typically HTML. For that, the information needs to be given to a view, typically a JavaServer Page (JSP). One of the last things a controller does is package up the model data and identify the name of a view that should render the output. It then sends the request, along with the model and view name, back to the DispatcherServlet
5. So that the controller doesn’t get coupled to a particular view, the view name passed back to DispatcherServlet doesn’t directly identify a specific JSP. It doesn’t even necessarily suggest that the view is a JSP. Instead, it only carries a logical name that will be used to look up the actual view that will produce the result. The DispatcherServlet consults a view resolver to map the logical view name to a specific view implementation, which may or may not be a JSP.
6. Now that DispatcherServlet knows which view will render the result, the request’s job is almost over. Its final stop is at the view implementation, typically a JSP, where it delivers the model data. The request’s job is finally done.
7. The view will use the model data to render output that will be carried back to the client by the response object