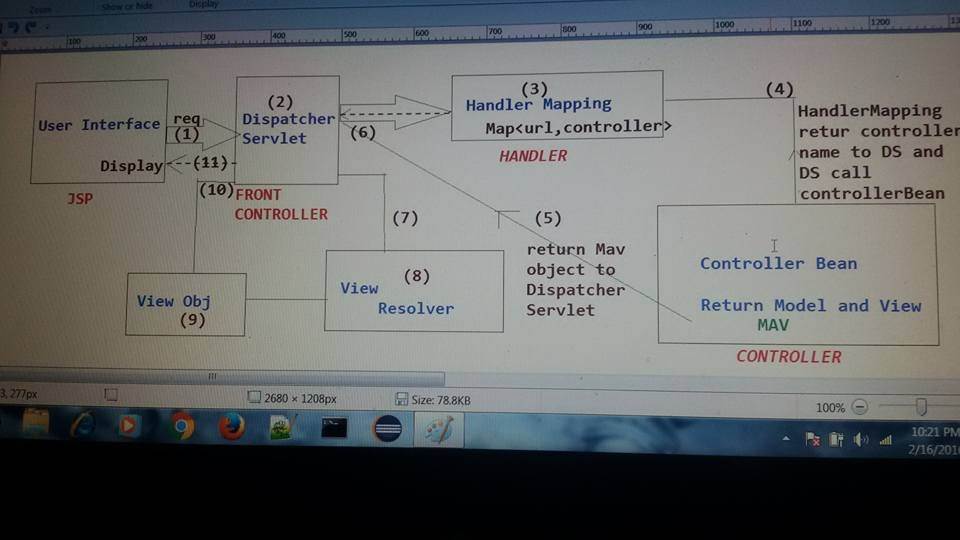
MVC:

**Model view controller** is a software architecture design pattern. It provides solution to layer an application by separating three concerns business, presentation and control flow.

* The **Model** can be some DAO layer or some Service Layers which give some information about request or requested information or Model can be a POJO which encapsulates the application data given by the controller.
* 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 appropriate model and passes it to the view for rendering.

The Flow of Spring MVC is attache in the Screenshots.



First request is trapped by the **Dispatcherservlet**, then dispatcher servlet forward to the **handlermapping**, Handlermapping identify the appropriate handler or controller class. The controller class perform operartion like Dao or business then the resultant page with data will be forward to **dispatcherservlet,** the dispatcger servlet forward it with logical view name to **view resolver**, view resolver resolves it and create the view object , it calls the render method on view object then the view page will be displayed

Well this seems like an interesting one:

The only answer that I can tell you is this: it depends! Why, you might ask, well cope with me a while longer.

It all comes down to your software architecture in the middle term and how is the pattern and layers that you designed for your software. But if I abstract that and think that the person designed as he/she should and following the good patterns, we would have something like this:

1. Let’s imagine that our user is in a nice jsp page and wants to search for something, and that's what he does! He hits the search button
2. Our nice Javascript would take charge of sending a pretty (not usually) JSON with a GET request to a Spring Controller sitting on the back end. Let's call it CustomerController.
3. CustomerController being the smart guy that it is, knows that he should't process any information and just pass it through, so he passes the ball to CustomerService.
4. Well, according to our bussines rule, CustomerService needs to get some data from database to continue with its nice logic, but as it shouldn't connect directly to the repository, he calls CustomerDAO.
5. CustomerDAO who is in love with the database, but wouldn't mess directly with it, calls CustomerRepository
6. Then, CustomerRepository process your query and goes to the database to retrieve our result! Now the data starts to go back because it wants to appear to everyone!
7. CustomerRepository handles data to CustomerDAO
8. CustomerDAO returns the data to CustomerService
9. CustomerService finishes its logic and handles the result to CustomerController
10. CustomerController, a pure gentleman transforms your data into JSON and returns it to the frontend

And voila! That would be the process, in short, the flow is something like this:

Jsp -> Controller -> Service -> DAO -> Repository -> Database -> Repository -> DAO -> Service -> Controller -> Jsp

Hope to have clarified it a little bit other than confusing you more.

Good programming!

The flow will be in sequence JSP->Spring Controller->Spring Services->Daos->Hibernate->MySQL.

Any request made by client first get passed to spring controller by JSP ( assuming you’r using JSP for only view part). Then, Controller will get request object & based on the request URL will decide which Service function to call. The service function will communicate with hibernate DAOs if needed. Hibernate then will fetch data from database & will return it to services function as POJO( plane old java object). This POJO will be processed by services function & will be returned to calling controller function. Then the controller function will return it to JSP to display.