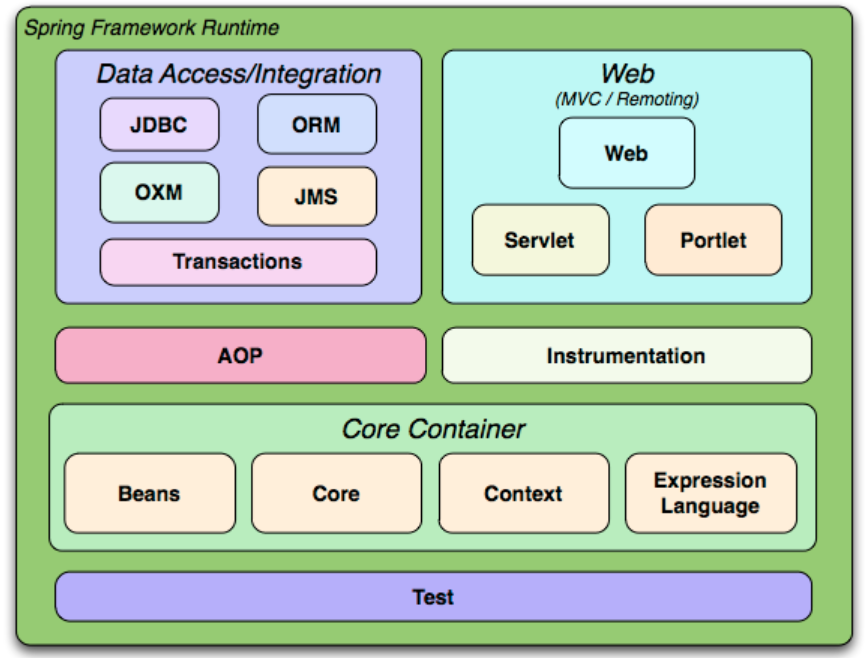
**Spring Modules – Introduction**

**Architecture**



**Reference**: <https://docs.spring.io>

Spring Framework Modules can be highly classified into,

**Core Container** acts as the main/base module of the Spring which contains the below sub modules such as Beans, Core, Context & Expression Language

**Web Layer** is responsible for creating Web Applications. Sub Modules includes Web, Servlet & Portlet

**Data Access/Integration Layer** is responsible for interacting with the database & Integration with ORM frameworks. Sub Modules includes JDBC, ORM, OXM, JMS & Transactions.

**Test Layer** has its own Testing framework which has good binding & support for testing the application using Junit or Mockito or TestNG.

**Miscellaneous Layer** includes AOP (Aspect Oriented Programming) & Instrumentation.

**Spring Modules Explanation**

**Core Container – Beans & Core**

Core Container, the brain of the Spring Module is responsible for implementing the **Inversion of Control (IoC) or Dependency Injection (DI)**. This helps in achieving the loose coupling between classes.

In a conventional way, developers will have the control over the code in creating the objects & injecting the dependencies. Here the Spring frameworks takes the control of doing the above-mentioned activities, that’s why the term is coined as ***“Inversion of Control”*** - i.e., the control is inversed!

**Dependency Injection** is a pattern which adheres/implemented Inversion of Control for resolving the dependencies.

Objects, which acts as the backbone of your application & are managed (i.e., instantiation, configuration & assembling) by the IoC Container are called as **Beans**.

Until here, you should be clear about IoC concepts. But how does the beans get instantiated? It is taken care by means of **Autowiring**. Spring framework makes use of Autowiring concepts for instantiating the beans into the required class at the time of booting (starting) up the application.

**Spring Context**

Spring Context is a sub-module of Core Container which has been built on the base of Spring Core & Beans. It is mainly responsible for instantiating & configuring the beans by scanning the configurations from Spring annotations or XML files.

Additionally, it provides support for Internationalization(i18N), resource loading & event-propagation.

**Internationalization (i18N)**

Spring Internationalization helps in designing the Web Application to be accessed/adapted to various languages across the world. This will help the end users to access/walk through the application or website in their own language.

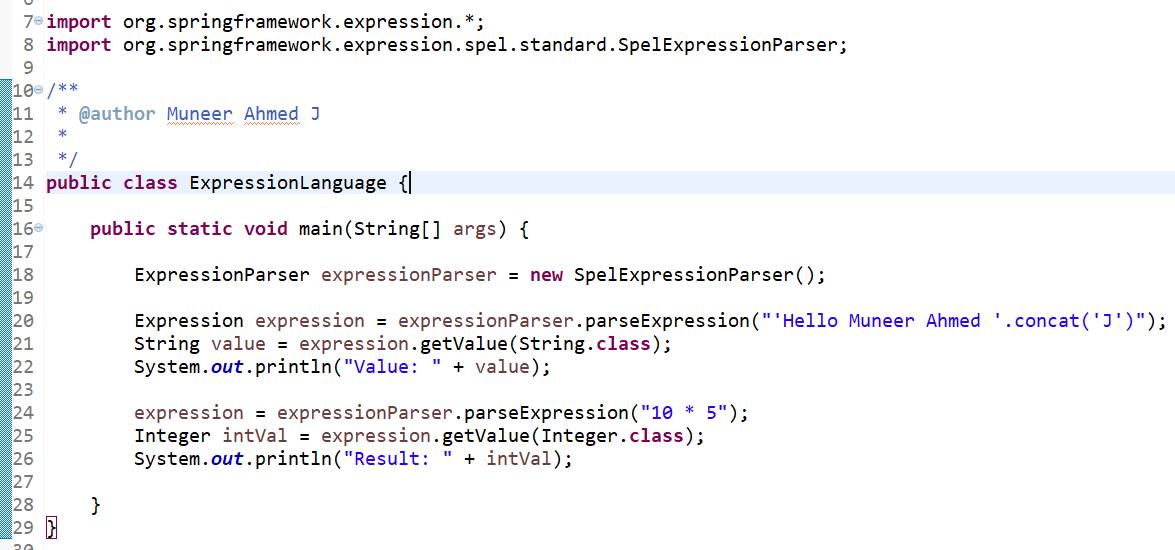
i18N can be achieved by configuring the messages/sentences in message-{language}.properties file.

Example, for English – the words & sentences can be maintained in message-en.properties whereas for Hungarian, it can be maintained in message-hgr.properties file.

**Spring Expression Language (SpEL)**

SpEL is a Spring based Expression Language for evaluating the expression such as Mathematical Operations, Logical & Relation Operations, Conditional Operations & RegEx.

**Real time example:**



**Output**

Value: Hello Muneer Ahmed J

Result: 50

**Data Access/Integration Layer**

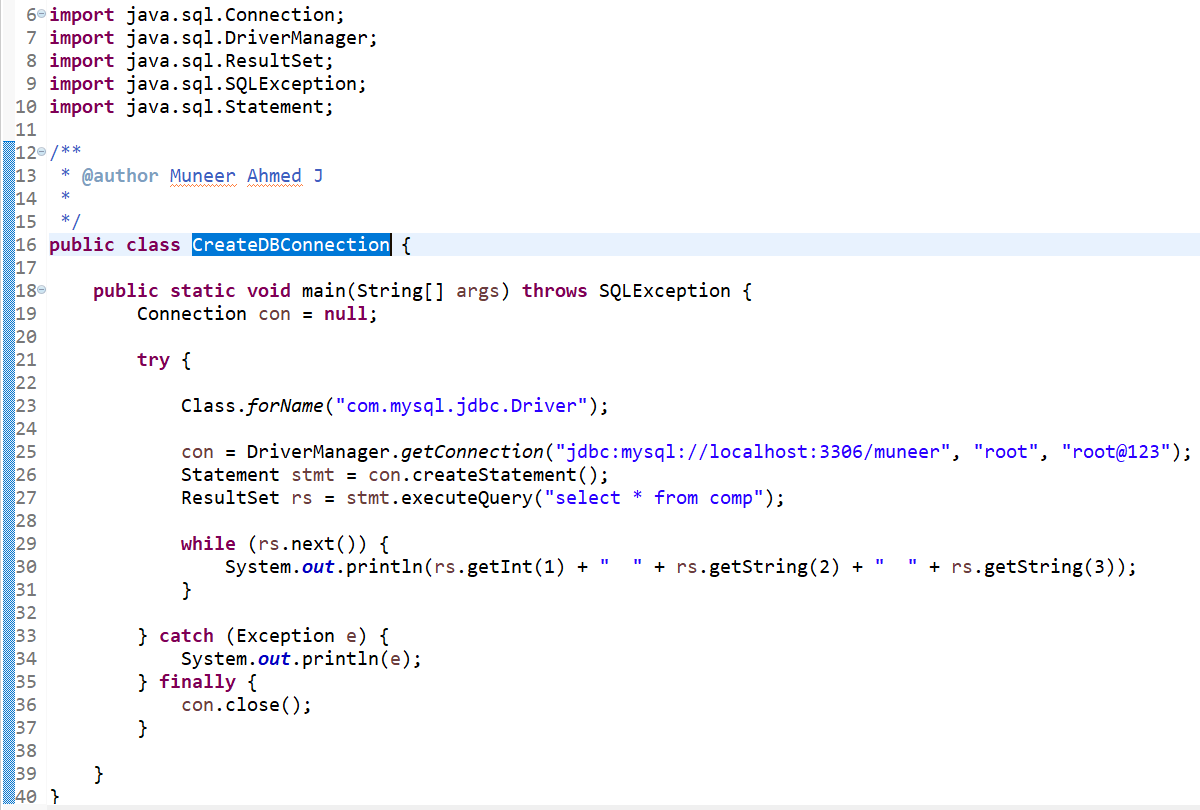
**JDBC Layer**

JDBC Layer provides a wrapper or an abstraction over the conventional JDBC.

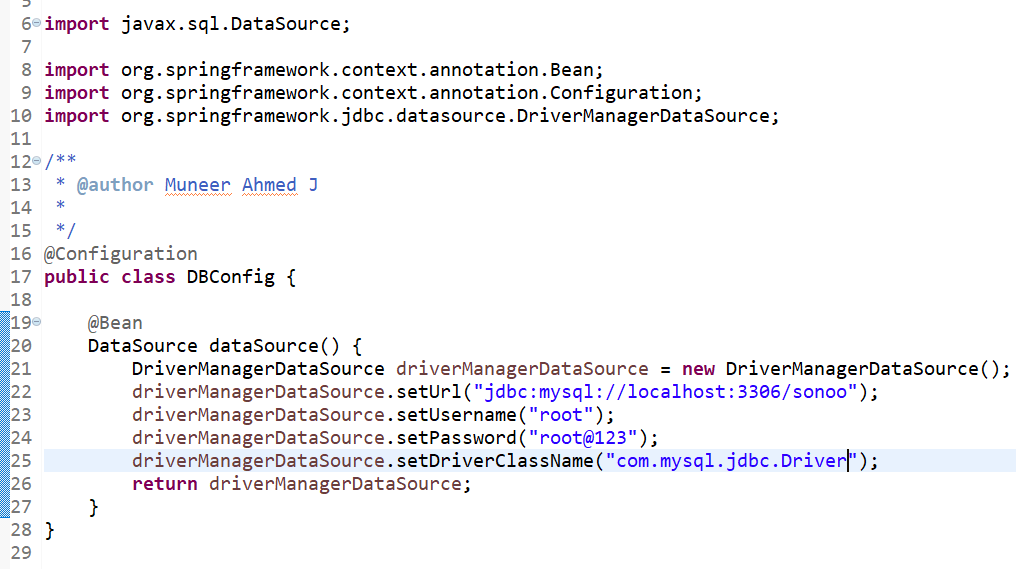
In JDBC Connection, developers needs to write boiler plate coding for Creating, Opening & Closing of DB Connection, Executing Queries etc., But with the help **Spring** **JDBC Template**, the Spring Framework takes cares of the above-mentioned tasks.

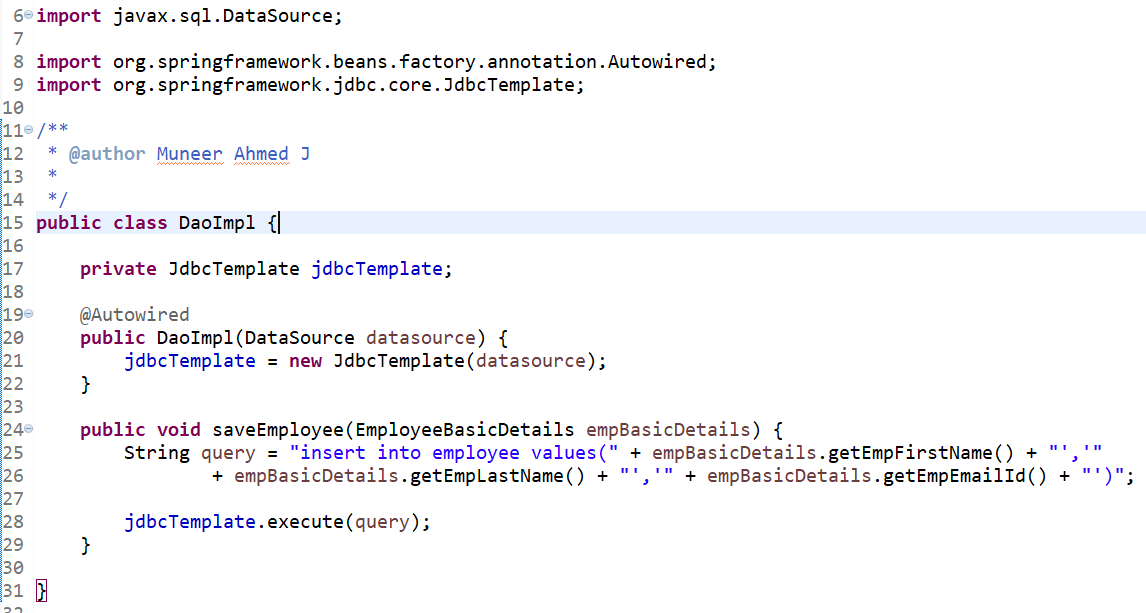
Additionally, JDBC Templates are thread safe, i.e., with a single instance of JDBC Template, we can make use of it in different DAO classes.

**Conventional way of establishing a JDBC Connection**



**Using Spring JDBC Template**





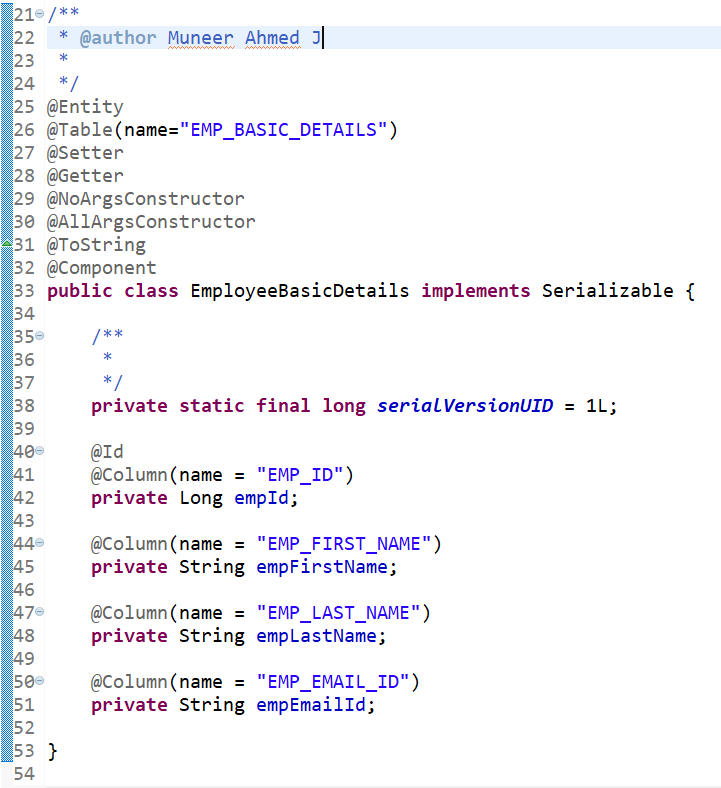
DaoImpl class which makes use of JDBC Template for executing the queries. Here you can notice that, boiler plate codes i.e., explicitly registering the drivers, opening/closing the DB Connection have not been used.

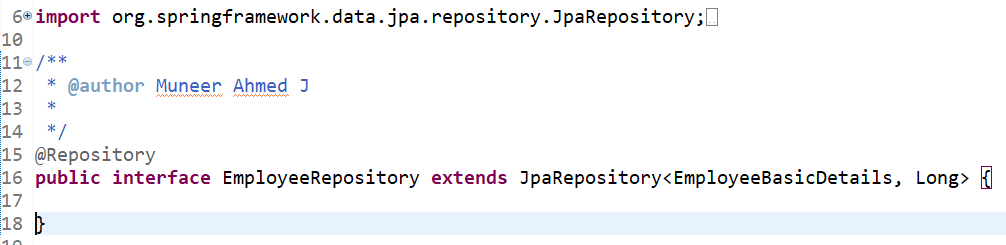
**ORM Layer**

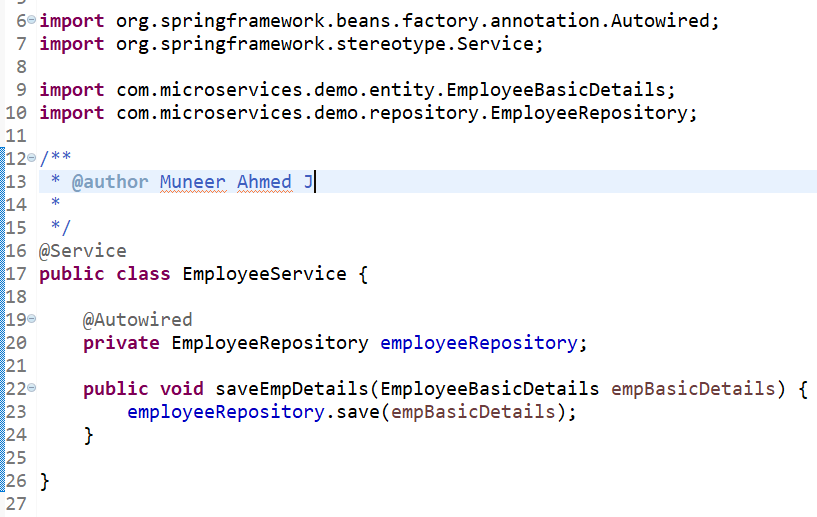
Spring ORM (Object Relational Mapping) Layer provides support for integrating with various ORM frameworks such as Hibernate, JPA etc., Spring doesn’t provide any changes in the ORM layer, but it provides some additional features for handling the ORM layer efficiently.

To make things clear, JPA is only a specification. To make use of it, you will need an implementation for it. Spring Data JPA is one such implementation.

Real time example of Spring Data JPA







EmployeeRepository class acts as the Repository layer has been extended to JpaRepository. Spring Data JPA Layer provides readymade methods such as save, findById, deletebyId etc.,

As you can see in EmployeeService class, we have not written any query for saving the Object, instead we are making of in-built ‘**save’** method for saving the employee basic details object into database.

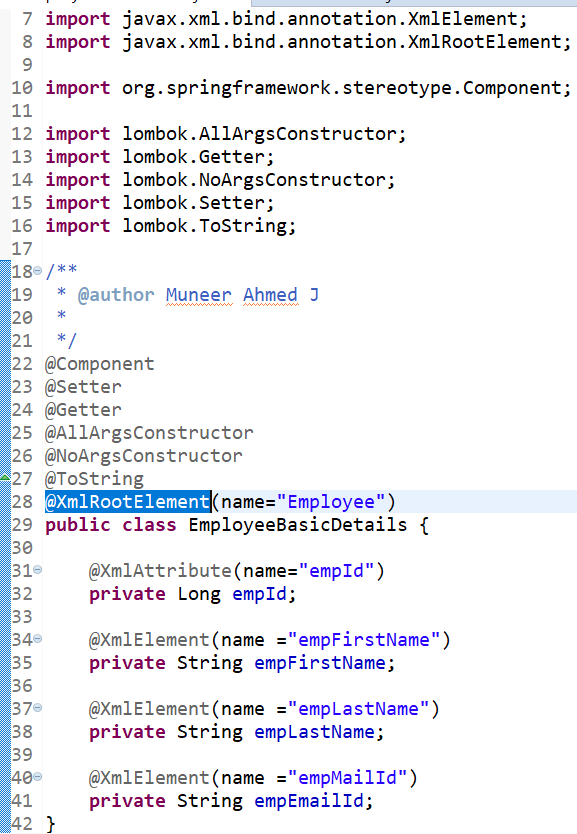
**OXM Layer**

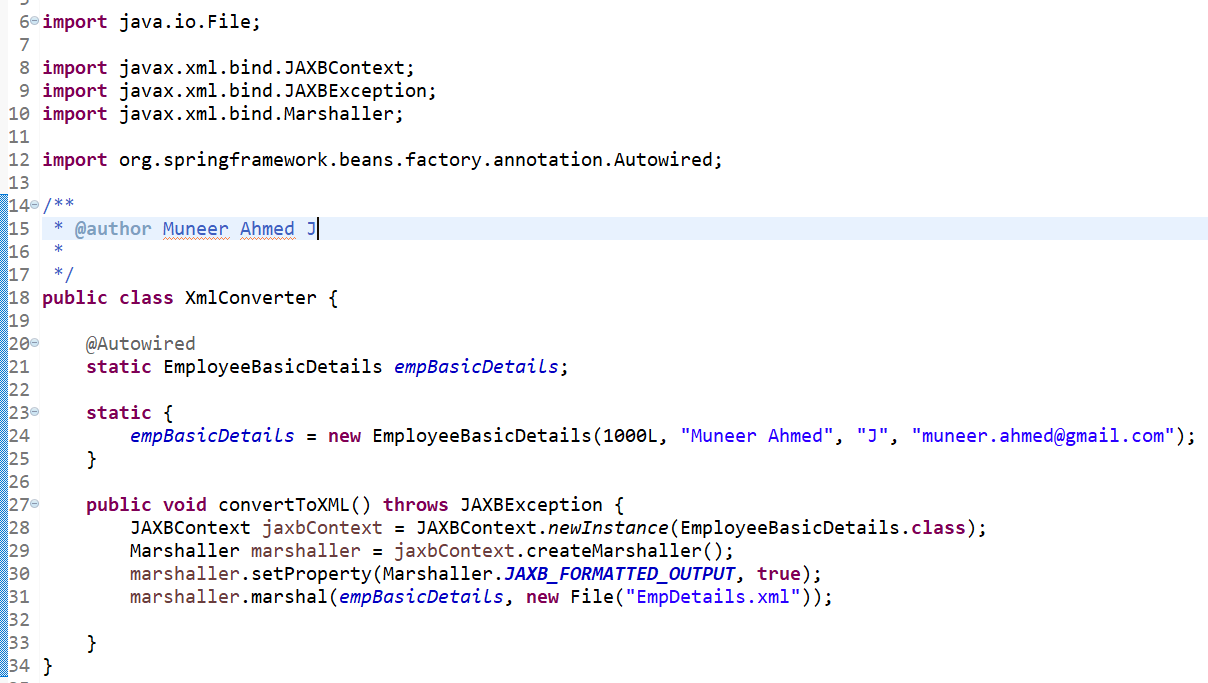
Spring OXM Layer provides a wrapper or an abstraction layer for supporting Java Object to XML Document mappings & vice versa (i.e., XML Marshalling & Unmarshalling) with various implementation frameworks such as JAXB, XStream, Castor etc.,

**Real time Use Case:**

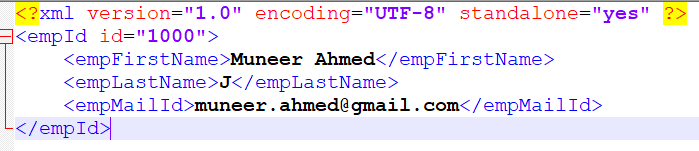
In SOAP Web Service Environment, we need to accept the requests and return the response to the clients in XML format. In this case, developers needs to convert the XML into Java Objects for processing (Marshalling) and after processing, the Java Object needs to converted back into XML Response & send it back to the Clients.

**Real time example using JAXB**





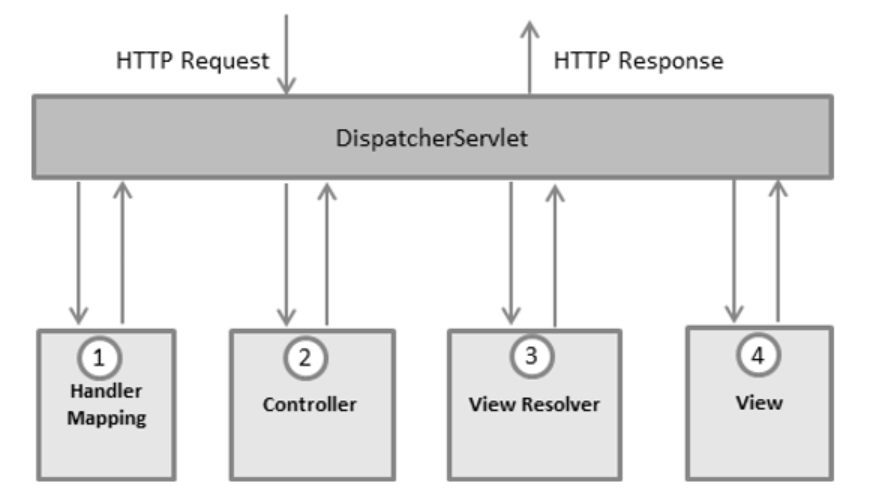
**Output Response**:



**Spring Web/MVC Layer**

Spring MVC refers to Model, View & Controller.

**Architecture of Spring MVC**



**Reference**: <https://tutorialspoint.com>

Steps of Spring MVC Model

* Once a HTTP request is made from the Client layer, Dispatcher Servlet will provide the request endpoint to the Handler Mapping to route it to the required controller.
* From Controller, it is routed to the Service Layer to implement any business logics & then to the DAO Layer (in case of any DB transactions).
* Once done, data will be set into Model Objects & then will be routed to the View Layer.
* Here, Dispatcher Servlet will provide the details to the View Resolver in finding the corresponding view.
* Once done, the HTTP response along with Model data objects will be routed to the View (HTML file) which will be rendered in the browser.

To help the Dispatcher Servlet to handle the HTTP request, mapping of the URL will be maintained in web.xml file (i.e., Deployment Descriptor).

**Spring Test Module**

Spring Test Module has a testing framework which provides support for testing the spring components independently using Junit or TestNG.

Since Spring framework uses Dependency Injection for loose coupling, it helps the developer to perform the Unit Testing easily with the help of Junit Framework.

In a traditional way, Integration testing can be performed once after the application is deployed. But Spring provides an easy way to Mock or Stub the objects & perform the Integration Testing.

**Conclusion**

From this article, we have done deep dive of Spring Modules with real time scenarios & code snippets. In a nut shell, Spring Modules helps the developers to handle the application more efficiently with less efforts & coding.