Spring Framework

Spring is the most popular application development framework for enterprise Java. Millions of developers around the world use Spring Framework to create high performing, easily testable, reusable code. Also, Spring framework is an open source Java platform and is lightweight when it comes to size and transparency.

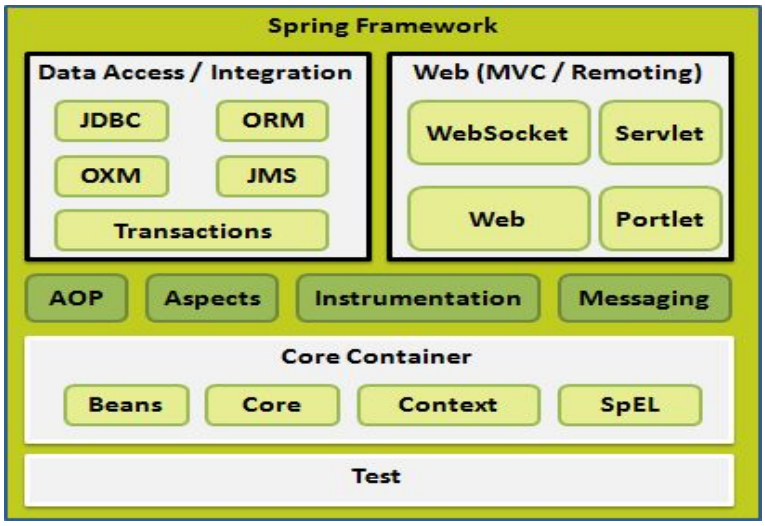
Functional Features:

- Spring enables developers to develop enterprise-class applications using POJOs (Plain Old Java Object).
- Spring is organized in a modular fashion.
- Testing an application written with Spring is simple because environment-dependent code is moved into this framework. Furthermore, by using JavaBean-style POJOs, it becomes easier to use dependency injection (DI) for injecting test data.
- Spring provides a consistent transaction management interface that can scale down to a local transaction (for example, using a single database) and scale up to global transactions (using Java Transaction API).

NonFunctional Features:

- Applications build using Spring are highly powerful as every layer is clearly separated from other layers.
- Spring provides scalability to its applications to handle more requests.
- Cost of Spring in terms of memory is very low i.e 2MB.
- Backup is created by Spring itself whenever there is a failure/crash.

Architecture Design



Elaborating the design

The Spring Framework has about 20 modules which can be used based on an application requirement.

- *Core Container*:
  The Core Container consists of the Core, Beans, Context, and Expression Language modules whose detail is as follows:

a. The **Core** module provides the fundamental parts of the framework, including the IoC and Dependency Injection features.
b. The **Bean** module provides BeanFactory which is a sophisticated implementation of the factory pattern.
c. The **Context** module builds on the solid base provided by the Core and Beans modules and it is a medium to access any objects defined and configured.
d. The **SpEL** module provides a powerful expression language for querying and manipulating an object graph at runtime.

- *Data Access Integration*:
  The Data Access/Integration layer consists of the JDBC, ORM, OXM, JMS and Transaction modules whose detail is as follows:

a. **JDBC** module provides a JDBC-abstraction layer that removes the need to do tedious JDBC related coding.
b. The **ORM** module provides integration layers for popular object-relational mapping
APIs, including JPA, JDO, Hibernate, and iBatis.
c. The **OXM** module provides an abstraction layer that supports Object/XML mapping implementations for JAXB, Castor, XMLBeans, JiBX and XStream.
d. The Java Messaging Service **JMS** module contains features for producing and consuming messages.
e. **Transaction** module supports programmatic and declarative transaction management for classes that implement special interfaces and for all your POJOs.

- *Web:*
  The Web layer consists of the Web, Web-MVC, Web-Socket, and Web-Portlet modules whose detail is as follows:

a. The **Web** module provides basic web-oriented integration features such as multipart file-upload functionality and the initialization of the IoC(Inversion of Control) container using servlet listeners and a web-oriented application context.
b. **Web-MVC** module contains Spring's model-view-controller (MVC) implementation for web applications .

c. **Web-Socket** module provides support for WebSocket-based, two-way communication between client and server in web applications.

d. **Web-Portlet** module provides the MVC implementation to be used in a portlet environment and mirrors the functionality of Web-Servlet module

- *Test:*
  The **Test** module supports the testing of Spring components with JUnit.

Alternative Design:
I think, for better developer control over the framework, the views can be generated directly from the controller instead of having a separate views layer. Rest of the desing is fine.