<http://www.java-samples.com/showtutorial.php?tutorialid=466>

A complete web application requires a web framework to generate HTML pages, receive user input, and manage the navigation flow. The consensus among most web developers is that a Model-View-Controller (MVC) architecture is the best for web frameworks. In an MVC framework, the controller takes user input and decides which view to show next; The view renders the HTML page for the browser; The model encapsulates data captured from the web form and to be displayed on the web page.

There are many web frameworks available. JavaServer Faces (JSF), is the standard Java EE web framework . While JSF does have a leg up on the competition by being a standards-based technology, the decision to embrace JSF over the alternatives goes much deeper.

JSF is a well-designed and easy-to-use component-based web framework. This component-based development model aligns perfectly with the lightweight POJO approach we are promoting. The development model is clean and simple.

**Shield developers from the messy HTML / JavaScript / CSS details**

Enterprise Java developers have long embraced ORM solutions, which automatically generate relational database access code from the data model component in the application. JSF is the "ORM solution for the web layer". It automatically generates HTML / JavaScript / CSS code from well formed UI components.

A JSF application primarily contains two types of components -- both are easy to use and conforming to the POJO philosophy.

* A JSF page is composed of XML tags. Each tag element corresponds to a UI component in JSF. As a web developer, you do not need to deal with the HTML markup or JavaScript details, as they are all generated from the JSF component tags. Since each UI component is self-contained and has well-defined behavior (i.e., it understand how to render itself and how to handle its data), JSF provides a very POJO-like experience for developing web UI.
* Dynamic data on JSF pages and forms are modeled in POJOs known as JSF backing beans. The backing bean lifecycle is managed by the JSF server. For instance, a backing bean can have a session scope to track a user session. A backing bean can be dependency injected into another bean via an XML configuration file, and it can be injected into a JSF UI component via the JSF Expression Language (EL). The backing bean also provides the integration points between the web UI and the EJB3 business components.

The componentized UI and POJO data model make it easy to support JSF in IDE tools. In fact, many Java IDEs now support drag-and-drop UI designers for JSF. The JSF component model also allows third party vendors to develop and sell reusable UI component libraries. Those component libraries make it easy for server-side developers to take advantage of the latest browser technology without the need to mess with JavaScripts and such.

The JSF request model is powerful and easy to extend. Technologies like Facelets and Seam have been developed on top of JSF, providing even richer development environments. While competing frameworks all have nice features, JSF seems to be the easiest to extend. We expect even more innovative extensions in the near future, clearly making JSF the framework of choice for most EE projects.

Of course, JSF isn't without its drawbacks. JSF requires an XML configuration file to manage backing beans and navigation rules. As we will also soon see, the integration between JSF backing beans and EJB3 beans is not entirely smooth. In addition, JSF usage from JSPs can get tricky at times. However, these problems are not entirely unique to JSF. Other frameworks exhibit similar problems. The good news is that the JSF model is advanced enough that solutions to all of the issues have been developed.