Java is a trademark of Sun Microsystems, Inc.

JavaOne℠

Choosing the Right Technology Stack for Your Java™ Website.

Ian Robertson
Sean Landis
Overstock.com

So you want to build a Java™ Website

So many choices

- Container
- UI layer
- Persistence layer
- Build tools

So you want to build a Java website

With so many considerations

- Buy vs Build
- Commercial vs Open Source
- Standards-based vs proprietary
- Standards-based vs Standards-based

> The nice thing about standards is that there are so many to choose from. - Andrew Tanenbaum

Understand your requirements

- Staff skills and experience
- Time and resource availability
- Rapid Application Development vs. Rapid Application Maintenance
- Stability, Reliability, Scalability, *ility...
- The need to interface with other technologies (Database, JMS, AJAX, REST, etc)

Don't let others define your requirements for you

- Vendors, Industry Analysts and hypesters all are willing to do so
  - A product is not the same thing as a solution
  - A solution is not necessarily a solution for any of **your** problems
- Beware of vendor lock-in
  - That said, when there is a solution to your problem, it can be wonderful
- Listen to others who have been where you're going

Remember that things change

- With time
  - Technologies, Companies, Employees and Requirements all come and go
- With scale
  - Don't get caught with your servers down
  - But don't let next years design keep you from reaching next month!

Beware of internal politics

- Staff wants to do new cool technology A
- Staff is afraid of technology B
- Manager just read about technology C
- Manager is afraid of technology D
  - “I hear that Java is slow”
- "Nobody ever got fired for choosing X"
  - But people do get fired for failed projects, cost overruns, etc...

Choosing a web UI framework

So Many Choices:

JSP, Struts1, Struts2, Java Server Faces, Wicket, Spring MVC, WebFlow, Tapestry, Jamon, Velocity, Jboss Seam, Restlet, JAX-RS, GWT, Cocoon, MyFaces, WebWork, Turbine, Rife, Echo, AppFuse, Stripes, ...

There is no one-size-fits-all solution; all frameworks have to make trade-offs

UI Frameworks Trade-offs

- Avoiding repetition (DRY)
- Type safety
- Accessibility to UI designers (non-programmers)
- Performance
- Ramp-up time for developers
- Separation of layers
  - does the business layer need to know about the UI Layer? (annotations?)
- Form handling capabilities

Choosing a Container

- JavaEE
- Spring/Guice
- Tomcat/Jetty
- public static void main(String args[])

JavaEE

- Plenty of providers
  - both commercial and open source
- Swiss Army Knife - can do everything
- Some restrictions
  - Thread creation is not allowed
  - Nor is direct file access

Dependency Injection (Spring, Guice)

- Plenty of documentation
- Large communities of support
  - Easy to find developers who know it.
- Most common needs can be met

Tomcat

- Primarily “just” a servlet container
- A great choice if all you need is a web server
- A good platform for non-standard requirements (build your own)
- Increasingly, Jetty is emerging as another popular alternative for a light-weight server

public static void main(String[] args)

- Gives total control, with higher development costs
- Can make sense if its core to your business
- Do not choose this lightly

Persistence Layer

Here, at least, the answer has always been clear:

- Use JDBC

Persistence Layer

Here, at least, the answer has always been clear:

- ~~Use JDBC~~
- Use EJB/CMP/BMP

Persistence Layer

Here, at least, the answer has always been clear:

- ~~Use JDBC~~
- ~~Use EJB/CMP/BMP~~
- Use JDO

Persistence Layer

Here, at least, the answer has always been clear:

- ~~Use JDBC~~
- ~~Use EJB/CMP/BMP~~
- ~~Use JDO~~
- Use Hibernate

Persistence Layer

Here, at least, the answer has always been clear:

- ~~Use JDBC~~
- ~~Use EJB/CMP/BMP~~
- ~~Use JDO~~
- ~~Use Hibernate~~
- Use JPA

Persistence Layer

Here, at least, the answer has always been clear:

- ~~Use JDBC~~
- ~~Use EJB/CMP/BMP~~
- ~~Use JDO~~
- ~~Use Hibernate~~
- Use JPA
- or maybe iBATIS...

Build Tools

- Ant: the assembly language of build tools
  - Can do any task, but you have to do it your self, and there are problems scaling to large projects
- Maven: the BEPL of build tools
  - It's “easy”, except for the hard stuff
- The industry is still searching:
  - Ivy provides dependency management, but builds on ant

Build vs “Buy”

If it's not an important differentiator, you're probably better off using an off-the-shelf component (ideally open source)

If it's critical and there is a matching solution, use it

But don't be afraid to invent when needed

- Not as much work as some make it out to be, but requires design skill
- The bigger the shop, the more that custom-built components make sense
- Innovation requires this....

Things to consider when “buying”

- Support
- Provider viability
- Community
- Size of user base
- Trend

Commercial vs Open source

Even if you don't plan to change the source, having access can be invaluable for debugging and diagnosis

OpenSource projects often have **better** support than their commercial counterparts

Case Study – Overstock.com

Two separate use cases:

- Public facing websites (i.e. www.overstock.com)
- Internally-facing websites (customer service, item creation, etc)

Public-facing website requirements

- Scalable – 2000 requests/second
- Responsive - < .25 second page load time
- Customizable – never say “we cannot do that” to the business
- Reliable – no site, no money
- Maintainable – lots of developers, lots of changes

Public-facing website non-requirements

- Fast developer ramp-up
- Support from a third party for a framework
- Heavy form processing capabilities

Site revenue allows making non-trivial investments

Public-facing website stack

- Java 6 SE
- Hibernate
- Tomcat
- Jamon (templating engine)
- Restlet (ReSTful backend web services)
- Ehcache (taxonomy data, sales tax rates, etc)
- Oracle Coherence (caching search results)
- JMS (click tracking, order processing)
- JMX (monitoring, cache invalidation, etc)

What's **not** in the stack

- Session – can limit scalability
- EJB container – no need
- JSP – not enough type safety
- Spring – use a lightweight generic factory pattern instead

Internal website requirements

Quick to develop, maintain

Heavy form processing

Small number of users, all trained on using the application

Internal website non-requirements

- Highly scalable
- Highly customizable
- Ability to support parallel development

Internal website stack

- Java 6 SE
- Hibernate
- Tomcat
- **Jboss Seam**
- Restlet (ReSTful backend web services)
- JMX (monitoring, cache invalidation, etc)

Thank You

Ian Robertson
irobertson@overstock.com
Sean Landis
slandis@overstock.com