Full-stack Java web dev with Vaadin

Take advantage of the latest advances in HTML5 and JavaScript in your 100-percent Java web apps

Sing Li

September 30, 2015

Learn how the Vaadin framework puts Java™ developers on the fast track to modern web development. With Vaadin, you don't need to master HTML5, CSS, and JavaScript to create a compelling user experience. The framework deploys those technologies for you while you program in 100-percent Java in your favorite IDE. Get started with hands-on examples.

To create a compelling interactive user experience in web applications, Java developers often must learn the non-Java technologies that underlie modern browsers: JavaScript, CSS, and HTML5. Interfacing between an HTML5/JavaScript front end and a Java back end — typically, via Ajax — tends to require a complex assembly that's difficult to maintain. But now, thanks to the open source Vaadin framework, Java developers can gain full-stack control of their modern web apps with an all-Java programming environment.

Learn more. Develop more. Connect more.

The new developerWorks Premium membership program provides an all-access pass to powerful development tools and resources, including 500 top technical titles (dozens specifically for Java developers) through Safari Books Online, deep discounts on premier developer events, video replays of recent O'Reilly conferences, and more. Sign up today.

As a mature framework, Vaadin supports all popular Java development environments (Eclipse, IntelliJ, NetBeans, Maven, and so on; this tutorial's examples use Eclipse). And Vaadin has hundreds of community-contributed and commercially available UI components. These components are crafted and maintained to work with all modern browsers, and they use the latest available HTML5, CSS, and JavaScript features.

Vaadin's server-side UI-creation support enables you to compose and deploy reusable components in Java technology. The framework manages the complexity of client-side UI creation and management. Integration with all the mid-tier and back-end Java technologies you're already familiar with — such as Enterprise Java Beans (EJB) technology, the Java Persistence API (JPA), and Spring — is easy. Vaadin's server-centric nature makes the framework compatible with all modern cloud-hosted solutions (including IBM Bluemix™), so you can readily deploy your highly
interactive web applications in the cloud. Vaadin is also compatible with many other programming languages in the JVM-powered ecosystem, including Python (Jython), Groovy, and Scala.

Put your Vaadin skills to work


This tutorial introduces Vaadin, reveals its architecture, explains how it works, and offers multiple hands-on examples that familiarize you with Vaadin development in an Eclipse environment. A final Jython example highlights Vaadin's flexibility in working with many JVM-supported programming languages.

Inside Vaadin

Architecturally, Vaadin is similar to AWT, Swing, and SWT. You instantiate and wire together ready-to-go components on the server side, and the framework automatically generates and deploys the UI elements. The main difference between Vaadin and those traditional Java toolkits is that instead of desktop GUI components rendered using Java (or through the specific graphical API of the supported OS) the components are HTML5/CSS/JavaScript components, communicating via Ajax with the server and managed automatically by the framework.

Figure 1 shows the architecture of Vaadin.

Figure 1. Vaadin architecture

Client-side programming model

This tutorial focuses on the more popular server-side programming model of Vaadin. Vaadin also supports a lesser-known client-side programming model. Code running in the browser can access widgets and remotely call server-side methods but must first be compiled from Java to JavaScript.

In Figure 1, you can see how Vaadin completely shields the developer from the complexity of the UI-rendering technology stack. As a Vaadin developer, you create your UI, including event handlers for various interactions, by wiring prefabricated components using Java inside a Java servlet (a VaadinServlet). Then Vaadin takes care of the rest. Together, the client-side Vaadin engine (written in HTML5/CSS/JavaScript, and based on the Google Web Toolkit) and the server-side framework support realize (generate) and manage the UI for you. The network of server-side UI components is realized by a network of client-side UI widgets in the browser.
Server-side UI components can support data binding in Vaadin, making it easy to work with back-office databases. Your Java application code can continue to take advantage of any features offered by the server, including Java Platform, Enterprise Edition (Java EE) and web services.

In accordance with industry best practices, presentation is separated from business logic when you use Vaadin. You can create layouts declaratively using Vaadin HTML design documents, or by programming in Java. Optionally, you can further customize and tune the look of the application by using themes in CSS or Sass.

**Get up and running with Vaadin**

With an understanding of all the pieces that make up Vaadin, you’re almost ready to start building something with the framework. To try the following examples, you need to install:

- Java Development Kit (JDK) 1.7 or later (1.7.0_60-b19 was used in the development of the example code).
- Eclipse EE Edition (Luna or later).
- Tomcat 8. If you don’t already have the Tomcat 8 server configured in Eclipse, follow these instructions to configure it.
- The Vaadin Eclipse Plugin. Find the Vaadin Plugin for Eclipse in the Eclipse marketplace, as shown in Figure 2.
Figure 2. Vaadin Plugin for Eclipse in Eclipse Marketplace

The declarative UI example later in this tutorial makes use of the Vaadin designer. To save your design, you'll need a trial license of the Vaadin designer. Your Vaadin Eclipse Plugin installation can also include a beta of the UI designer; accept the license and restart Eclipse.

Create and run your first Vaadin application

Follow these steps in Eclipse to create the Vaadin starter app:

1. Select File > New > Other > Vaadin > Vaadin 7 Project, as shown in Figure 3.
2. Name your project `FirstProject`, then select `Project > Build Project`. If you're using JDK 1.7, you might need to adjust the Eclipse project aspects to include Java 1.7 instead of 1.8.

3. Highlight the project in the Enterprise Explorer pane, right-click, select `Run As > Run on server`, then select the Tomcat 8 server to run your application.

When you create the project, an app is generated with two Vaadin components: a `VerticalLayout` and a `Button`. Figure 4 shows the app in operation. Every time you click the button in the UI, a new `Vaadin Label` component is created. This simple app is great for learning the rudiments of Vaadin development.

Figure 4. Wizard-generated Vaadin starter app

**Component composition**

Figure 5 illustrates component composition. A `VerticalLayout` contains a `Button`. One or more `Labels` can be created and added to the `VerticalLayout` when the `Click Me button` is clicked.
Listing 1 shows the code that wires up the components in Figure 5.

Listing 1. Code from the generated skeletal app

```java
public class FirstprojectUI extends UI {

@WebServlet(value = "/*", asyncSupported = true)
@VaadinServletConfiguration(productionMode = false, ui = FirstprojectUI.class)
public static class Servlet extends VaadinServlet {
    
    protected void init(VaadinRequest request) {
        final VerticalLayout layout = new VerticalLayout();
        layout.setMargin(true);
        setContent(layout);

        Button button = new Button("Click Me");
        button.addClickListener(new Button.ClickListener() {
            public void buttonClick(ClickEvent event) {
                layout.addComponent(new Label("Thank you for clicking"));
            }
        });
        layout.addComponent(button);
    }
}
```

You construct your UI in one or more classes that extend the abstract `com.vaadin.ui.UI` class. The `init()` method is where the UI is constructed. Note the `@WebServlet` annotations around the `Servlet` class — derived from the `com.vaadin.server.VaadinServlet` — that will serve and manage the UI you create.

Event handling in Vaadin

In Listing 1, `button.addClickListener()` adds a callback for the button click. In this case, the click listener on the button simply creates a new `Label` component with the message "Thank you for clicking" and adds it to the `VerticalLayout` component. Each new line of "Thank you for clicking" appears below the existing label.

The next example goes much further and entails some more-complex Vaadin components, but the operation remains basically the same as in this first example.

Composing more-complex components

For the next example, load the ArticleViewer.zip project file from the sample code (see Download). Figure 6 shows the composition of Vaadin components in this example.
In Figure 6, the `VerticalLayout` contains a `HorizontalLayout`. The `HorizontalLayout` is populated by a `Table` component on the left and an `Embedded` component on the right.

Figure 7 shows the article-viewer application's UI. The app consists of a Vaadin table displaying a list of articles and their publication years on the left. The user selects any article from the list to fetch the article and display it within an `Embedded` (browser) Vaadin component on the right.

You'll find the code in the `ArticleViewer` class, structured the same way and served by the `VaadinServlet` as in the first example. As in the previous example, the code that creates the UI in the class's `init()` method, as shown in Listing 2.

**Listing 2. Code for the article-viewer app**

```java
@Override
protected void init(VaadinRequest request) {
    final VerticalLayout layout = new VerticalLayout();
    layout.setMargin(true);
    setContent(layout);

    HorizontalLayout horiz = new HorizontalLayout();
    horiz.addStyleName("outlined");

    Table table = new Table("Select article");
    table.addContainerProperty("Year", String.class, null);
    table.addContainerProperty("Article", String.class, null);

    int i = 1;
```
for (Object[] row: listdata) {
    table.addItem(getRowData(row), i++);
}
table.setSelectable(true);
table.setImmediate(true);

horiz.setSizeFull();      horiz.addComponent(table);

final Embedded e = new Embedded("Article view", new ExternalResource(
    getURL(listdata[0])));
e.setAlternateText("Article View");
e.setType(Embedded.TYPE_BROWSER);
e.setSizeFull();
horiz.addComponent(e);
horiz.setExpandRatio(e, 1);

In the Listing 2 code, selection of a row is handled by the ItemClickEvent listener. In this case, the handler finds the URL associated with the clicked article and sets the source property of the Embedded component and causes the article to be displayed.

Note also that in Listing 2, table data is populated by iterating through an array of listdata elements. This keeps our code short and simple.

Data binding in Vaadin

The same Vaadin Table element also supports data binding. If you import the ArticleViewerDatabinding.zip project file from the sample code, you will find an enhanced version of ArticleViewer that performs proper data binding with the Table element.

When compared to the original ArticleViewer project, you will see some differences:

- Article is a Plain Old Java Object (POJO) class that holds the data for a single article.
- ArticleService is a service class supplying a static data source; in production code, this service should fetch data from a database.
- ValueChangeListener is used instead of ClickListener on the Table element to support keyboard navigation.
• BeanItemContainer is used to directly bind Article POJOs to the Table element.

**Vaadin declarative UI**

Instead of constructing the UI with the Java programming language a line at a time, you can specify all the components and their containment relationships in a specially formatted HTML Vaadin *design* document. Vaadin can read this HTML document and generate the entire UI for you in one go.

Also, a visual drag-and-drop Vaadin designer tool is available for authoring UI design documents without writing a single line of Java code or HTML. Figure 8 shows the designer in action.

**Figure 8. Vaadin visual drag-and-drop UI designer**

For shops where UI designers and application developers work independently of each other, this feature can be especially applicable.

The next example creates the exact same UI as the article-viewer app, but now using a declarative HTML design document instead of Java code to specify the UI.

Import the DeclarativeUI.zip project into Eclipse and examine the ArticleViewer.html design document, shown in Listing 3. This Vaadin HTML design document specifies the UI and layout. Compare this code with the code in Listing 2 to see the similarities.

**Listing 3. Vaadin HTML design document**

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8" name="design-properties" ...
```
Custom components

The code that reads and realizes the UI based on the ArticleViewer.html design document is in ArticleViewDesign.java — a custom component. This class is also where the table is populated with data and the event handler is hooked up. Listing 4 shows how.

Listing 4. Processing and working with a declarative UI design document

```java
public ArticleViewDesign()
{
    Design.read("ArticleViewer.html", this);

    table.addContainerProperty("Year", String.class, null);
    table.addContainerProperty("Article", String.class, null);

    int i = 1;
    for (Object[] row: listdata )
    {
        table.addItem(getRowData(row), i++);
    }

    table.setSelectable(true);
    table.setImmediate(true);
    e.setType(Embedded.TYPE_BROWSER);
    e.setSource(new ExternalResource(getURL(listdata[0])));
    table.addItemClickListener(new ItemClickListener()
    {
        @Override
        public void itemClick(ItemClickEvent event)
        {
            // TODO Auto-generated method stub
            System.out.println("ID is " + event.getItemId());
            e.setSource(new ExternalResource(getURL(listdata[
                   Integer.parseInt(event.getItemId().toString()) - 1]));
        }
    });
}
```

The custom ArticleView component is instantiated and embedded in the UI by the DeclarativeUI Java class (which also extends com.vaadin.ui.UI and contains a VaadinServlet exactly like the other examples). Listing 5 shows DeclarativeUI.java.
Listing 5. The UI-creation class integrating the custom component

```java
public class DeclarativeUI extends UI {
    @WebServlet(value = "/*", asyncSupported = true)
    @VaadinServletConfiguration(productionMode = false, ui =
            DeclarativeUI.class)
    public static class Servlet extends VaadinServlet {
    }

    @Override
    protected void init(VaadinRequest request) {
        setContent(new ArticleViewDesign());
    }
}
```

Changing the look of an app instantly with Vaadin themes

You'll notice that app's appearance differs between the programmatic version and the declarative version because the two versions use themes. Themes are CSS or Sass code that customizes the look of the app and that can be applied separately from the rest of the application code.

Vaadin comes with four built-in themes you can readily change among. The built-in themes are valo, reindeer, chameleon, and runo. Look in WebContent/themes/declarativeui/declarativeui.scss. Changing the theme value specified toward the end of that file changes the theme the application uses.

You can find an almost unlimited variety of ready-to-apply Vaadin themes in the Vaadin Add-ons Directory.

Programming Vaadin in Python

By design, Vaadin is compatible with programming languages other than Java, supporting many of the programming languages in the JVM ecosystem (see Wikipedia's List of JVM languages) — among them Python (via Jython), Groovy, and Scala.

The final example in this tutorial rewrites the article-viewer app in Python, via Jython.

Before you can successfully load and run this example, you must have the following additional tools installed:

- Jython 2.7 or later.
- PyDev 4.3.0 or later, tested working with Jython (see the PyDev documentation for information on how to install on Eclipse).
- Vaadin 6.8.16 (or the latest 6.x) JAR; you must place this JAR file in the WebContent/WEB-INF/lib folder.

You can find the Eclipse project for this example in the JythonVaadin.zip archive (see Download). Import this archive into Eclipse to run the example.
Notice the similarity of the code structure to the Java version. If you look at the web.xml file of the web application, shown in Listing 6, you can see that it loads the specialized Python interpreter servlet instead of the regular VaadinServlet.

**Listing 6. web.xml descriptor for the JythonVaadin app**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" ...>
  <display-name>JythonVaadin</display-name>
  ...
  <servlet>
    <servlet-name>Jythonvaadin Application</servlet-name>
    <servlet-class>com.example.jythonvaadin.PythonServlet</servlet-class>
    <init-param>
      <description>Vaadin application class to start</description>
      <param-name>application</param-name>
      <param-value>PythonArticleViewer</param-value>
    </init-param>
  </servlet>
</web-app>
```

In the `PythonServlet` class, the Jython interpreter is loaded and supplied with the `PythonArticleViewer.py` source file, as shown in Listing 7.

**Listing 7. PythonServlet instantiating Jython interpreter**

```python
PythonInterpreter interpreter = new PythonInterpreter();
    interpreter.exec("from \"+applicationModuleName+" import \"+applicationClassName+\";
    PyObject pyObj = interpreter.get(applicationClassName).__call__();
    Application pyApp = (Application)pyObj.__tojava__(Application.class);
    applicationClass = pyApp.getClass();
```

Listing 8 shows part of the Python code that creates the article-viewer UI via Vaadin. If you're familiar with Python, you can see the similarity between this code and the Java version.

**Listing 8. Python article-viewer web app**

```python
from com.vaadin import Application
from com.vaadin.event import ItemClickEvent
from java.lang import String
from com.vaadin.terminal import ExternalResource
...
class PythonArticleViewer(Application, ItemClickEvent.ItemClickListener):
    def __init__(self):
        pass
    def init(self):
        ...
    mainWindow = Window("Python Article View")
    vertical = VerticalLayout()
    vertical.setMargin(True)
    horiz = HorizontalLayout()
    horiz.addStyleName("outlined")
    self.table = Table("Select article")
    self.table.addContainerProperty("Year", String().getClass(), None)
    self.table.addContainerProperty("Article", String().getClass(), None)
```
for idx, row in enumerate(self.listdata):
    self.table.addItem(self.getData(row), idx)

...  
vertical.setSizeFull()  
vertical.addComponent(horiz)  
vertical.setExpandRatio(horiz, 1)  
mainWindow.addComponent(vertical)  
self.setMainWindow(mainWindow)

def itemClick(self, event):
    ...

Conclusion

Vaadin boosts Java web developer productivity by offering a full-stack Java development solution. You can create UIs that take advantage of the latest advances in browser innovations, including HTML5, CSS, Ajax, and JavaScript — all without leaving the comfort of your Java IDE. Vaadin UIs can be created programmatically using Java code, or declaratively in an HTML design document. Vaadin even supports code written in other JVM-supported programming languages, including Python. The rich library of available add-ons lets you add features to your app painlessly.

Apps that you create using Vaadin and standard Java EE technologies can be easily deployed in the cloud. Go to my "Develop full-stack Java apps with Vaadin in the cloud" tutorial to quickly deploy a complete web app to the IBM Bluemix cloud.

Acknowledgements

The author would like to thank Marcus Hellberg of Vaadin Inc. for his kind assistance in the review of this tutorial and for supplying the data binding example.
## Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code for first three examples</td>
<td>vaadin_classic_code1_0928.zip</td>
<td>82KB</td>
</tr>
<tr>
<td>Code for Jython example</td>
<td>vaadin_classic_code2.zip</td>
<td>20MB</td>
</tr>
</tbody>
</table>
Related topics

- **developerWorks Premium** provides an all-access pass to powerful tools, curated technical library from Safari Books Online, conference discounts and proceedings, SoftLayer and Bluemix credits, and more.
- Check out these videos to learn more: **Vaadin introduction**
  - **Vaadin Step-by-step Tutorial**: Vaadin address book
- **Vaadin website**: Learn more about Vaadin.
- **Vaadin repository**: Explore the Vaadin source code in the project's GitHub repository.
- **Vaadin sampler**: Try out all of the built-in Vaadin UI components.
- **Vaadin Add-ons Directory**: Browse hundreds of third-party Vaadin components, add-ons, and themes.

© Copyright IBM Corporation 2015
**Trademarks**
(www.ibm.com/developerworks/ibm/trademarks/)