Community Experience Distilled

Learning Vaadin 7

Second Edition

Master the full range of web development features powered by Vaadin-built rich Internet applications

Foreword by Dr. Joonas Lehtinen, CEO and Founder, Vaadin

Nicolas Fränkel

[PACKT] PUBLISHING open source* community experience distilled

Learning Vaadin 7

Second Edition

Master the full range of web development features powered by Vaadin-built rich Internet applications

Nicolas Fränkel

[PACKT] PUBLISHING open source community experience distilled

BIRMINGHAM - MUMBAI

Learning Vaadin 7

Second Edition

First published: October 2011

Second edition: September 2013

Production Reference: 1050913

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78216-977-2

`www.packtpub.com`

Cover Image courtesy of `www.public-domain-image.com`

Credits

Author
Nicolas Fränkel

Reviewers
Martin Cremer
Jonatan Kronqvist
Jouni Lehto
Tomek Lipski
Michael Vogt

Acquisition Editor
Kartikey Pandey

Lead Technical Editor
Madhuja Chaudhari

Technical Editors
Dipika Gaonkar
Dennis John
Mrunmayee Patil
Shali Sasidharan
Sonali Vernekar

Project Coordinator
Wendell Palmer

Proofreader
Samantha Lyon

Indexer
Hemangini Bari

Graphics
Valentina Dsilva
Sheetal Aute
Disha Haria
Abhinash Sahu
Ronak Dhruv

Production Coordinator
Nitesh Thakur

Cover Work
Nitesh Thakur

Foreword

When we started designing Vaadin Framework in the year 2000—then called Millstone Framework—we had a clear vision of creating a platform that would make building web applications fast, easy, and modular; something that we wanted to use by ourselves in the process of building business-oriented web applications. We envisioned a full stack of technologies starting from a web server, an object relationship mapping tool, a rich set of user interface components, and an extensible theme system. Everything built from scratch with a tiny team with no funding and little experience. Fortunately we did not have a clue about the size and complexity of the task or the lack of our experience—otherwise we would have never dared to start working on such a huge task. Finally, it took two years and three complete rewrites to understand the value of focusing solely on the user interface layer and being able to release something solid that has outgrown all the expectations we had.

Now when I look back at the design principles we chose for Vaadin, three principles in particular seem to have contributed to the longevity of the framework. First, we reasoned that the diversity and incompatibility of the web browsers we experienced back in the year 2000 was not going away—quite the contrary. While the Web is today the de facto platform for building all kinds of user interfaces, the capabilities of web browsers seem almost unlimited today and the number of web browsers has grown to include smartphones and tablets in addition to the handful of desktop browsers that should be supported by most applications. So we chose to embrace this complexity and abstract away from the browser to make it easier for developers to support "all" browsers at once. Secondly, we set our optimization target to be developer efficient, which in most cases can be roughly measured by the number of code lines in the user interface layer of the program. This has been a good choice as developers continue to be a more expensive resource in business application projects than servers. Finally, we recognized the need to support the heterogeneous teams where some developers might be more experienced than others. Some of the mechanisms to support the teams include theme packaging, multiple levels of abstraction, support for data bindings side-by-side with internal data in components, and deep inheritance hierarchies for user interface components to name a few.

Vaadin 7 has been the holy grail to our team for as long as I can remember. Somewhere after releasing IT Mill Toolkit 4 in 2007 we started dreaming of all the changes we should be making to the core of the framework to remove some of the shortcomings we felt it had. We started building up a huge backlog of things we should fix sometime soon when we have the time to do so and the courage to tear down parts of the old design. Then came IT Mill Toolkit 5 with the GWT-based client side and later on Vaadin 6. While they both included important features, they skipped many of the hard design choices in our "someday when we have the time" backlog. And we still believed that the time to do these would come really soon after the release of Vaadin 6. What happened was that we underestimated the success of Vaadin 6 and ended up releasing eight minor releases to it during 2009 to 2012. This took all of our focus and we pushed Vaadin 7 even further. Finally near the end of 2011, we decided that we cannot push Vaadin 7 any further and this would be the right time to make all the important changes we have been dreaming of.

While we thought we succeeded in cutting down the number of features in the release to an amount that it would be doable and fully finished in nine months, we ended up using almost twice that time and even then had to leave big things out from the release. But oh boy! We managed to get in over 60 new features, including a huge deal of merging Google Web Toolkit directly inside the Vaadin Framework. With this release, we also shifted the underlying philosophy of the framework by recognizing that all the three layers of abstraction that the framework implements should be equally accessible to the software developers: server-side Java, client-side compiled Java, and client-side JavaScript. This effectively changed Vaadin from being a server-side only framework that abstracts away from the Web to being a full stack framework that bridges Java and HTML5 platforms in a coherent way that supports the developers on all of its levels.

I have always been a huge fan of open source since being introduced to it by starting to play around with Linux kernel 0.3 and early Linux distributions. Working on, living in, and breathing open source did make it natural to choose to release Vaadin with an open source license and to build a community around it. After years of trying and failing to build an impactful community, all pieces finally clicked together in 2009 with the release of Vaadin 6. Seeing how people all over the world started to use Vaadin for building applications their businesses depended on for years to come had been great. What has been even more amazing is how people have started to contribute back to Vaadin—in terms of add-on components, helping each other on the forums, and promoting the framework to their peers. At the end of the day, a lively and friendly community and an ecosystem around Vaadin has been the key to the rapid growth of adoption.

I think that I first heard of Nicolas Fränkel by reading one of his many insightful blog posts some years back. I also remember him being one of the more active Vaadin community members helping others on the forum. At one time Nicolas invited me to a really nice dinner in Geneva where I was visiting the SoftShake conference to discuss about Vaadin and overeat the excellent Swiss fondue. During the dinner, we ended up talking about the need for a book that would tutor beginners through Vaadin and would introduce them to common patterns for Vaadin development. I remember getting contacted by Packt Publishing about getting in touch with potential authors for such a book. Nicolas had quite a lot of Vaadin experience and I asked if he would be interested in considering writing the book. To my surprise he agreed and the first edition of this book was born. Later on when Vaadin 7 was published, Nicolas decided to update the book to cover this newly released Version 7. I am sure that Nicolas underestimated the effort needed in writing about Vaadin 7 the same way as our team did while developing it. Maybe even for the same reason—the list of new things in Vaadin 7 is huge.

You might be familiar with *Book of Vaadin*—a free book about Vaadin. While being a very complete reference to Vaadin and anything related to it, the amount of content and the reference-like approach can make it overwhelming for a beginner. This book takes another approach. Instead of trying to be a reference, it teaches Vaadin concepts by introducing them one-by-one in an order natural for learning. It is written as a journey of building a simple Twitter client while learning the most important aspects of Vaadin—one-by-one.

In conclusion, I'd like to give my deep thanks to Nicolas for taking the challenge of writing this book, which I am sure, will help many people to get a quick start for writing Vaadin-based applications. I hope that these applications will benefit the companies investing in them as well as save a lot of time and frustration from the end users. But at the end of the day, it is most important to me—and I am sure that Nicolas shares this too—that you as a developer of those applications will save your time and frustration and be able to accomplish something that would not be possible otherwise.

Dr. Joonas Lehtinen
CEO and Founder, Vaadin

About the Author

Nicolas Fränkel operates as a successful Java/Java EE architect with more than 10 years' experience in consulting for different clients.

Based in France, he also practices (or has practiced) as a WebSphere Application Server administrator, a certified Valtech trainer, and a part-time lecturer in different French universities, so as to broaden his understanding of software craftsmanship.

His interests in IT are diversified, ranging from Rich Client Application, to Quality Processes through open source software and build automation. When not tinkering with new products or writing blog posts, he may be found practicing sports: squash, kickboxing, and skiing at the moment. Other leisure activities include reading novels, motorcycles, photography, and drawing, not necessarily in that order.

I'd like to thank the Vaadin team for all its glory, the product is awesome, guys! Keep up the good work. I would also like to thank my wonderful wife Corinne for letting me sin once again, this time in full understanding of the time it takes. I love you more after each passing year. My son Dorian, this is only the beginning.

About the Reviewers

Martin Cremer is working as an architect for a company in the financial sector. His work focuses on maintaining and developing reference architecture for web-based enterprise applications with Vaadin as well as supporting developers in their daily work.

Jonatan Kronqvist completed his M.Sc. and has been working with Vaadin Ltd., the company behind the Vaadin framework, since 2006. During this time, he has been a Vaadin consultant, a project manager, and a core developer of the Vaadin framework. Currently he spends his time focusing on add-ons and tools for easing development with Vaadin.

Before going full time on Vaadin, he has worked on many different projects ranging from advanced 3D graphics at a CAD software company to leading the development of a popular computer game for children.

Jouni Lehto has over 10 years' experience on different kind of web technologies and has been involved in a few projects where Vaadin has been the choice.

Tomek Lipski is an open source enthusiast and evangelist. He has over 16 years' commercial experience in IT, and 10 years' experience working in portal, enterprise integration, VAS, and traditional IT areas for the biggest companies in Central Europe.

In 2011, Lipski designed and coordinated an implementation and launch of Aperte workflow—an open source BPMS. Aperte workflow utilizes the OSGi plugin management system to provide flexible solutions combining several popular open source Java-based technologies, such as Vaadin, Liferay, and Activiti.

You can follow `@tomeklipski` on Twitter or check out his blog at `http://blog.tomeklipski.com`.

Michael Vogt started his career in 2000 at Apple, Germany as a WebObjects developer. Since then he has worked in many different companies and countries, mostly as a freelancer on GWT projects. Currently he works in the service department of Vaadin.

www.PacktPub.com

Support files, eBooks, discount offers, and more

You might want to visit `www.PacktPub.com` for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

`http://PacktLib.PacktPub.com`

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at `www.PacktPub.com`, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface

Vaadin is a component-based Java web framework for making applications look great and perform well, making your users happy. Vaadin promises to make your user interfaces attractive and usable while easing your development efforts and boosting your productivity. After having read this book, you will be able to utilize the full range of development and deployment features offered by Vaadin while thoroughly understanding the concepts.

Learning Vaadin 7 Second Edition is a practical systematic tutorial to understand, use, and master the art of RIA development with Vaadin. You will learn about the fundamental concepts that are the cornerstones of the framework, at the same time making progress on building your own web application. The book will also show you how to integrate Vaadin with other popular frameworks and how to run it on top of internal, as well as externalized infrastructures.

This book will show you how to become a professional Vaadin developer by giving you a concrete foundation through diagrams, practical examples, and ready-to-use source code. It will enable you to grasp all the notions behind Vaadin one-step at a time: components, layouts, events, containers, and bindings. You will learn to build first-class web applications using best-of-breed technologies. You will find detailed information on how to integrate Vaadin's presentation layer on top of other widespread technologies, such as CDI and JPA. Finally, the book will show you how to deploy on different infrastructures, such as GateIn portlet container and Cloud platform Jelastic.

This book is an authoritative and complete systematic tutorial on how to create top-notch web applications with the RIA Vaadin framework.

What this book covers

Chapter 1, *Vaadin and its Context*, introduces Vaadin, its features, its philosophy, and its surrounding environment.

Chapter 2, *Environment Setup*, describes how to set up the development environment, whether using Eclipse or IntelliJ IDEA.

Chapter 3, *Hello Vaadin!*, creates a basic Vaadin project and explains what happens under the hood.

Chapter 4, *Components and Layouts*, presents simple building blocks for any Vaadin application worth its salt.

Chapter 5, *Event Listener Model*, illustrates the interactions between users and your application and how they are implemented in Vaadin.

Chapter 6, *Containers and Related Components*, explains not only components presenting beans collections, but also how they can be bound to the underlying data.

Chapter 7, *Core Advanced Features*, portrays advanced use-cases addressing real-life problems, such as using the Navigation API, running Vaadin applications inside legacy ones, error handling customization, data source binding, and the ever-popular server push.

Chapter 8, *Featured Add-ons*, lists some extra-features such as GUI declarative description through XML, client widgets integration, JPA and CDI integration. These are available with additional libraries known as add-ons.

Chapter 9, *Creating and Extending Components and Widgets*, details how to create new and extend existing server-side components and client-side widgets.

Chapter 10, *Enterprise Integration*, describes how to deploy Vaadin applications in other contexts commonly found in enterprise environment: portals such as JBoss GateIn, OSGi platforms using Glassfish and finally "the cloud" with provider Jelastic.

What you need for this book

In order to get the most out of this book, it's advised to have a computer, a Java Developer Kit 6/7 installed as well as an Internet access.

Helpful tools include a Java IDE, Eclipse or IntelliJ IDEA, the Tomcat servlet container, and the Glassfish Application Server.

Who this book is for

If you are a Java developer with some experience in Java web development and want to enter the world of Rich Internet Applications, then this technology and book are ideal for you. *Learning Vaadin 7 Second Edition* will be perfect as your next step towards building eye-candy dynamic web applications on the JVM.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: "Every component is declared as an attribute and assigned a `@AutoGenerated` annotation."

A block of code is set as follows:

```
public class DisableOnClickButtonExtension extends AbstractExtension {

  public DisableOnClickButtonExtension(String disabledLabel) {

    getState().setDisabledLabel(disabledLabel);
  }

  ...
}
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
@Connect(DisableOnClickButtonExtension.class)
public class DisableOnClickButtonConnector extends
AbstractExtensionConnector {

  @Override
public DisableOnClickButtonSharedState getState() {


 return (DisableOnClickButtonSharedState) super.getState();

  }

  ...
}
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Now, just go the **File** menu and click on **New**".

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

1
Vaadin and its Context

Developing Java applications and more specifically, developing Java web applications should be fun. Instead, most projects are a mess of sweat and toil, pressure and delays, costs and cost cutting. Web development has lost its appeal. Yet, among the many frameworks available, there is one in particular that draws our attention because of its ease of use and its original stance. It has been around since the past decade and has begun to grow in importance. The name of this framework is **Vaadin**. The goal of this book is to see, step-by-step, how to develop web applications with Vaadin.

Vaadin is the Finnish word for a female reindeer (as well as a Finnish goddess). This piece of information will do marvels to your social life as you are now one of the few people on Earth who know this (outside Finland).

We are going to see Vaadin in detail in later chapters; the following is a preview of what it is:

- A component-based approach that really works, and provides a bunch of out-of-the-box components as well as extensions
- Full web compatibility, in addition to Google Web Toolkit
- All development is made completely in Java
- Most of the code can be run server-side, taking advantages of Java static typing, with the full power of dynamic update tools such as JRebel
- Integration with Eclipse and IntelliJ IDEA IDEs
- Available with no charge under a friendly Open Source Apache license and much, much more

Before diving right into Vaadin, it is important to understand what led to its creation. Readers who already have this information (or who don't care) should go directly to *Chapter 2*, *Environment Setup*.

In this chapter, we will look into the following:

- The evolution from mainframe toward the rich client.
    - The concept of application tier
    - The many limits of the thin-client approach
    - What stands beyond those limits
- Why choose Vaadin today?
    - The state of the market
    - Vaadin's place in the market
    - A preview of what other frameworks Vaadin can be integrated with and what platforms it can run on

Rich applications

Vaadin is often referred to as a **Rich Internet Application** (**RIA**) framework. Before explaining why, we need to first define some terms which will help us describe the framework. In particular, we will have a look at application tiers, the different kind of clients, and their history.

Application tiers

Some software run locally, that is, on the client machine and some run remotely, such as on a server machine. Some applications also run on both the client and the server. For example, when requesting an article from a website, we interact with a browser on the client side but the order itself is passed on a server in the form of a request.

Traditionally, all applications can be logically separated into tiers, each having different responsibilities as follows:

- **Presentation**: The presentation tier is responsible for displaying the end-user information and interaction. It is the realm of the user interface.
- **Business Logic**: The logic tier is responsible for controlling the application logic and functionality. It is also known as the application tier, or the middle tier as it is the glue between the other two surrounding tiers, thus leading to the term middleware.

- **Data**: The data tier is responsible for storing and retrieving data. This backend may be a file system. In most cases, it is a database, whether relational, flat, or even an object-oriented one.

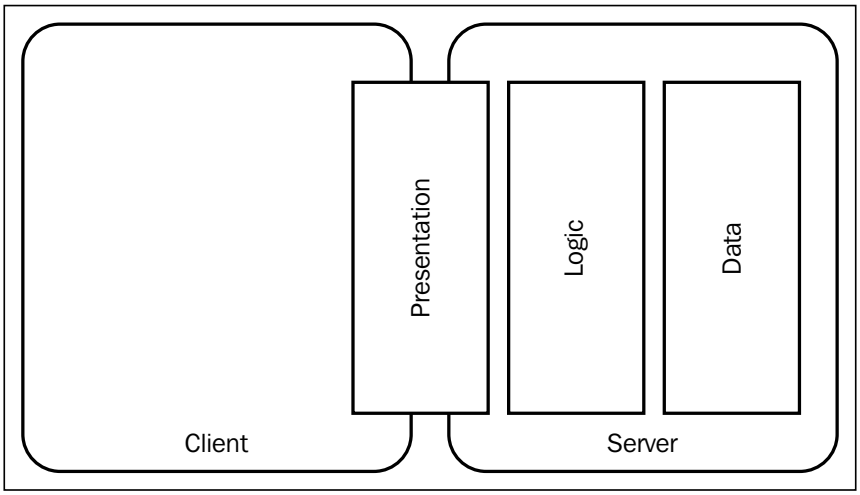
This categorization not only naturally corresponds to specialized features, but also allows you to physically separate your system into different parts, so that you can change a tier with reduced impact on adjacent tiers and no impact on non-adjacent tiers.

Tier migration

In the history of computers and computer software, these three tiers have moved back and forth between the server and the client.

Mainframes

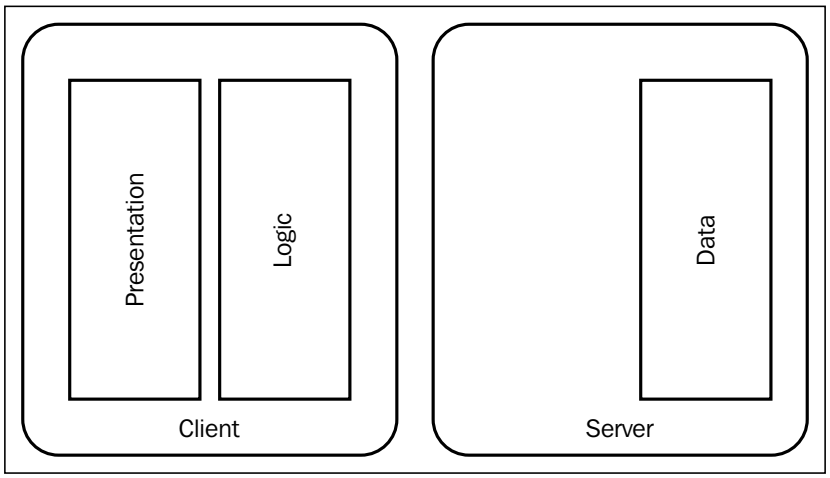
When computers were mainframes, all tiers were handled by the server. Mainframes stored data, processed it, and were also responsible for the layout of the presentation. Clients were dumb terminals, suited only for displaying characters on the screen and accepting the user input.



Client server

Not many companies could afford the acquisition of a mainframe (and many still cannot). Yet, those same companies could not do without computers at all, because the growing complexity of business processes needed automation. This development in personal computers led to a decrease in their cost. With the need to share data between them, the network traffic rose.

This period in history saw the rise of the personal computer, as well as the **Client server** term, as there was now a true client. The presentation and logic tier moved locally, while shared databases were remotely accessible, as shown in the following diagram:



Thin clients

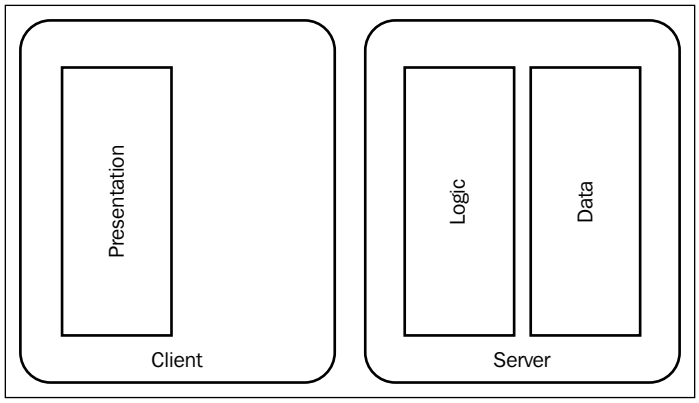
Big companies migrating from mainframes to client-server architectures thought that deploying software on ten client machines on the same site was relatively easy and could be done in a few hours. However, they quickly became aware of the fact that with the number of machines growing in a multi-site business, it could quickly become a nightmare.

Enterprises also found that it was not only the development phase that had to be managed like a project, but also the installation phase. When upgrading either the client or the server, you most likely found that the installation time was high, which in turn led to downtime and that led to additional business costs.

Around 1991, *Sir Tim Berners-Lee* invented the **Hyper Text Markup Language**, better known as **HTML**. Some time after that, people changed its original use, which was to navigate between documents, to make HTML-based web applications. This solved the deployment problem as the logic tier was run on a single-server node (or a cluster), and each client connected to this server. A deployment could be done in a matter of minutes, at worst overnight, which was a huge improvement. The presentation layer was still hosted on the client, with the browser responsible for displaying the user interface and handling user interaction.

This new approach brought new terms, which are as follows:

- The old client-server architecture was now referred to as **fat client**.
- The new architecture was coined as **thin client**, as shown in the following diagram:



Limitations of the thin-client applications approach

Unfortunately, this evolution was made for financial reasons and did not take into account some very important drawbacks of the thin client.

Poor choice of controls

HTML does not support many controls, and what is available is not on par with fat-client technologies. Consider, for example, the list box: in any fat client, choices displayed to the user can be filtered according to what is typed in the control. In legacy HTML, there's no such feature and all lines are displayed in all cases. Even with HTML5, which is supposed to add this feature, it is sadly not implemented in all browsers. This is a usability disaster if you need to display the list of countries (more than 200 entries!). As such, ergonomics of true thin clients have nothing to do with their fat-client ancestors.

Many unrelated technologies

Developers of fat-client applications have to learn only two languages: SQL and the technology's language, such as Visual Basic, Java, and so on.

Web developers, on the contrary, have to learn an entire stack of technologies, both on the client side and on the server side.

On the client side, the following are the requirements:

- First, of course, is HTML. It is the basis of all web applications, and although some do not consider it a programming language *per se*, every web developer must learn it so that they can create content to be displayed by browsers.
- In order to apply some common styling to your application, one will probably have to learn the **Cascading Style Sheets** (**CSS**) technology. CSS is available in three main versions, each version being more or less supported by browser version combinations (see *Browser compatibility*).
- Most of the time, it is nice to have some interactivity on the client side, like pop-up windows or others. In this case, we will need a scripting technology such as **ECMAScript**.

ECMAScript is the specification of which JavaScript is an implementation (along with **ActionScript**). It is standardized by the ECMA organization. See `http://www.ecma-international.org/publications/standards/Ecma-262.htm` for more information on the subject.

- Finally, one will probably need to update the structure of the HTML page, a healthy dose of knowledge of the **Document Object Model** (**DOM**) is necessary.

As a side note, consider that HTML, CSS, and DOM are W3C specifications while ECMAScript is an ECMA standard.

From a Java point-of-view and on the server side, the following are the requirements:

- As servlets are the most common form of request-response user interactions in Java EE, every web developer worth his salt has to know both the Servlet specification and the Servlet API.

- Moreover, most web applications tend to enforce the **Model-View-Controller** paradigm. As such, the Java EE specification enforces the use of servlets for controllers and **JavaServer Pages** (**JSP**) for views. As JSP are intended to be templates, developers who create JSP have an additional syntax to learn, even though they offer the same features as servlets.
- JSP accept scriptlets, that is, Java code snippets, but good coding practices tend to frown upon this, however, as Java code can contain any feature, including some that should not be part of views—for example, the database access code. Therefore, a completely new technology stack is proposed in order to limit code included in JSP: the tag libraries. These tag libraries also have a specification and API, and that is another stack to learn.

However, these are a few of the standard requirements that you should know in order to develop web applications in Java. Most of the time, in order to boost developer productivity, one has to use frameworks. These frameworks are available in most of the previously cited technologies. Some of them are supported by Oracle, such as Java Server Faces, others are open source, such as **Struts**.

JavaEE 6 seems to favor replacement of JSP and Servlet by **Java Server Faces** (**JSF**). Although JSF aims to provide a component-based MVC framework, it is plagued by a relative complexity regarding its components lifecycle.

Having to know so much has negative effects, a few are as follows:

- On the technical side, as web developers have to manage so many different technologies, web development is more complex than fat-client development, potentially leading to more bugs
- On the human resources side, different meant either different profiles were required or more resources, either way it added to the complexity of human resource management
- On the project management side, increased complexity caused lengthier projects: developing a web application was potentially taking longer than developing a fat-client application

All of these factors tend to make the thin-client development cost much more than fat-client, albeit the deployment cost was close to zero.

Browser compatibility

The Web has standards, most of them upheld by the World Wide Web Consortium. Browsers more or less implement these standards, depending on the vendor and the version. The ACID test, in version 3, is a test for browser compatibility with web standards. Fortunately, most browsers pass the test with 100 percent success, which was not the case two years ago.

Some browsers even make the standards evolve, such as Microsoft which implemented the `XmlHttpRequest` object in Internet Explorer and thus formed the basis for Ajax.

One should be aware of the combination of the platform, browser, and version. As some browsers cannot be installed with different versions on the same platform, testing can quickly become a mess (which can fortunately be mitigated with virtual machines and custom tools like `http://browsershots.org`). Applications should be developed with browser combinations in mind, and then tested on it, in order to ensure application compatibility.

For intranet applications, the number of supported browsers is normally limited. For Internet applications, however, most common combinations must be supported in order to increase availability. If this wasn't enough, then the same browser in the same version may run differently on different operating systems.

In all cases, each combination has an exponential impact on the application's complexity, and therefore, on cost.

Page flow paradigm

Fat-client applications manage windows. Most of the time, there's a main window. Actions are mainly performed in this main window, even if sometimes managed windows or pop-up windows are used.

As web applications are browser-based and use HTML over HTTP, things are managed differently. In this case, the presentation unit is not the window but the page. This is a big difference that entails a performance problem: indeed, each time the user clicks on a submit button, the request is sent to the server, processed by it, and the HTML response is sent back to the client.

For example, when a client submits a complex registration form, the entire page is recreated on the server side and sent back to the browser even if there is a minor validation error, even though the required changes to the registration form would have been minimal.

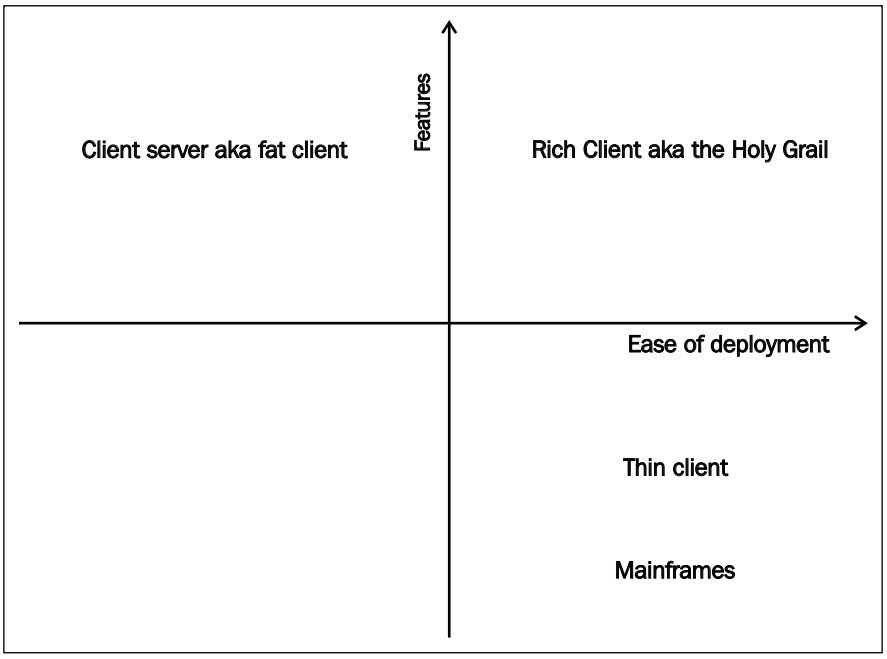
Beyond the limits

Over the last few years, users have been applying some pressure in order to have user interfaces that offer the same richness as good old fat-client applications. IT managers, however, are unwilling to go back to the old deploy-as-a-project routine and its associated costs and complexity. They push towards the same deployment process as thin-client applications. It is no surprise that there are different solutions in order to solve this dilemma.

What are rich clients?

All the following solutions are globally called rich clients, even if the approach differs. They have something in common though: all of them want to retain the ease of deployment of the thin client and solve some or all of the problems mentioned previously.

Rich clients fulfill the fourth quadrant of the following schema, which is like a dream come true, as shown in the following diagram:



Some rich client approaches

The following solutions are strategies that deserve the rich client label.

Ajax

Ajax was one of the first successful rich-client solutions. The term means **Asynchronous JavaScript with XML**. In effect, this browser technology enables sending asynchronous requests, meaning there is no need to reload the full page. Developers can provide client scripts implementing custom callbacks: those are executed when a response is sent from the server. Most of the time, such scripts use data provided in the response payload to dynamically update relevant part of the page DOM.

Ajax addresses the richness of controls and the page flow paradigm. Unfortunately:

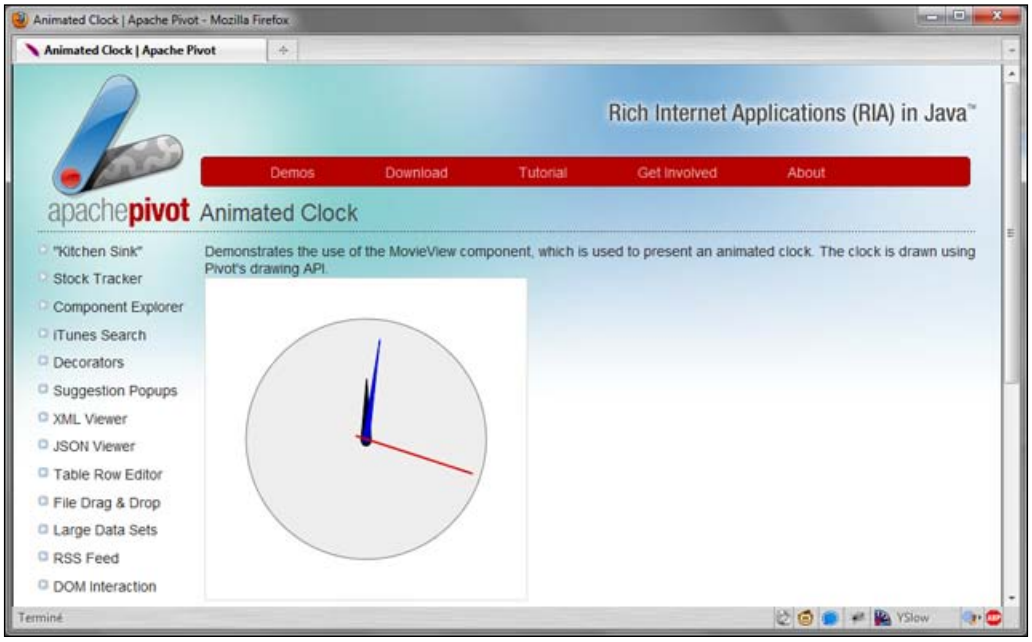
- It aggravates browser-compatibility problems as Ajax is not handled in the same way by all browsers.
- It has problems unrelated directly to the technologies, which are as follows:
    - Either one learns all the necessary technologies to do Ajax on its own, that is, JavaScript, Document Object Model, and JSON/XML, to communicate with the server and write all common features such as error handling from scratch.
    - Alternatively, one uses an Ajax framework, and thus, one has to learn another technology stack.

Richness through a plugin

The oldest way to bring richness to the user's experience is to execute the code on the client side and more specifically, as a plugin in the browser. Sun—now Oracle—proposed the applet technology, whereas Microsoft proposed **ActiveX**. The latest technology using this strategy is **Flash**.

All three were failures due to technical problems, including performance lags, security holes, and plain-client incompatibility or just plain rejection by the market.

There is an interesting way to revive the applet with the Apache Pivot project, as shown in the following screenshot (`http://pivot.apache.org/`), but it hasn't made a huge impact yet;



A more recent and successful attempt at executing code on the client side through a plugin is through Adobe's Flex. A similar path was taken by Microsoft's Silverlight technology.

Flex is a technology where static views are described in XML and dynamic behavior in ActionScript. Both are transformed at compile time in Flash format.

Unfortunately, Apple refused to have anything to do with the Flash plugin on iOS platforms. This move, coupled with the growing rise of HTML5, resulted in Adobe donating Flex to the Apache foundation. Also, Microsoft officially renounced plugin technology and shifted Silverlight development to HTML5.

Deploying and updating fat-client from the web

The most direct way toward rich-client applications is to deploy (and update) a fat-client application from the web.

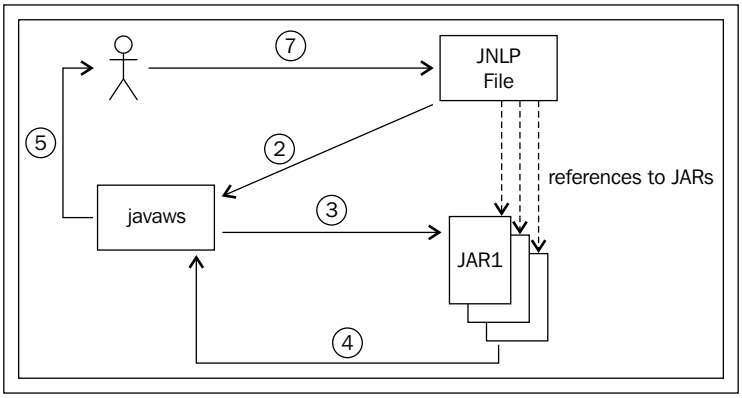
Java Web Start

Java Web Start (**JWS**), available at `http://download.oracle.com/javase/1.5.0/docs/guide/javaws/`, is a proprietary technology invented by Sun. It uses a deployment descriptor in **Java Network Launching Protocol** (**JNLP**) that takes the place of the manifest inside a JAR file and supplements it. For example, it describes the main class to launch the classpath, and also additional information such as the minimum Java version, icons to display on the user desktop, and so on.

This descriptor file is used by the `javaws` executable, which is bundled in the Java Runtime Environment. It is the `javaws` executable's responsibility to read the JNLP file and do the right thing according to it. In particular, when launched, `javaws` will download the updated JAR.

The detailed process goes something like the following:

1. The user clicks on a JNLP file.
2. The JNLP file is downloaded on the user machine, and interpreted by the local `javaws` application.
3. The file references JARs that `javaws` can download.
4. Once downloaded, JWS reassembles the different parts, create the classpath, and launch the main class described in the JNLP.



JWS correctly tackles all problems posed by the thin-client approach. Yet it never reaches critical mass for a number of reasons:

1. First time installations are time-consuming because typically lots of megabytes need to be transferred over the wire before the users can even start using the app. This is a mere annoyance for intranet applications, but a complete no go for Internet apps.
2. Some persistent bugs weren't fixed across major versions.
3. Finally, the lack of commercial commitment by Sun was the last straw.

A good example of a successful JWS application is **JDiskReport** (`http://www.jgoodies.com/download/jdiskreport/jdiskreport.jnlp`), a disk space analysis tool by *Karsten Lentzsch*, which is available on the Web for free.

Update sites

Updating software through update sites is a path taken by both **Integrated Development Environment** (**IDE**) leaders, NetBeans and Eclipse. In short, once the software is initially installed, updates and new features can be downloaded from the application itself.

Both IDEs also propose an API to build applications.

This approach also handles all problems posed by the thin-client approach. However, like JWS, there's no strong trend to build applications based on these IDEs. This can probably be attributed to both IDEs using the **OSGI** standard whose goal is to address some of Java's shortcomings but at the price of complexity.

Google Web Toolkit

Google Web Toolkit (**GWT**) is the framework used by Google to create some of its own applications. Its point of view is very unique among the technologies presented here. It lets you develop in Java, and then the GWT compiler transforms your code to JavaScript, which in turn manipulates the DOM tree to update HTML. It's GWT's responsibility to handle browser compatibility. This approach also solves the other problems of the pure thin-client approach.

Yet, GWT does not shield developers from all the dirty details. In particular, the developer still has to write part of the code handling server-client communication and he has to take care of the segregation between Java server-code which will be compiled into byte code and Java client-code which will be compiled into JavaScript. Also, note that the compilation process may be slow, even though there are a number of optimization features available during development. Finally, developers need a good understanding of the DOM, as well as the JavaScript/DOM event model.

Why Vaadin?

Vaadin is a solution evolved from a decade of problem-solving approach, provided by a Finnish company named Vaadin Ltd, formerly IT Mill.

Therefore, having so many solutions available, could question the use of Vaadin instead of Flex or GWT? Let's first have a look at the state of the market for web application frameworks in Java, then detail what makes Vaadin so unique in this market.

State of the market

Despite all the cons of the thin-client approach, an important share of applications developed today uses this paradigm, most of the time with a touch of Ajax augmentation.

Unfortunately, there is no clear leader for web applications. Some reasons include the following:

- Most developers know how to develop plain old web applications, with enough Ajax added in order to make them usable by users.
- GWT, although new and original, is still complex and needs seasoned developers in order to be effective.

From a Technical Lead or an IT Manager's point of view, this is a very fragmented market where it is hard to choose a solution that will meet users' requirements, as well as offering guarantees to be maintained in the years to come.

Importance of Vaadin

Vaadin is a unique framework in the current ecosystem; its differentiating features include the following:

- There is no need to learn different technology stacks, as the coding is solely in Java. The only thing to know beside Java is Vaadin's own API, which is easy to learn. This means:
    - The UI code is fully object-oriented
    - There's no spaghetti JavaScript to maintain
    - It is executed on the server side

- Furthermore, the IDE's full power is in our hands with refactoring and code completion.
- No plugin to install on the client's browser, ensuring all users that browse our application will be able to use it *as-is*.
- As Vaadin uses GWT under the hood, it supports all browsers that the version of GWT also supports. Therefore, we can develop a Vaadin application without paying attention to the browsers and let GWT handle the differences. Our users will interact with our application in the same way, whether they use an outdated version (such as Firefox 3.5), or a niche browser (like Opera).
- Moreover, Vaadin uses an abstraction over GWT so that the API is easier to use for developers. Also, note that Vaadin Ltd (the company) is part of GWT steering committee, which is a good sign for the future.
- Finally, Vaadin conforms to standards such as HTML and CSS, making the technology future proof. For example, many applications created with Vaadin run seamlessly on mobile devices although they were not initially designed to do so.

Vaadin integration

In today's environment, integration features of a framework are very important, as normally every enterprise has rules about which framework is to be used in some context. Vaadin is about the presentation layer and runs on any servlet container capable environment.

Integrated frameworks

A whole chapter (see *Chapter 9*, *Creating and Extending Components and Widgets*) is dedicated to the details of how Vaadin can be integrated with some third-party frameworks and tools. There are three integration levels possible which are as follows:

- **Level 1**: out-of-the-box or available through an add-on, no effort required save reading the documentation
- **Level 2**: more or less documented
- **Level 3**: possible with effort