O'REILLY®

Client-Server Web Apps with JavaScript and Java

RICH, SCALABLE, AND RESTFUL

Casimir Saternos

O'REILLY®

Client-Server Web Apps with JavaScript and Java

As a Java programmer, how can you tackle the disruptive client-server approach to web development? With this comprehensive guide, you'll learn how today's client-side technologies and web APIs work with various Java tools. Author Casimir Saternos provides the big picture of client-server development, and then takes you through many practical client-server architectures. You'll work with hands-on projects in several chapters to get a feel for the topics discussed.

User habits, technologies, and development methods have drastically altered web app design in recent years. But the Web itself hasn't changed. This book shows you how to build apps that conform to the Web's underlying architecture.

- Learn the advantages of using separate client and server tiers, including code organization and speedy prototyping
- Explore the major tools, frameworks, and starter projects used in JavaScript development
- Dive into web API design and the REST style of software architecture
- Understand Java's alternatives to traditional packaging methods and application server deployment
- Build projects with lightweight servers, using jQuery with Jython, and Sinatra with Angular
- Create client-server web apps with traditional Java web application servers and libraries

Casimir Saternos is a Software Architect at Synchronoss Technologies, Inc. His articles have appeared in *Java Magazine* and on the Oracle Technology Network, and his screencasts are available through Pluralsight.

"Given the migration of the client-server paradigm to the browser, new technologies and architectural challenges face today's programmers. Cas' book gets right to the heart of those complexities, giving the reader an immediate view of the state of web application development."

—**Tony Powell**
Director, Solutions Delivery at Trifecta Technologies

JAVASCRIPT

US $29.99 CAN $31.99
ISBN: 978-1-449-36933-0

Twitter: @oreillymedia
facebook.com/oreilly

Client-Server Web Apps with JavaScript and Java

Casimir Saternos

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo O'REILLY®

Client-Server Web Apps with JavaScript and Java

by Casimir Saternos

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editors: Simon St. Laurent and Allyson MacDonald
Production Editor: Kristen Brown
Copyeditor: Gillian McGarvey
Proofreader: Amanda Kersey
Indexer: Judith McConville
Cover Designer: Karen Montgomery
Interior Designer: David Futato
Illustrator: Rebecca Demarest

April 2014: First Edition

Revision History for the First Edition:

2014-03-27: First release

See *http://oreilly.com/catalog/errata.csp?isbn=9781449369330* for release details.

ISBN: 978-1-449-36933-0

[LSI]

Table of Contents

Preface

There are only two hard things in Computer Science: cache invalidation and naming things.

—Phil Karlton

While cache invalidation is not a difficulty encountered when writing a book, choosing a suitable title is. The title of this book is intended to represent a broad area of changes in web development that have resulted in a new approach to designing web applications.

Of course, many aspects of web development can be considered new. Developers scramble to keep up with enhancements to desktop browsers, new mobile device clients, evolving programming languages, the availability of faster processors, and an increasingly discerning audience of users with growing expectations about usability and interactivity. These changes require developers to continually innovate when coming up with solutions for their specific projects. But many of these solutions have broader implications and are not isolated to any particular project.

Therefore, I chose "client-server" as the term which in many ways captures the changes to web development that have occurred in response to these innovations. Other descriptions of modern development practices currently in vogue don't adequately represent the problem domain. Web application development is associated with desktop browsers, but excludes the increasingly relevant area of mobile applications.

The terms Single Page Application and Single Page Interface have been used to distinguish modern web applications from earlier static websites. These terms correctly identify modern sites as far more dynamic and interactive than their predecessors.

However, many modern dynamic applications are made up of multiple pages rather than a single page. The focus in these terms is on the page, the client portion of an application. They make no specific statement about corresponding server-side development. There are JavaScript frameworks that are also associated with highly dynamic pages (such as Angular, Ember, and Backbone), but these are also concerned with the

client tier. I wanted the title of this book to encompass more than front-end innovations and to recognize the corresponding server-side design and web service messaging.

The method of communication captured by the popular acronym *REST (Representational State Transfer)* does suggest the web service messaging style. But the definition of REST as specified by its author Roy Fielding is very limiting. On his blog, Fielding lists specific restrictions to REST that are commonly violated in so-called RESTful APIs. And some even question whether a JSON API can be truly RESTful due to the fact that it does not satisfy all of the constraints associated with the style of architecture. There is a continuum by which REST services can be described; so that an API can be described as RESTful only to the degree that it adheres to the constraints. REST does include client-server as one of its constraints, and the verb and URL naming conventions are certainly applicable.

So a JavaScript client consuming messages from a pragmatic "RESTful" API is a significant part of the method of development. What about the server component?

Java Enterprise Edition (JEE) includes the JAX-RS API, which uses Java's flavor of REST (which is not inherently strict) and is demonstrable using the Jersey reference implementation. But limiting to *JAX-RS web application development* ignores frameworks and alternate JVM language solutions that are available and particularly appealing for quick prototypes.

And so crystallizing the intentions of a book in a simple, catchy title is not an easy task. Fortunately, James Ward did a presentation at OSCON 2012 in which he described the development of "Client-Server Web Applications with HTML5 and Java." He listed the benefits of a method of web application development that is increasingly popular, a method that I have been involved with in recent years on various projects. And the phrase "client-server" is the key to understanding what this method is. It captures the fundamental architectural changes that include aspects of the terms listed above, but represents the distinct partitioning between the client and server and considers each of the roles significant.

A client-server architecture of web applications requires a shift (in some cases seismic) in the way programmers work. This book was written to enable developers to deal with this revolution. Specifically, it is intended to provide a proper perspective in building the latest incarnation of modern web applications.

Who Is This Book For?

This book is written for web application developers who are are familiar with the Java programming language, as well as HTML, JavaScript, and CSS. It is geared toward those who "learn by doing" and prefer to see and create specific examples of new technologies and techniques integrated with standard tools. If you want a better understanding of

recent developments in JavaScript and how the language and its development process compare with those of Java, this book is for you.

A bit of a balancing act is evident as you read this book. On the one hand, the most important thing you can take away is a sense of the "big picture"—the influences and trends causing a shift in the technologies in use. On the other hand, technologies are often best understood by seeing specific examples. If you are interested in an overview of how these technologies actually fit together, you will benefit from this book.

My goal in writing this is to help you to make informed decisions. Good decisions result in the right technologies being used on new projects. They allow you to avoid pitfalls caused by mixing incompatible technologies or having the wrong expectations about the implications of a given decision. They help you to step into projects in process and better support existing code. In short, informed decisions will make you a more productive programmer. They help you make effective use of your time in researching areas of specific interest in your work now and in the future.

How This Book Is Organized

Chapter 1 provides a general overview of the client-server web application architecture. It discusses the history of web development and provides a justification for the paradigm shift in development. This leads into the next three chapters that will describe the tools used in the development process.

Chapter 2 describes JavaScript and the tools used in JavaScript development.

Chapter 3 introduces web API design, REST, and the tools used when developing RESTful applications over HTTP.

Chapter 4 pertains to Java and other software that's used in the remainder of this book.

The next section of the book discusses higher-level constructs (such as client libraries and application servers) and how these provide separation and allow for rapid development.

Chapter 5 describes major client-side JavaScript frameworks.

Chapter 6 addresses Java API servers and services.

Chapter 7 discusses rapid development practices.

Chapter 8 delves into API design in greater depth.

With an understanding of libraries and a process for speedy development of prototypes, the next several chapters apply these to specific projects using various JVM languages and frameworks. The next two chapters use lightweight web servers and microframeworks instead of traditional Java web application packaging and servers.

Chapter 9 provides an overview of a project using jQuery and Jython.

Chapter 10 documents the development of a project using JRuby and Angular.

The final chapters detail projects using traditional Java web application servers and libraries.

Chapter 11 looks at the range of packaging and deployment options available in the Java ecosystem.

Chapter 12 explores virtualization and innovations emerging from the management of large server environments.

Chapter 13 draws attention to testing and documentation.

Chapter 14 wraps up with some final thoughts on responding to the tumultuous changes to Internet-related technologies and software development.

Appendix A describes how to explore and manipulate Java classes interactively.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to variables, method names, and other code elements, as well as the contents of files.

`Constant width bold`
: Highlights new code in an example.

`Constant width italic`
: Shows text that should be replaced with user-supplied values.

This element signifies a tip, suggestion, or general note.

This element indicates a warning or caution.

Code Examples

Projects and code examples in this book are hosted on *https://github.com/java-javascript/client-server-web-apps*. You can view them online or download a *.zip* file for local use. The assets are organized by chapter.

The code examples provided in this book are geared toward illustrating specific functionality rather than addressing all concerns of a fully functional application. Differences include:

- Production systems include greater refinement of selected data types, validation rules, exception handing routines, and logging mechanisms.
- Most production systems will include one or more backend datastores. To limit the scope of discussion, databases are not accessed in most of the examples.
- The modern web application includes a large amount of infrastructure geared toward mobile device access and browser compatibility. Again, unless these are the specific topic of discussion, responsive design is eschewed for a more minimal design.
- The practice of some degree of unobtrusive JavaScript to separate CSS and JavaScript from HTML is a generally accepted best practice. In the examples in this book, they are frequently commingled because all aspects of a given application can be immediately apprised by viewing a single file.
- Unit tests and testing examples are only included when they are directly related to the topic under discussion. Production systems would include far greater test coverage and extensive testing in general.

That said, this book is intended to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you are reproducing a significant portion of the code. For example, writing a program that uses several sections of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Client-Server Web Apps with JavaScript and Java*" by Casimir Saternos (O'Reilly). Copyright 2014 EzGraphs, LLC., 978-1-449-36933-0."

If you feel your use of code examples falls outside fair use or the permission given here, feel free to contact us at *permissions@oreilly.com*.

Long Command Formats

Code displayed inline will be adjusted to be readable in this context. One convention used is that of backslashes to allow newlines in operating system commands. So for instance, the following commands are equivalent and would execute the same way in a bash session. (Bash is a standard operating system shell that you see when accessing a Linux server or Mac OS X at the command line.)

```
ls -l *someVeryLongName*
...
ls -l \
*someVeryLongName*
```

The same convention also appears in other settings where OS commands are used, such as Dockerfiles.

Similarly, JSON strings, being valid JavaScript, can be broken up to fit on multiple lines:

```
o={"name": "really long string here and includes many words"}

// The following, as expected, evaluates to true.
JSON.stringify(o)=='{"name":"really long string here and includes many words"}'

// The same string broken into multiple lines is equivalent.
// So the following statement also evaluates to true.
JSON.stringify(o)=='{"name":' +
                   '"some really long ' +
                   'JSON string is here' +
                   ' and includes many, many words"}'
```

Safari® Books Online

Safari Books Online is an on-demand digital library that delivers expert content in both book and video form from the world's leading authors in technology and business.

Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of product mixes and pricing programs for organizations, government agencies, and individuals. Subscribers have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and dozens more. For more information about Safari Books Online, please visit us online.

How to Contact Us

Every example in this book has been tested, but occasionally you may encounter problems. Mistakes and oversights can occur and we will gratefully receive details of any that you find, as well as any suggestions you would like to make for future editions. Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *http://oreil.ly/client-server-web-apps-js*.

To comment or ask technical questions about this book, send email to *bookquestions@oreilly.com*.

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgments

Thank you to the following people:

- Meg, Ally, Simon, and the gang at O'Reilly for the opportunity to write this book.
- My brother Neal Saternos and Dr. James Femister for the early suggestions from days gone by that I might be able to do the "programming thing."
- Michael Bellomo, Don Deasey, and Scott Miller for their time and expertise as technical reviewers.
- Charles Leo Saternos for taking a break from Lua game development to do some fine image and design work.
- Caleb Lewis Saternos for inspiration in perserverence (early morning run anyone?) and editorial work.
- David Amidon for the first opportunity to work as a software developer and Doug Pelletier for first the opportunity to develop Java web apps.

- All the folks that headed up the projects that inspired this book, including managers Wayne Hefner, Tony Powell, Dave Berry, Jay Colson, and Pat Doran, and chief software architects Eric Fedok and Michael Bellomo.
- Geoffrey Grosenbach from PluralSight, Nat Dunn from Webucator, Caroline Kvitka (and others from Oracle and *Java Magazine*) for technical writing opportunities over the past several years that led to the current one.
- My parents Leo and Clara Saternos for bringing me up in a loving household that included a Radio Shack Color Computer when having a PC at home was still a novelty and my sister Lori for reminders of important things that have nothing to do with programming.

My love and thanks to my wonderful wife Christina and children Clara Jean, Charles Leo, Caleb Lewis, and Charlotte Olivia for the consistent love, support, patience, and inspiration while this project was underway.

Finally, J.S. Bach serves as a creative inspiration on many levels. Not the least of which is the dedication that would appear at the beginning of his works—and so I say with him, Soli Deo Gloria.

CHAPTER 1
Change Begets Change

The entrepreneur always searches for a change, responds to it, and exploits it as an opportunity.

—Peter Drucker

What kinds of changes encourage developers to adopt a client-server approach? Shifts in user behavior, technology, and software development process are the significant forces that have driven developers to change their patterns of design. Each of these factors, in a unique and significant way, makes established patterns obsolete. Together they have encouraged related innovations and a convergence in practice despite the absence of enforcement or mandated standardization.

Web users have changed. In the early days of the Web, users were satisfied with static pages and primitive user interfaces. The modern web user has come to expect a high-performance, interactive, well-designed, dynamic experience. These higher expectations were met with an explosion in new technologies and expansion of web browser capabilities. Today's web developer needs to use tools and a development approach that are aligned with the modern web scene.

Technology has changed. Browsers and JavaScript engines are faster. Workstations and laptops are far more powerful, to say nothing of the plethora of mobile devices now being used to surf the Web. Web service APIs are the expectation for a modern web application rather than a rare additional feature. Cloud computing is revolutionizing the deployment and operation of web applications.

Software development has changed. The now popular "Agile Manifesto" values:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

It is now possible to quickly spin up web applications that prove—at least on a small scale—the viability of a given technology. There is tremendous value to prototyping. As Fred Brooks, author of *The Mythical Man Month* (Addison-Wesley Professional), famously stated: "Plan to throw one away; you will, anyhow." A prototype can allow for early customer or end user interaction that helps solidify requirements early in the process. It is no longer an insurmountable task to write a functional web application in a matter of minutes.

Web Users

Modern web application users have well-defined expectations about how they will be able to interact with a web application:

- Web applications will be available across *multiple platforms*.
- They will provide a *consistent experience across devices*.
- They will respond with *little or no latency*.

The Gartner group claims that in 2014, the personal cloud will replace the PC at the center of users' digital lives. There are many implications for web app development. Users are more technologically savvy and have high expectations for site responsiveness. They are less passive than in previous years and instead are interactive and engaged. Websites need to be designed in a way that suggests no limitations in the ability of a browser to mimic native application experience.

Users expect an application to be exposed in various ways and available in different situations. Responsive design and support for multiple browsers, platforms, and devices are the new norm. The use of JavaScript libraries and frameworks is essential to support the wide variety of target clients.

The *New York Times* recently reported on the impatience of web users. Among its findings: a company's website will be visited less often than that of a close competitor if it is slower by more than 250 milliseconds. Performance needs to be a key consideration in web application development.

Technology

Java web application developers are typically familiar with server-side dynamic content. J2EE and JSP have been refined into JEE and JSF. Projects such as Spring provide additional capabilities geared toward server-side development. This mode of development made a great deal of sense in the early days of the Web, when web pages were relatively static, servers were relatively fast, JavaScript engines were slow, and there were few libraries and techniques to address browser incompatibilities.

By way of contrast, a modern client-server approach involves a server largely responsible for providing access to resources (typically communicated as messages in XML or JSON) in response to client requests. In the old server-driven approach, the browser requested an entire page and it was generated (along with relevant data) for rendering in the browser. In the client-server approach, the server initially serves pages with little data. The pages make asynchronous requests to the server as the user interacts with it and the server simply responds to these events with messages that cause the current page to be updated.

Initial web development efforts consisted of the creation of static HTML sites. Later, these sites were augmented with dynamic content using server-side processing (CGI, Java Servlets). Subsequently, more structured language integration emerged using server-side templating (ASP, PHP, JSP) and MVC frameworks. More recent technologies continue in the same tradition and provide additional abstractions of one sort or another.

Based upon a desire to shield developers from design concerns and the underlying architecture of the Web, component-based frameworks have emerged. Tag libraries were an early innovation, and now a component-based approach has been widely adopted in several popular frameworks:

- Java Server Faces (JSF), an XML-based templating system and component framework with centralized configurable navigation.
- The Google Web Toolkit is another component framework that leverages the abilities of Java programmers by letting them focus on Java coding with little need to directly modify HTML, CSS, or JavaScript.

Each of these frameworks has its place and has been used successfully in production systems. But like many solutions that try to hide underlying complexities, their usage is problematic in situations where you need greater control (such as the ability to integrate large amounts of JavaScript) or you do not conform to the framework assumptions (for instance, availability of server sessions). This is because these solutions attempt to hide the fundamental architecture of the Web, which uses an HTTP request-response protocol following the client-server computing model.

Browser innovations also led to a shift of responsibility from the server to the client. In the late 1990s, Microsoft developed the underlying technologies that led to Ajax (a term coined on February 18, 2005 by Jesse James Garrett). *Ajax* is an acronym for "asynchronous JavaScript and XML," but is more generally applied to various technologies used to communicate with the server within the context of a given web page. This allowed small messages to be sent, which made better use of bandwidth when designing JavaScript-based web applications. Browser performance has increased significantly due to processor improvements and optimizations to JavaScript engines, so it has made sense to offload more work from the server to the browser. User interface responsiveness has evolved to a new level of sophistication.

Mobile device browsers have also provided an additional incentive to further isolate client-side code from the server. In some cases, a well-designed application leveraging responsive design principles can be created. If this is not an option, a single consistent API available for all device clients is very appealing.

Roy Fielding's doctoral dissertation in 2000 led Java EE 6 to new APIs that deviated from the previous component-based trajectory. JAX-RS (Java API for RESTful Web Services) and Jersey (a "production quality reference implementation") are designed to create applications reflecting a client-server architecture with RESTful communications.

Software Development

In the past, setting up a new Java project was a rather monumental task. A vast array of configuration options made it tedious and error-prone. Very little was automated, as the assumption was that each project would have unique characteristics that developers would want to account for to meet their specific requirements.

Later influences led to innovations that made setting up a project much simpler. "Convention over configuration" was an influential mantra of the Ruby on Rails community. Maven and other Java projects also chose sensible defaults and target easy setup for a subset of popular use cases.

The availability of scripting languages on the JVM makes it possible to speed development by bypassing the somewhat rigorous type checking of Java. Languages like Groovy, Python (Jython), and Ruby are loosely typed and constructed in a manner that requires less code to accomplish equivalent functionality. So-called microframeworks like Sinatra or Play provide minimal *Domain Specific Languages* (DSLs) to quickly write web applications and services. And so today, it is a trivial task to set up a minimal set of web services in a development environment.

The failure of enough large-scale waterfall-style software projects has also made it clear that there are many advantages to producing a small-scale version of the final product. A prototype (or prototypes) of the final product can serve many purposes:

- Verify technical foundation of the project
- Create constructs that bridge disparate technologies to be used together
- Allow end user interaction to clarify intended usage and user interface design
- Allow system designers to clarify the interfaces and data structures to be passed between systems
- Allow programmers to work on different parts of the application in parallel

Prototypes have numerous benefits:

- They are a specific, tangible asset representing the final system to be designed. As such, they incorporate information that is otherwise stored in design documents, diagrams, and other artifacts (and frequently in more informal locations like email and people's memories of water-cooler conversations).
- Prototypes are concrete implementations. As such, they present the requirements in a much more tangible form. This can lead to a better understanding of the extent and quality of the requirements gathered, and can suggest areas where there is need of clarification.
- Prototypes can immediately expose potential points of failure that are not apparent before attempting a specific implementation.
- The preceding benefits can lead to better estimates and scheduling due to a more comprehensive understanding of what is intended.

Prototyping can be leveraged extensively in client-server web application development because of the clear and unambiguous separation between the client and server. Prototypes of the server can be provided to the client developers (and vice versa) while development proceeds in parallel. Or if development is not proceeding in parallel, server-side calls can be quickly stubbed out so that client-side code can be developed.

What Has Not Changed

The fundamental nature of the Web (a client-server architecture transmitted over HTTP) has not changed.

New technology does not change everything. High-level programming languages have not removed the need to understand operating system specifics. Object-relational mapping frameworks have not removed the need to understand relational databases and SQL. In like manner, there have been consistent attempts to ignore the underlying architecture of the Web in an effort to emulate the experience of desktop applications.

Medium Specificity

Medium specificity is a term that appears in aesthetics and modern art criticism but which can be applied to technology as well. It indicates the "appropriateness" of a given artistic subject to be presented by a given medium. The idea has been around for centuries. Gotthold Ephraim Lessing states in his Lacoon:

> [B]odies, with their visible properties, are the legitimate subjects of painting. [A]ctions are [therefore] the legitimate subjects of poetry.
>
> — *The Limits of Poetry and Painting*

Its application in modern art is usually to challenge traditional limits that appeared in the arts. Technology is a creative activity, but our primary concern is working systems, not abstract beauty. The idea of medium specificity is important in that, *if you ignore the underlying nature of a platform, the resulting system will never perform in an optimal manner or will not work at all*. This has become painfully obvious in many areas of technology. The goal of this book is to promote web application design strategies that are aligned with the way the Web itself is designed. Such applications operate well because they work within the Web's fundamental constraints rather than ignoring them.

The Nature of the Web

The essence of the Web has not changed. It is still made up of servers that serve HTML documents to clients via the HTTP protocol. See Figure 1-1.

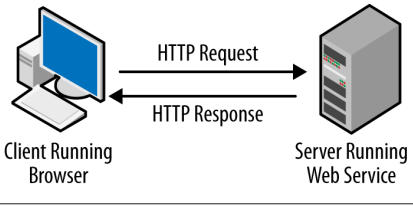

Figure 1-1. HTTP request and response

A client-server web architecture more closely maps to the underlying architecture of the Web itself. Although not technically protocol-specific, REST was developed based upon and in conjunction with HTTP. REST essentially defines constraints on the usage of HTTP. It seeks to describe a well-designed web application: a reliable application that performs well, scales, has a simple elegant design, and can be easily modified (Figure 1-2).