Federico Kereki

Essential GWT

Building for the Web with Google Web Toolkit 2

Developer's Library

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States please contact:

International Sales
international@pearson.com

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data

Kereki, Federico, 1960-
Essential GWT : building for the web with Google Web toolkit 2 / Federico Kereki.
p. cm.
Includes index.
ISBN-13: 978-0-321-70514-3 (pbk. : alk. paper)
ISBN-10: 0-321-70514-9 (pbk. : alk. paper)
1. Ajax (Web site development technology) 2. Java (Computer program language)
3. Google Web toolkit. 4. Application software–Development. I. Title.
TK5105.8885.A52K47 2011
006.7'6–dc22

2010018606

Copyright © 2011 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc.
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116
Fax: (617) 671-3447

ISBN-13: 978-0-321-70514-3
ISBN-10: 0-321-70514-9
Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, IN.
First printing, July 2010

Editor-in-Chief
Mark Taub

Acquisitions Editor
Trina MacDonald

Development Editor
Songlin Qiu

Managing Editor
John Fuller

Project Editor
Anna Popick

Copy Editor
Apostrophe Editing Services

Indexer
Jack Lewis

Proofreader
Linda Begley

Editorial Assistant
Olivia Basegio

Technical Reviewers
Jason Essington
Jim Hathaway
Daniel Wellman

Cover Designer
Gary Adair

Compositor
Rob Mauhar

Contents at a Glance

This page intentionally left blank

Contents

Preface

Developing modern, interactive, complex web sites has become a harder task since users' expectations are higher today. The bar has been raised by the current crop of applications such as Gmail or Google Maps, and developers are expected to work up to that level and provide similarly powerful new web sites. The style, speed, and interaction levels of modern sites practically rival those of classical desktop installed applications, and of course users don't want to go back. How do you develop such sites?

It can be said that the usage of Ajax was what started the trend toward such distinctive applications, but even given that technique, the rest of the development of web pages was the same, tools were the same, testing methods were the same, and the whole result was that the programmers' jobs had gotten much harder than needed.

(Personally, I should confess that I really never liked classic-style web development: Building large-sized applications was harder than it needed to be, JavaScript was—and still is—missing constructs geared to complex systems, the click-wait-click-wait again cycle was inevitably slow and not very interactive, and, to top it all, unless you were quite careful with your testing, your design was prone to fail on this or that browser in unexpected ways.)

GWT, in just a very few years, has grown into a powerful tool by harnessing the power of Java and its considerable programming environment and many development tools, and producing efficient and consistent output, despite the too-many and well-known incompatibilities between browsers.

Getting started with GWT isn't that hard—documentation is reasonably good, the development environment can be Eclipse or several other equally powerful IDEs, and programming is quite similar to old-fashioned Java Swing coding—so you can have your first short application up and running in a short time.

Creating production-quality, secure, internationally compliant, high-level code can be, however, a bit more complex. You need to take many factors into account, from the initial setup of your project and development of the user interface, to the final compile and deployment of your application.

Similarly, we'll also have to focus on methodologies and on software design patterns, so we can go forth in a safer, more organized way toward the complete application. For example, we'll consider how the model-view-presenter (MVP) pattern can not only enhance the design of the application, but also help run fully automatic tests, in modern Agile programming style, to attain higher quality, better tested software.

We'll be working with the latest tools and versions; not only GWT's (2.0.3 just now), but also Eclipse, Subversion, Tomcat, Apache, MySQL, and so on. Because all these tools

are open source, we can support the notion that an appropriate software stack can be built starting with GWT and ending with a full open web solution.

After my earlier confession on my dislike of classic web development strategies, I should now aver that GWT did change that for me. Working in a high-level setting, with plenty of tools, and practically forgetting about browser quirks, HTML, CSS, and JavaScript, while gaining in clarity, maintainability, and performance, has made web application creation an enjoyable task again!

The Structure of This Book

Chapters 1 through 3 deal with the basic setup for working with GWT. After considering the main reasons and objectives for using GWT, we'll study what other tools are required for serious code development, the methodology to use, and the internal aspects of projects.

Chapters 4 and 5 are the backbone for the book, for they deal with the basic design patterns that we use for building the User Interface. The code style and idioms developed here will be used throughout the rest of the book.

Chapters 6 and 7 deal with communications with servers, either through RPC (to connect with servlets) or through direct Ajax (to communicate with remote services).

Chapters 8 and 9 study how to add both JavaScript coding and third-party APIs to your application. Together with the previous two chapters, everything that's needed for mashing up services and getting information from different sources will have been covered.

Chapters 10 and 11 have to do with common server related problems, such as security aspects, and file upload and download.

Chapter 12 deals with developing GWT applications that will be used worldwide and covers both internationalization and localization.

Finally, Chapters 13 through 15 consider general themes such as testing GWT applications, optimizing their performance, and finally deploying them.

Who Should Read This Book

This book goes beyond "just learn GWT," and is targeted to programmers who already have a basis of GWT programming and want to encompass other web applications, services, APIs, and standards as well, to produce Web 2.0-compliant Rich Internet Applications (RIAs). A previous experience with web development, possibly in a J2EE environment, will come in handy.

Having read this book through, the reader should not only be able to develop a RIA on his own by just using GWT, but he will also have a reference book to help solve the common problems that arise in such applications. Complete source code is given for all examples, so getting started is quicker.

Web Resources for This Book

The Google Web Toolkit site at http://code.google.com/webtoolkit/ is a mandatory reference, and so is the forum at http://groups.google.com/group/google-web-toolkit.

The code examples for this book are available on the book's web site at www.informit.com/title/9780321705143.

1

Developing Your Application

Why would you use GWT? What can you develop with it and how? Before delving into specifics (as we'll be doing in the rest of the book) let's consider the answers to these questions, so you'll know what to focus on.

Developing applications with GWT can be seen as a straightforward job, but you should ask some interesting questions to unlock the way to powerful, distinct, applications. What kind of applications should you develop with GWT? (And, given the current push for Cloud Computing, you can even add "Where would you deploy your application?") How can you go about it? And, why would you use GWT?

Let's consider all these questions in sequence to start you on your way through this book, knowing your goal and the road to it.

Rich Internet Applications

When you start reading about Rich Internet Applications (RIAs), your JAB (Jargon, Acronyms, and Buzzwords) warning should go off because there are many words that are bandied about, without necessarily a good, solid definition or a clear delimitation of their meanings.

Basically, what we build are web applications that have the look and feel of classic desktop applications but that are delivered (and "installed") over the web. Many tools have been used for this purpose, such as Java (through applets), Adobe Flash, and more recently, Microsoft Silverlight, but used in this way, all these tools are beaten, in terms of practicality, by simple HTML-based systems.

The RIAs that we will be developing are based on JavaScript and Ajax and just require an appropriate browser to run. Classic web applications were developed with a different set of tools, subjected the user to frequent waits (the hourglass cursor was often seen), and had severe restrictions as to usability, with a much clunkier feel to them than desktop installed programs.

Although some people distinguish between RIAs and the kind of interactive web applications we build, the frontiers are getting blurrier and blurrier. You could argue that Flash or Silverlight require preinstalled plugins, or that development runs along different

lines, but in terms of the final result (which is what the user experiences) differences are not so marked, and well-designed HTML/JavaScript/Ajax applications can compete for equality with applications developed with the other tools. (Also, some people opine that HTML 5 can seriously challenge Flash, up to the point of making it obsolete, but that's still to come.[1]) There used to be obvious differences—the ability to store local data at the user's machine was the biggest one—but tools such as Google Gears or current developments in HTML 5 have provided this feature to web applications.[2]

Given its ubiquity (from desktops to netbooks, and from cell phones to tablet PCs) the browser can be considered a universal tool, and Ajax provides the best way for the creation of highly interactive applications. Of course, a few years ago there weren't many tools for doing this (GWT itself appeared in 2006) and creating heavy-lifting interactive code with just JavaScript wasn't (and still isn't) an appealing idea.[3]

Furthermore, given that users have been subjected for many years to web applications, and are familiar with their idioms, you are a bit ahead in terms of user interface design by keeping to a reasonable standard.

As for the language itself, using Java as a tool—even if it gets compiled into JavaScript, as GWT does—provides both a way around JavaScript's deficiencies and introduces a widely used language with plenty of development tools, which has been used over and over for all kinds of applications and has been proved to scale to large-sized applications.[4]

Web 2.0

Web 2.0 is another expression that has been bandied about a lot since its invention in 2004. Though there are way too many definitions for it, most seem to agree on the idea of using the "Web as Platform," where all applications run in a browser instead of being preinstalled on your desktop. Furthermore, the idea of allowing users to produce their own contents (à la Wikipedia) is also included, highlighting the collaborative aspect of work, and thus bringing into the fold all kind of community and social networking sites (think Facebook or YouTube). Finally (and that's what actually works for us) the concept of *mashing* together different data sources (probably from many web services) is also included.

1. See www.ibm.com/developerworks/web/library/wa-html5webapp/ for an article of some HTML 5 features already available in current browsers.

2. Google Gears' development was practically stopped (other than support for currently available versions) by the end of 2009 because of the upcoming HTML 5 features for local storage.

3. It might be said that developing large applications with, say, Flash, isn't a walk in the park either, for different reasons to be sure, but complicating the programmer's job in any case.

4. It should be remarked that GWT isn't the only such compile-to-JavaScript solution; for example, the Python-based Pyjamas project (http://code.google.com/p/pyjamas/) provides Python-to-JavaScript translation, and there are many more similar tools.

GWT applications can obviously be used for producing highly interactive people sites, but they can also link together information from different origins, consuming web services with no difficulty, either connecting directly to the server or by means of proxy-based solutions. Various data formats are also not a problem; if you cannot work with such standards as XML or JSON, you can include external libraries (or roll out your own) through JSNI or Java programming. (We cover this in Chapter 8, "Mixing in JavaScript," and Chapter 9, "Adding APIs.")

In this context, the phrase Service-Oriented Architectures (SOA) frequently pops up. Instead of developing tightly integrated, almost monolithic, applications, SOA proposes basing your systems on a loosely integrated group of services. These services are general in purpose and can be used in the context of different applications—and, as previously mentioned, GWT is perfectly suited to "consuming" such services, dealing with different protocols and standards. (We'll cover this in Chapter 6, "Communicating with Your Server," and Chapter 7, "Communicating with Other Servers.") If your company is centered on an SOA strategy, your GWT-developed applications will fit perfectly well.

Cloud Computing

Next to the idea of using the browser as the basis for the user's experience, the most current term related to modern application development is *Cloud Computing*. This idea reflects the concept of sharing resources over the web, on demand, instead of each user having a private, limited pool of resources. In this view, software is considered a "service" (the acronym SAAS, which stands for "Software as a Service," is often used) and a resource similar to more "tangible" ones as hardware.

(As an aside, the vulnerability of some operating systems, most notably Windows, to viruses, worms, and similar attacks, has given a push to the idea of using a simple, secure, machine and storing everything "on the web," letting the cloud administrators deal with hackers and program infections.)

For many, this concept is yet another cycle going from centralized resources (think mainframes) to distributed processing (PCs, possibly in client/server configurations) and now to having the web as your provider. The main requirements for such an architecture involve reliable services and software, delivered through specific data centers, and running on unspecified servers; for the user, the web provides an access to a cloud of resources.

For GWT applications, your applications are basically destined from the ground up to be used "in the cloud" because of the standard restrictions imposed by browsers. Distributing an application over the web, accessing it from anywhere, and having your data stored in a basically unknown place are all characteristics of any applications you might write.[5]

5. With current (or forthcoming) standards, you might also resort to storing data locally, or to using your own private, dedicated, resources, but that's not original and more often associated with classic desktop applications.

The "Death of the Desktop"

The trend toward Cloud Computing has even spawned a new concept: the "Death of the Desktop." This presents rather starkly the problem of going overboard, to the limit: From the appearance of mini netbooks (with flash-based disks, slow processors, not much RAM) and iPhone-look-alike cell phones, some have reached the conclusion that desktop applications (and even desktop computers!) are on their way out. If this were true, it could be great for GWT developers, but things are a bit different.

Despite several impressive opinions and pronouncements from people all over the industry, the trend toward more powerful machines, with CPUs, memory, and I/O facilities that put to shame the supercomputers of just a few years ago, doesn't seem to be slowing down. Even if you are enamored with the latest netbooks or high-powered cellphones, you should accept that working all the time with minimal screens isn't the way that things can get done at a company. (And for gaming or graphic-intense usages, small machines aren't so hot either; they may do, however, for business-oriented applications.) In any case, GWT can help you because you can use its layout facilities and CSS styling to produce applications for just about any device out there.

Also, remove the rosy glasses for an instant. Cloud computing offers several advantages (and GWT applications can be considered to be right in the middle of that concept) but also presents problems, so you need to plan accordingly. Aside from the obvious difficulty of dealing with possibly flaky web connections, security and compatibility can be stumbling blocks. (On the other hand, scalability is well handled; there are plenty of large sites, with hundreds or thousands of servers, proving that web applications can scale well.) The important point is, with or without desktops, GWT provides some ways around these kind of problems, and we'll study this in upcoming chapters.[6]

Advantages of GWT

Why would you develop with GWT? Shouldn't directly using JavaScript make more sense? How do you manage with browser quirks? Let's consider the reasons for GWT.

HTML Ubiquity and Browser Differences

The first reason for GWT applications is the ubiquity of HTML. Even if some time ago browsers for, say, cell phones, weren't as capable as their desktop brethren, nowadays you can basically find the exact same capabilities in both. In terms of GWT, this is a boon because it means that a well-designed application can run and look pretty in devices from 3 inches to 25 inches.[7]

6. And, of course, these inconveniences haven't stopped anyone from developing HTML-based applications!

7. Don't expect to get the screen design right the first time; managing to build clear, small screen browser applications is more an art than a science.

This availability is somehow tempered because today's browsers are not created equal—but you certainly knew that if you designed web pages on your own! When Microsoft's Internet Explorer ruled the roost, having practically 100% of the browser market, this wasn't a noticeable problem. However, today browser usage statistics point to a different status quo: Mozilla Firefox and Safari, among others, have started carving larger and larger niches in the market, and in some countries (mostly European) they have outnumbered Internet Explorer. The current trend is toward applying web standards, and that bodes well for web developers. In any case, GWT is quite adept at solving browser quirks and differences, so the point may be considered moot for the time being.

JavaScript Deficiencies

Even assuming fully standard-compliant browsers, the fact remains that JavaScript, no matter how powerful, isn't a good language from the specific point of view of software engineering. Because this isn't a book on JavaScript, we won't delve in its main problems, but using it for large-sized application development can be, to say the least, a bit complicated.

This language isn't well adapted either to development by large groups of people, and the tools it provides for system development aren't that adequate, so the programmer must add extra code to bridge the distance between a modern object-oriented design and its actual implementation.

One solution that has been applied is the usage of different libraries that provide a higher-level way of using the language.[8] GWT solves this problem in a radically different way, by enabling the use of the higher level Java language, for which there are plenty of modern development, testing, and documentation tools.

Software Methodologies to Apply

For classic application development, many well-known methodologies exist, but in the context of modern web development, you should definitely use some techniques.

Classic Development Problems

If you learned to develop systems years ago, you were surely exposed to the Waterfall Model or some other methodologies directly based on it. In this model for the development process, progress is seen as flowing like a waterfall from stage to stage, through

8. You could consider Google's "Closure" library (see http://code.google.com/closure/) used for Gmail's development, or Yahoo!'s YUI library (see http://developer.yahoo.com/yui/), jQuery (http://jquery.com/), Dojo (www.dojotoolkit.org/), Prototype (www.prototypejs.org/), MooTools (http://mootools.net/), and many others. The functionality of these libraries isn't always the same, but there's considerable overlap between them, showing the problems they set out to solve are real and well known.

well-defined phases (see Figure 1.1) starting with the Analysis of Requirements, following with the Design of the Solution and its Implementation, then to Testing (or Quality Assurance), and finally to Installation and future Maintenance.

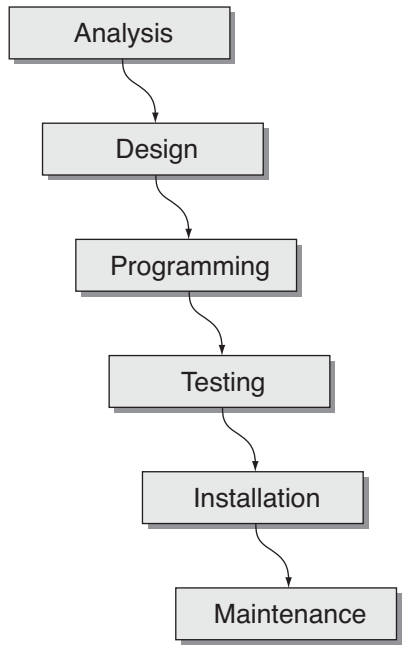

Figure 1.1 The classic Waterfall Model isn't the best possible for GWT development.

This model is flawed in several ways (and of course, there are some fixes for that) but its main problem is its orientation to highly regimented industries such as Construction, in which late changes can be quite costly to implement, usually requiring tearing down what was done and practically starting anew.

Another point—and an important one—is that you cannot expect users to be fully aware of what they require; it is sometimes said "Users don't know what they want, but they know what they don't want."[9] Classical methodologies do not take this into consideration, and might thus incur important costs, because newly discovered or determined requirements can invalidate a previous design.

Finally, it's difficult to predict where difficulties will occur; problems with functionality are usually found "on the go," and if going back to change something to help future development is too costly, you can face a dilemma: Spend money and time revising your

9. "I'll know it when I see it" is another way of expressing this.

design, or keep your substandard design, and spend money and time later trying to make your software do tasks it wasn't well designed to do.

It has been said that the Waterfall Model, and similar ones, are based on the old "Measure Twice, Cut Once" saw, but you cannot actually apply this when you don't actually know what's being measured! (And, furthermore, what happens if requirements change along the way, and by the time you finish with development, the problem has actually changed?) Modern, agile technologies try to take this into account and work in a radically different way, and that's the way you should use with GWT.

Agile Methodologies

Several software development methodologies seek to reduce the time between the requirement analysis phase and the development phase to develop at least parts of the system in shorter times, using possibly an iterative method to advance to the final application. Prototypes are frequently used to bridge the distance between the user and the developer, helping both to understand what's actually required. Instead of attempting to do a whole system at once, development is parceled in smaller subsystems. The user is involved all the time, instead of providing his input (in the form of requirements) only at the beginning and then dealing with the system after its installation.

All these suggestions are currently applied in Agile Software Methodologies (born in 2001) that emphasize collective (i.e., users plus programmers) development of systems, in highly iterative steps, with frequent verification and (if needed) adaptation of the written code.

Agile Methodologies usually break a complex system into several short stages, substituting short, easily measured and controlled iterations, for long-term (and hard to do) planning. Each iteration (usually shorter than a month) involves a mini development cycle that includes all the stages associated with a Waterfall Model but finishes with giving the users a working product with increasing functionality that serves not only as a measure of advance, but also as an aid to determine if changes are needed. The delivered software is used as the main measurement for progress, instead of depending on a Gantt chart or other documents.

GWT is perfectly suited to such methodologies, because it can offer iterative development, rapid prototyping (and here tools such as UiBinder, which we will study, can help quickly develop appropriate interfaces), and automated testing. The latter point is particularly important: Given that development can (and will) go back and forth, and code used in a previous iteration can be modified several times along the complete development process, it's important to check whether old functionality hasn't been lost and whether bugs have been introduced. GWT has tools that provide for both unit testing (at the lowest level) and acceptance testing (at the user level).

Forever Beta?

As a side effect of the iterative development process, it's usually hard to define what constitutes a "version" of the final system. Because practically every iteration produces new

functionality, and the final goal isn't as well defined as with classic methodologies (in which the complete roadmap is laid out at the beginning and then preferably left unchanged) with iterative development, you deliver the system in many small steps, rather than in large ones.

In this context, it's not unknown for systems to be considered in "perpetual beta"; beta testing refers to the tests done by actual users with a system that is close to the full product but not necessarily complete. (An extreme case of this is Google's Gmail, which was considered to be at beta level from 2004 to 2009!) With GWT, you can provide functionality increases in short steps, and the web model enables for easy distribution of the updated code.[10]

Summary

We touched upon several considerations that impact web application development. In the rest of the book, we will be elaborating on them and provide specific techniques to help you develop company-sized RIAs with the expected levels of quality and functionality.

10. This could be said, of course, of any web-based application not necessarily written with GWT; the point is that GWT helps you work this way.

Index

Numbers and Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q–R

S

T

U

V

W

X

Y–Z