APACHECON
Europe

10-10.11.16

SEVILLE, SPAIN

Apache Wicket: The story so far and beyond

Andrea Del Bene, Java Senior consultant, Innoteam srl

APACHE WICKET™

THE LINUX FOUNDATION

Agenda

1) The story of Wicket so far
2) The state of Wicket and its community
3) Wicket against the *client-side* revolution ☺
4) A look to three distinguishing Wicket features
   - Resource handling
   - AJAX support
   - Testing support

Who I am

Andrea Del Bene

- Senior Java consultant at Innoteam srl
- Apache PMC member since 2013
- ASF member since 2015

Wicket in a super-small nutshell

- Created in 2004 by Jonathan Locke
- Component-oriented framework *a-la* Swing:

Java Code

```
import org.apache.wicket.markup.html.WebPage;
import org.apache.wicket.markup.html.basic.Label;

public class HelloWorld extends WebPage {
  public HelloWorld() {
      add(new Label("message", "Hello World!"));
  }
}
```

HTML Template

```
<html>
<body>
    <span wicket:id="message">
      Message goes here
    </span>
</body>
</html>
```

- Just Java and plain HTML (and JavaScript and CSS)
- Each component (Label, Link, Button, etc…) is binded to HTML with attribute `wicket:id`

Wicket rationale

Wicket was designed to:

- make components reuse possible
- minimize configuration artifacts
- make server side state management easy
- be as type safe as possible

When Wicket was born there were MANY alternatives to it...

APACHECON Europe

A a long time ago, in this galaxy....

THE LINUX FOUNDATION

Few have survived...one is Wicket

We're still in the game!

JSF	34.5%		463
Spring MVC	34.2%		459
Wicket	16.3%		219
Other - Write In	7.8%		104
Play!	7.3%		98
I build web apps without a framework	6.7%		90
Struts	6.4%		86
GWT	6.1%		82
Vaadin	5.2%		70
Grails	5.1%		69
Tapestry	1.7%		23
Lift	0.2%		3
		Total	1,342

Web frameworks usage survey by DZone, October 2015

https://dzone.com/articles/survey-confirms-jsf-remains-leading-web-framework-2

Wicket state of health

- Current major version: 7 (supports Java 7 and Servlet API 3.0)
- Last released version: 7.5.0
- Version 8 on its way (M2) with support for Java 8
- Full support for the last web technologies and protocols (HTML5, Web Socket, HTTP 2,...)

Since 2013 lot has been done also to revamp our community

Rebooting a community...

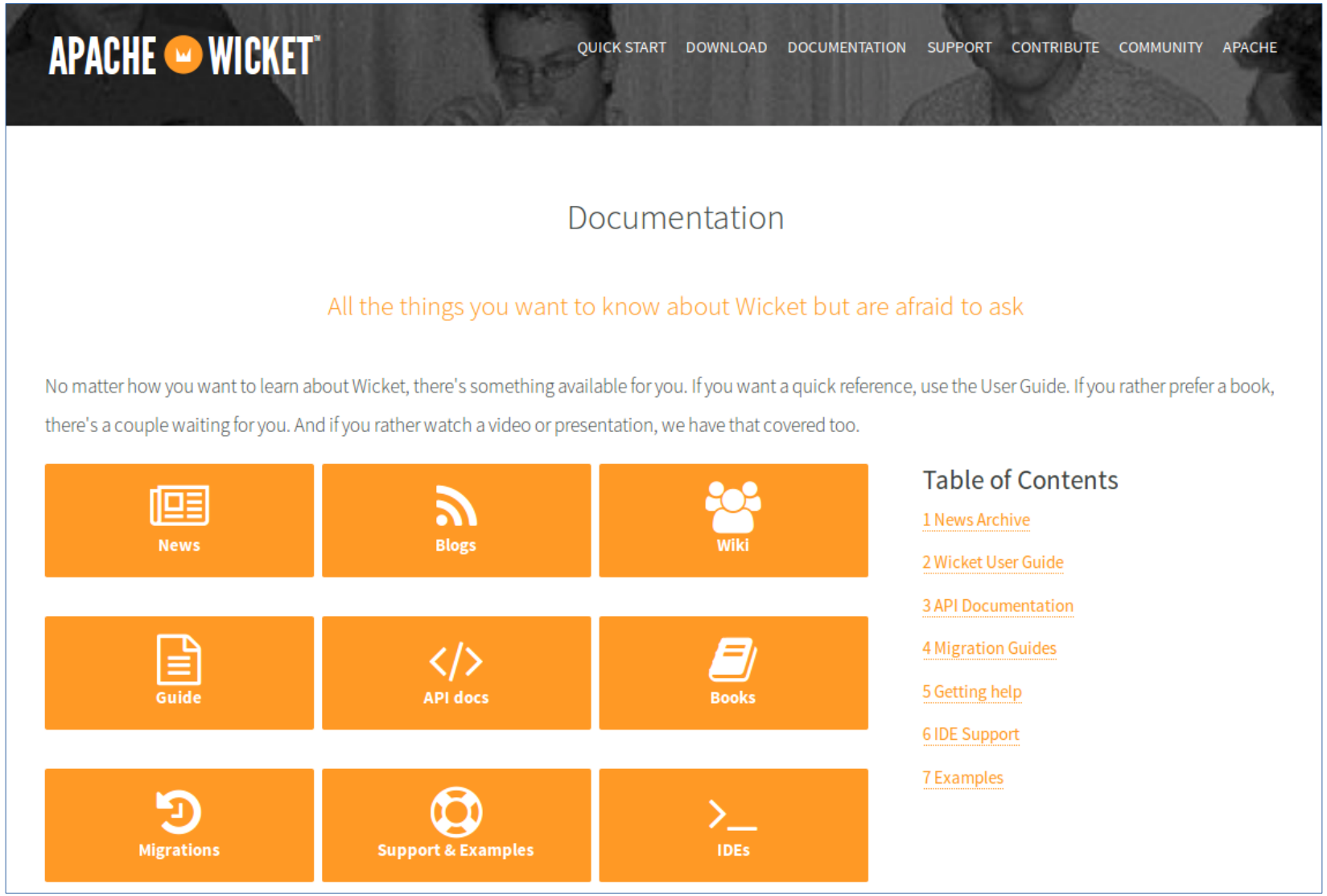
Back in 2013 Wicket was loosing its appeal due to the lack of two fundamental elements:

- Visibility on the web: a modern and well-structured site, a presence on social channels (GitHub, Tweeter, Linkedin, etc..), etc...
- A free, exhaustive and up to date documentation.

And that's what we did to cope with this issues

A new revamped site

Well-organized contents



An extensive user guide

wicket.apache.org/learn/#guide

Wicket User Guide

Learn building web applications with Wicket from scratch reading its 200+ page user guide. The guide gradually introduces you to the various features of the framework with many real-world examples. It covers subjects such as models, behaviours, testing and integration with other projects.

The guide is available as PDF or html file for the following versions:

1. Wicket 8.x
   - HTML
   - HTML (single page)
   - PDF
2. Wicket 7.x
   - HTML
   - HTML (single page)
   - PDF
3. Wicket 6.x
   - HTML
   - HTML (single page)
   - PDF

You can use the guide for older releases even though there will be differences. We urge you however to upgrade your project to the latest stable release rather than sticking on an older version.

Friend projects

The Apache Software Foundation

Proudly Powered by the ASF

It is nice to have a welcoming home, and when you find that home, friends are not far away. Home to over a hundred projects, the Apache Software Foundation is a power house for open source community driven development. Wicket is proud to be part of the Apache Software Foundation since 2007.

Friends

Being a member of the Apache Software Foundation creates opportunities for communities to collaborate and stand on one's shoulders. The following friends within the Apache community are proud members using Wicket.

Apache Isis

Apache Isis is a full-stack framework for rapidly developing domain driven apps and RESTful APIs in Java. It uses Wicket 6.x as a default viewer for Isis domain model.

Apache OODT

Apache OODT is a data grid framework for transparent search and discovery of disparate science resources. It uses Apache Wicket for its CAS File Manager and CAS Workflow monitor.

Apache Openmeetings

Apache Openmeetings provides video conferencing, instant messaging, white board, collaborative document editing and other groupware tools using API functions of the Red5 Streaming Server for Remoting and Streaming. It uses Apache Wicket for its web interfaces.

Apache Syncope

Apache Syncope is an Open Source system for managing digital identities in enterprise environments. It uses Apache Wicket for its web management console.

Apache Nutch

Apache Nutch is a well matured, production ready web crawler. Since version 2.3 it uses Apache Wicket for its web management console.

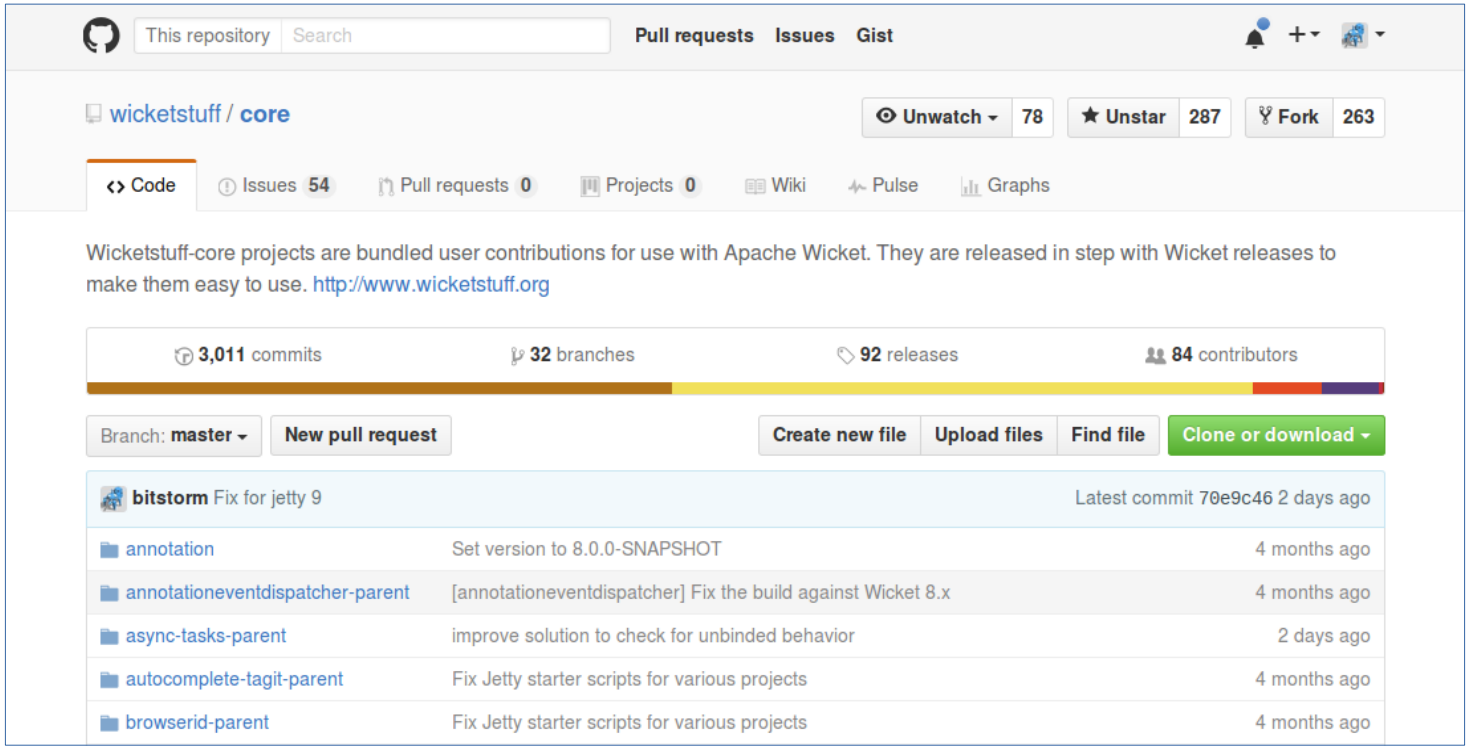
Apache Fortress

Apache Fortress is a standards based and Open Source Identity Access Management Java SDK for LDAP v3 compliant systems. Fortress is 100% compliant with RBAC, ARBAC02 and IETF's password policy draft. It uses Apache Wicket for its RBAC Web Management UI.

Table of Contents

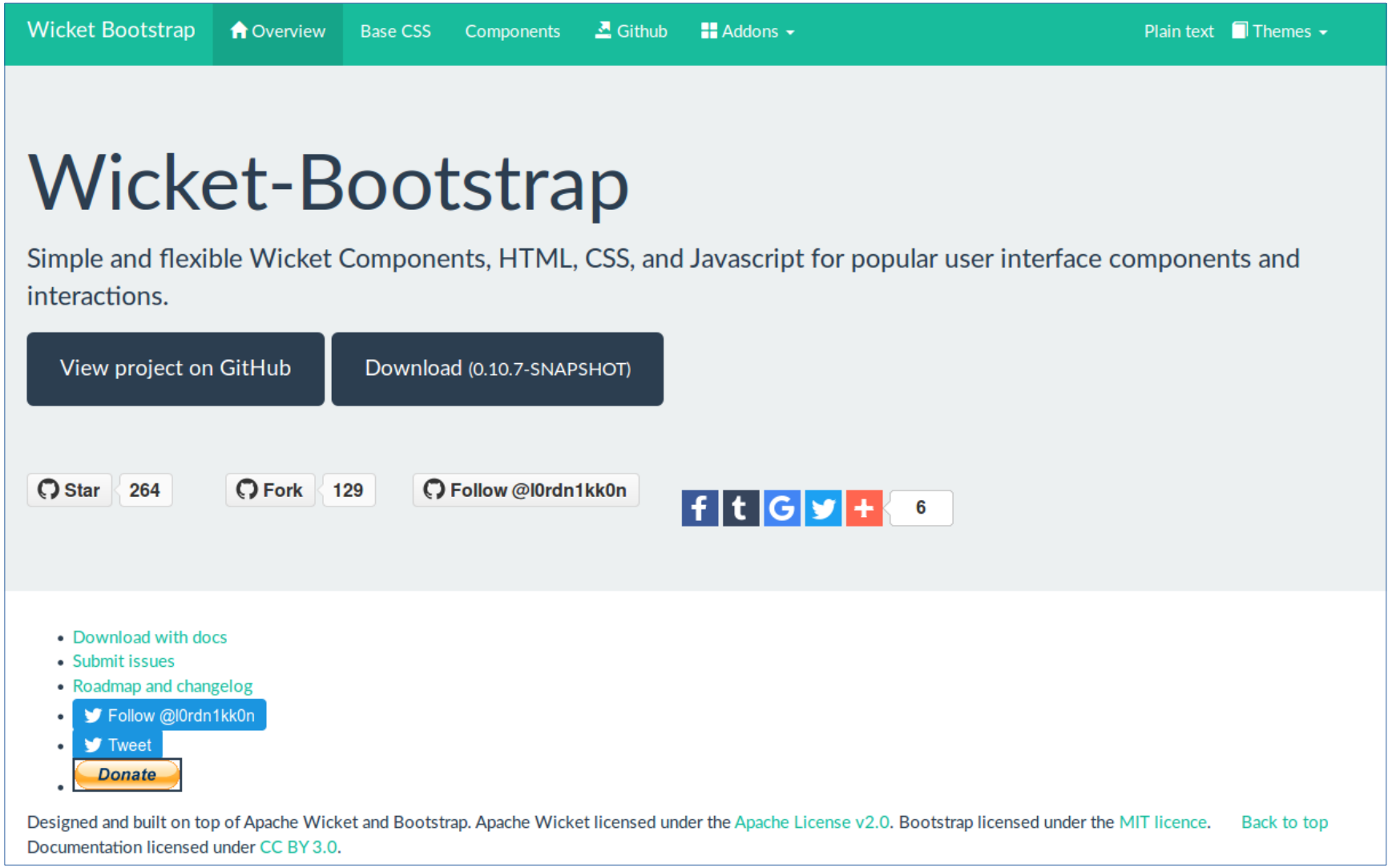
WicketStuff project

- *WicketStuff* is an umbrella project that gathers many other community-provided sub-projects offering many functionalities spanning from integration with popular JavaScript frameworks (like *TinyMCE*, *FoundationJS*, etc...) to advanced features such as async tasks execution or *RESTFull* applications.



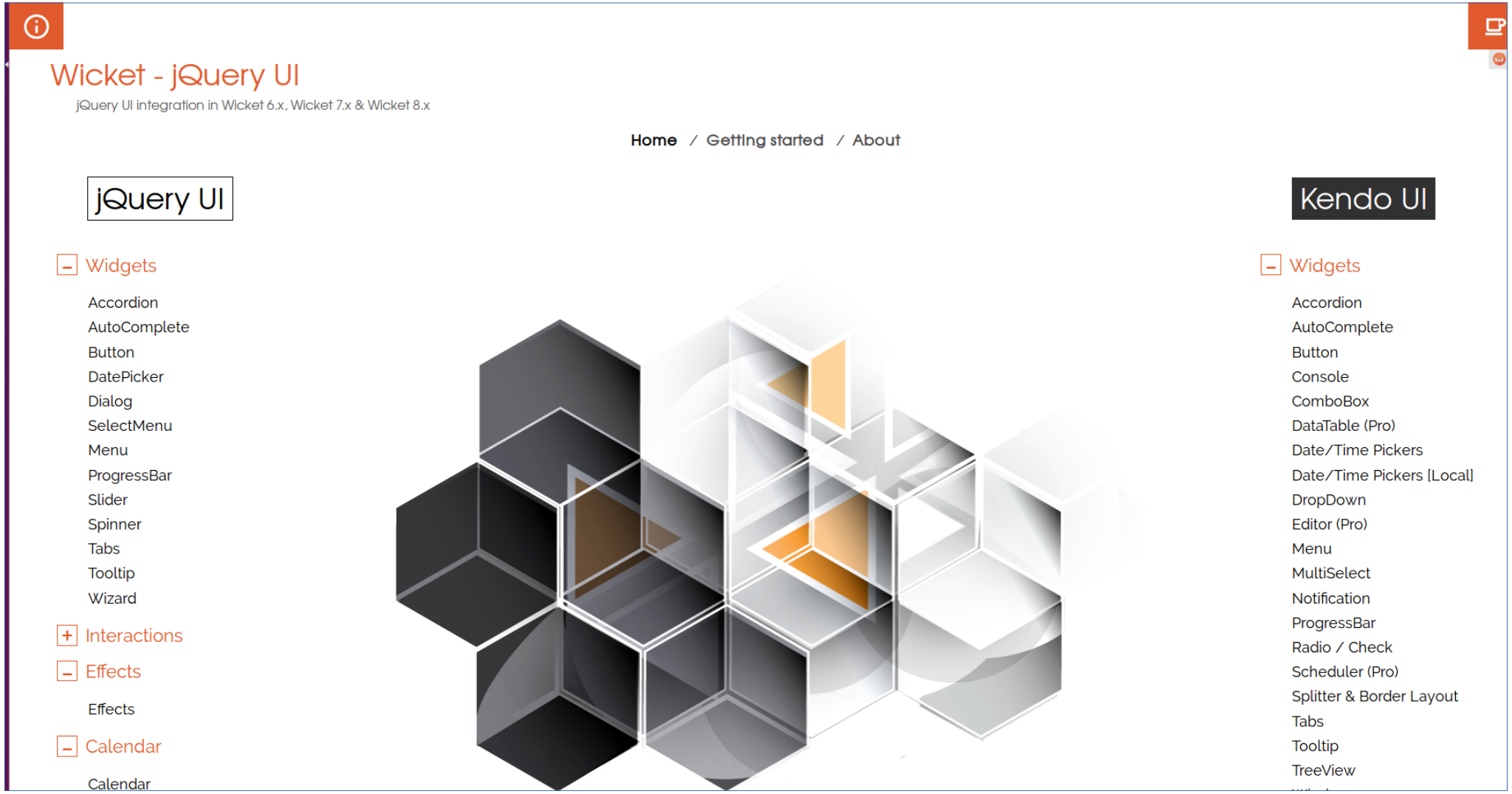
https://github.com/wicketstuff/core

Integration with Bootstrap



http://wb-mgrigorov.rhcloud.com/

Integration with JQuery UI and Kendo UI



http://www.7thweb.net/wicket-jquery-ui

Who is using Wicket?

So everything is good for Wicket... or not?

The client side “revolution”

- In the last 3-4 years the trend is to move our application from server to client
- REST architecture has become very popular and We have seen the rise of client side JavaScript frameworks (Angular, React, etc...).
- The adoption of JSON as data format (both for exchange and for persistence) has made quite cheap developing CRUD application in JavaScript: a single language (JavaScript) for both code and data model.

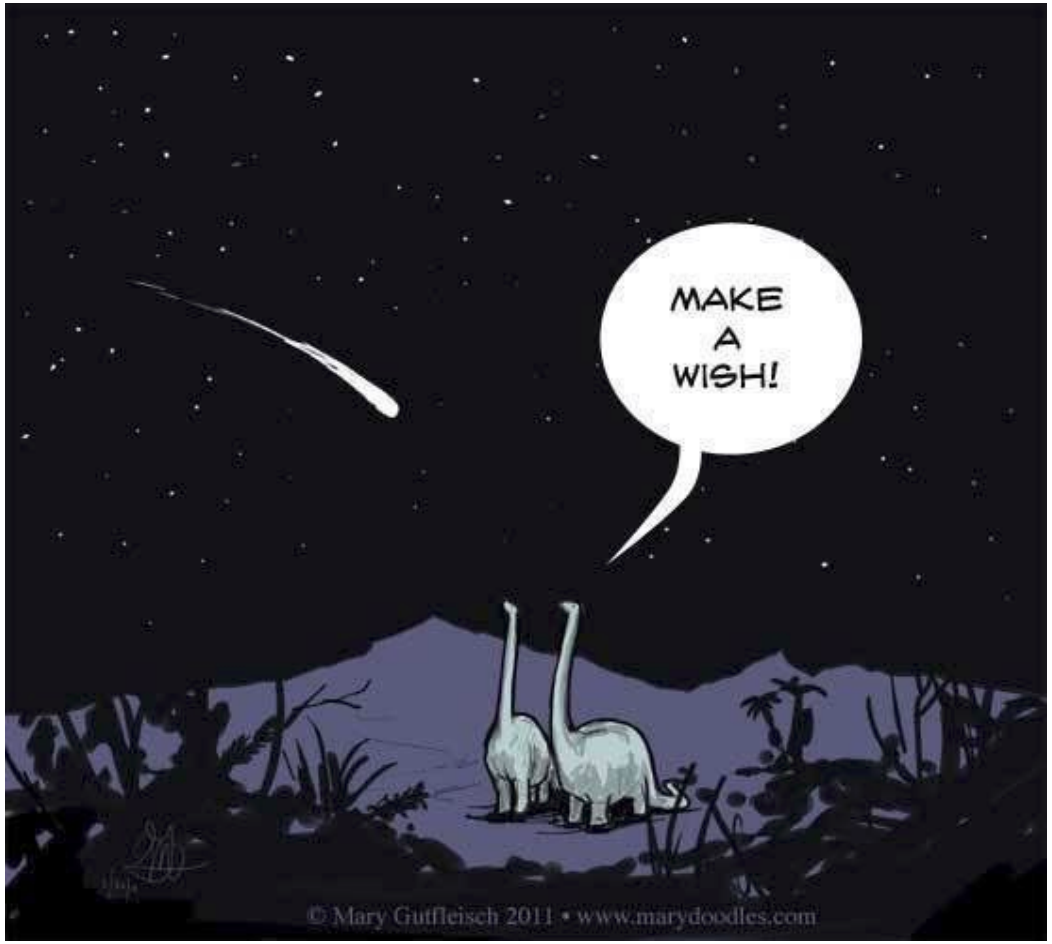
The client side “revolution”

- Single-page application (SAP) everywhere: no room for other options.
- JavaScript also on server-side (Node.js).

The hipster era has begun!

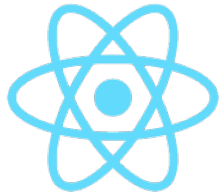
It seems server-side frameworks are now old-fashion and uncool :-(

Are we headed for extinction?



REST + JavaScript + SPA are the new kings. Shall we prepare to extinction?

The dawn of a new frameworks war?



React

polymer

Vue.js

MVVM Made Simple

Who’s gonna be the next *ultimate* framework?

And we have brand new language(s)

polymer

Vue.js

MVVM Made Simple

Who's gonna be the next *ultimate* framework?

Old-school JavaScript is not enough

- Developing in JavaScript today requires a complex stack of tools (NPM, Node.js, Bower, Grunt/Gulp, Babel, etc...)
- We have to compile a Script language...

- For some of these tools is really hard to cope with their release policy:
  Node.js has 4 Major active release (4, 5, 6, 7). Versioning for LTS is unclear (4.2.0, 6.9.0,...)
- A very fragile dependencies handling. Remeber what happened on March?

Jenga-style dependencies

Log in | Sign up | Forums Events | Whitepapers | The Channel | The Next Platform

The Register®
Biting the hand that feeds IT

DATA CENTRE SOFTWARE NETWORKS SECURITY TRANSFORMATION DEVOPS BUSINESS HARDWARE SCIENCE
EMERGENT TECH BOOTNOTES

Software ▸ Developer

How one developer just broke Node, Babel and thousands of projects in 11 lines of JavaScript

Code pulled from NPM – which everyone was using

Careful, careful ... Don't fumble this like the JS world (Credit: Claus Rebler)

23 Mar 2016 at 01:24, Chris Williams 1322

Updated Programmers were left staring at broken builds and failed installations on Tuesday after someone toppled the Jenga tower of JavaScript.

A couple of hours ago, Azer Koçulu unpublished more than 250 of his modules from NPM, which is a popular package manager used by JavaScript projects to install dependencies.

More like this

Javascript

Most read


Windows updates? Just trust us, says Microsoft executive


Hubble telescope spies massive 'cannonballs' of fire from dying star


Don't panic, but a 'computer error' cut the brakes on a San Francisco bus this week


FBI wants to unlock another jihadist's iPhone


Dell has laid off EMC people at Hopkinton

Spotlight

http://www.theregister.co.uk/2016/03/23/npm_left_pad_chaos/

Someone is changing is mind...

Post excerpt:

A: I would still use a Typescript + SystemJS + Babel combo if I were you.

B: I need to display data on a page, not perform Sub Zero's original MK fatality.

Maybe server side frameworks are not ready yet to be put aside...

Original article

Want more fun?

HACKERNOON Follow Sign in / Sign up

HOME TECH PRODUCTIVITY GROWTH FOUNDER STORIES LETTERS FAVORITES

Liz Bennett Follow

Java engineer working on scalable data platforms. A lover of engineering philosophy and a studen…

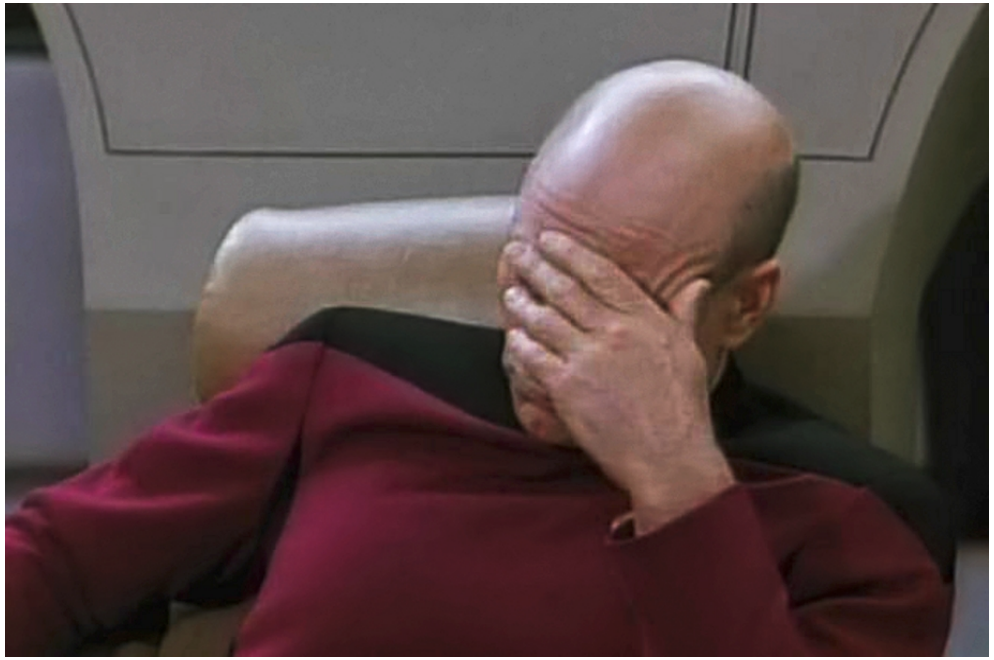
Sep 18 · 6 min read

Why Learning Angular 2 Was Excruciating

A few months ago I decided to try my hand at web development. Angular 2 was new and shiny at the moment, so I decided to download it and start building a website. Now, as someone who is primarily a Java developer, this was an altogether new and very interesting experience. I followed the alleged "5 Minute Quick Start Guide" and after an hour and a half of wrangling with Angular 2 and its plethora of dependencies, I was able to get something up and running.

Next, I started to build a real app. I decided to build a personal blog platform from scratch, mostly for the learning, and partially because I've been meaning to start a blog for ages. The desire to have a blog would be the carrot keeping me motivated to learn new technologies and keep building my project.

Yah, yah, but JS is stateless and scalable, Wicket isn't, right?

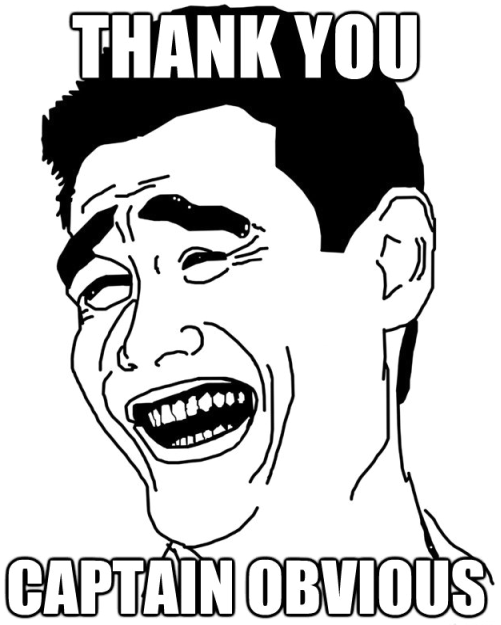


- Let's keep it short: if users can login into your application, then it can not be stateless! User status must be kept somewhere and somehow.
- For many years Wicket was labeled as *stateful* framework. This is absolutely wrong! It was designed to make server side state management easy, but you can choose to be stateless as well!

The bottom line (by Captain Obvious)

- The easiest way to build a *scalable* applications is to keep it *stateless* ☺.

- If you can not keep the application *stateless*, than keep user session as small as possible ☺☺☺.



Ok, but why choose Wicket?

There are some unique features that give Wicket an edge on its competitors s

- Resource handling
- Testing support: applications and components can be tested in isolation.
- AJAX support: develop AJAX-enhanced applications (almost) without writing any JavaScript code.

Resource handling

APACHE WICKET™

THE LINUX FOUNDATION

Resource handling

- With “resource” we indicate both static resources (such as JS and CSS files) and dynamic resources (those who returns they value on the fly [ex: a RSS]).
- From a technical point of view, in Wicket a resource is just an implementation of interface *org.apache.wicket.request.resource.IResource*.
- Working with dynamic resources is less frequent in Wicket and it implies a custom implementation of *IResource*.
- On the contrary handling static resources is a very common task. With Wicket we can specify not just the static resource to load, but also its dependencies and its loading priority.

Static resources

- In general static resources are loaded from 3 possible sources:
  - A generic file from local filesystem
  - A file from classpath (via ClassLoader)
  - An URL
- In Wicket resources are instantiated using a *reference* to them rather than directly. In this way they can be lazy-loaded the first time they are requested.
- Resource references are instances of class *org.apache.wicket.request.resource.ResourceReference*.
- Most of the time static resources are JS or CSS files which can be referred to as *header items.*

Header items

- As the name suggests, an header item is simply an element that is placed inside the <head> tag of the page. In Wicket header items are usually built from a resource reference and are instances of class *org.apache.wicket.markup.head. HeaderItem* (for example *JavaScriptHeaderItem*)
- A page or one of its component can add an header item overriding its method *renderHead*:

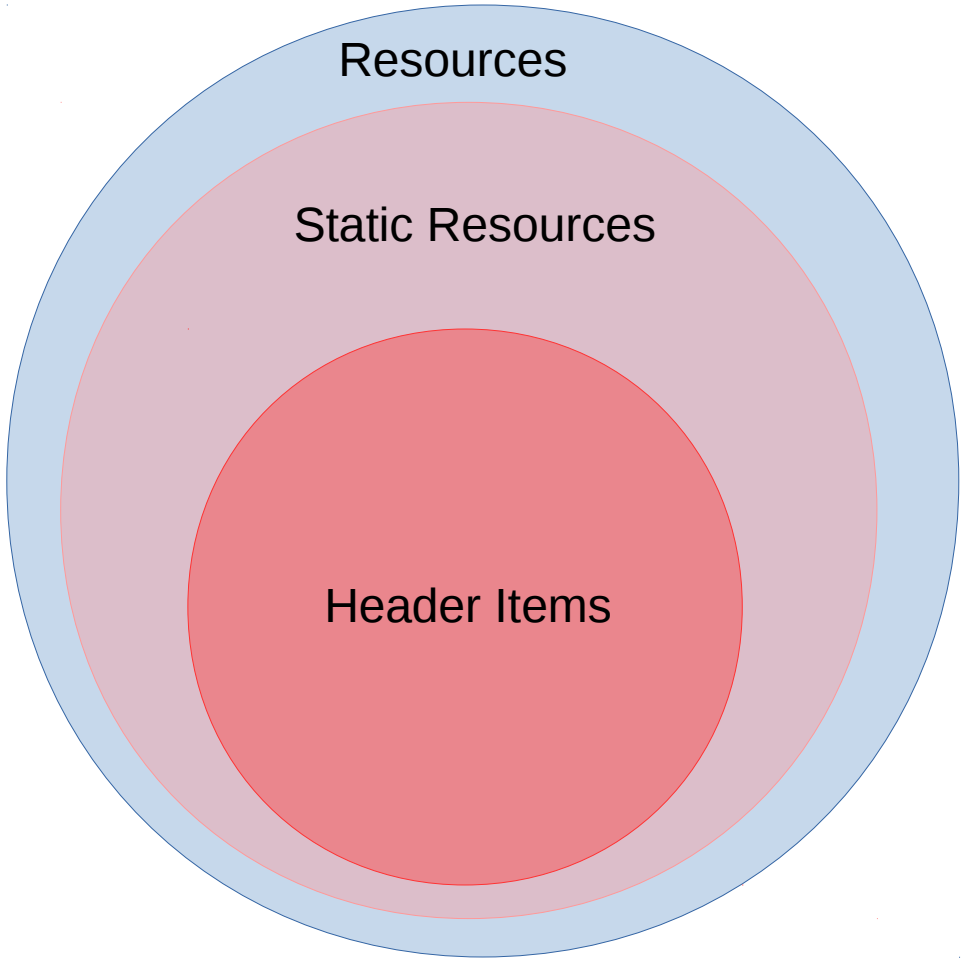
```
class MyPanel extends Panel {

  public void renderHead(IHeaderResponse response) {
    response.render(JavaScriptHeaderItem.forUrl("https://code.jquery.com/"
            + "jquery.min.js"));
    response.render(JavaScriptHeaderItem.forScript("alert('page loaded!');"));
  }
}
```

Static resources

- In general static resources are loaded from 3 possible sources:
  - A generic file from local filesystem
  - A file from classpath (via ClassLoader)
  - An URL
- In Wicket resources are instantiated using a *reference* to them rather than directly. In this way they can be lazy-loaded the first time they are requested.
- Resource references are instances of class *org.apache.wicket.request.resource.ResourceReference*.
- Most of the time static resources are JS or CSS files which can be referred to as *header items.*

Ok, before you fall asleep, let's recap....



Resources: ANY kind of resources. i.e. both dynamic (RSS, dynamic PDF, etc…) and static (JS, CSS, pictures, etc…)

Static Resources: usually loaded from files (JS, CSS, pictures, etc…)

Header Items: those resources that must be placed in the header section (aka <head> tag). JS, CSS, script sections, etc...

Built-in Header Items and dependencies

- **CssHeaderItem**: for CSS content.
- **JavaScriptHeaderItem**: for JavaScript content.
- **StringHeaderItem**: render free text in the header section.

- As we said before, we can declare dependencies on header items and resources:

```
Url jqueyuiUrl = Url.parse("https://ajax.googleapis.com/ajax/libs/jqueryui/" +
                "1.10.2/jquery-ui.min.js");
UrlResourceReference jqueryuiRef = new UrlResourceReference(jqueyuiUrl){
  @Override
  public List<HeaderItem> getDependencies() {
    Application application = Application.get();
    ResourceReference jqueryRef = …;

    return Arrays.asList(JavaScriptHeaderItem.forReference(jqueryRef));
  }
};
JavaScriptReferenceHeaderItem javaScriptHeaderItem =
                                        JavaScriptHeaderItem.forReference(jqueryuiRef);
```

Priority Header Item

- **PriorityHeaderItem**: wraps another header item and ensures that it will have the priority over the other items.

```
Url jqueyuiUrl = Url.parse("https://ajax.googleapis.com/ajax/libs/"
                 + "jqueryui/1.10.2/jquery-ui.min.js");
UrlResourceReference jqueryuiRef = new
                 UrlResourceReference(jqueyuiUrl);
JavaScriptReferenceHeaderItem javaScriptHeaderItem =
                 JavaScriptHeaderItem.forReference(jqueryuiRef);

PriorityHeaderItem item = new
                 PriorityHeaderItem(javaScriptHeaderItem);
```

The item wrapped by PriorityHeaderItem will be contributed before any other non-priority item, including its dependencies.

Header Items for JavaScript

There are also header items meant to work with JavaScript events. In this way we can execute our code only when a specific event occurs.

- **OnDomReadyHeaderItem**: JavaScript code that will be executed after the DOM has been built, but before external files will be loaded.

```
OnDomReadyHeaderItem item = new OnDomReadyHeaderItem(";alert('hello!');");
```

- **OnLoadHeaderItem**: execute JavaScript code after the whole page is loaded.

```
OnLoadHeaderItem item = new OnLoadHeaderItem(";alert('hello!');");
```

- **OnEventHeaderItem**: execute JavaScript code when a specific event is triggered.

```
OnEventHeaderItem item = new OnEventHeaderItem("elementId",
                                "eventName", ";alert('Hello!');");
```

Bundle resources

- To reduce the number of requests to the server, resources can be aggregated in *bundles*. A resource bundle can be declared during application initialization listing all the resources that compose it:

```
@Override
public void init()
{
    getResourceBundles()
         .addJavaScriptBundle(getClass(), "plugins-bundle.js",
             jqueryPlugin1, jqueryPlugin2, jqueryPlugin3
         );
}
```

Now, when one of the resources included in the bundle is requested, the entire bundle is served, i.e. the page will contain the JavaScript entry *plugins-bundle.js* , which includes all the bundle resources.

Header Items for JavaScript

There are also header items meant to work with JavaScript events. In this way we can execute our code only when a specific event occurs.

- **OnDomReadyHeaderItem**: JavaScript code that will be executed after the DOM has been built, but before external files will be loaded.

```
OnDomReadyHeaderItem item = new OnDomReadyHeaderItem(";alert('hello!');");
```

- **OnLoadHeaderItem**: execute JavaScript code after the whole page is loaded.

```
OnLoadHeaderItem item = new OnLoadHeaderItem(";alert('hello!');");
```

- **OnEventHeaderItem**: execute JavaScript code when a specific event is triggered.

```
OnEventHeaderItem item = new OnEventHeaderItem("elementId",
                                "eventName", ";alert('Hello!');");
```

AJAX support

APACHE WICKET™

THE LINUX FOUNDATION

“Transparent” AJAX

Wicket simplifies AJAX development controlling via Java the following basic operations:

- Generate a page unique id for the DOM element of a component.
- Write a callback for an event triggered on the page or on a specific component (with AJAX behaviors).
- Refresh the HTML of a component.
- Execute JavaScript code as response to a specific event.

For the first three operations we won’t write a single line of JavaScript!

Generate unique markup ID

To enhance our components with AJAX the first thing to do is to provide them with a unique id attribute:

```
<span wicket:id="message" id="uniqueId"></span>
```

This can be done invoking method setOutputMarkupId(true). Wicket will take car of generating a page-unique id for the component:

```
component.setOutputMarkupId(true);
```

Alternatively we can force component id to specific value with setMarkupId():

```
component.setMarkupId("myId");
```

Handle AJAX events

- Now we can “ajaxify” components adding AJAX *behaviors*. In Wicket a behaviors are quite like *plug-ins* that can enrich a component with new features.
- For example *org.apache.wicket.ajax.AjaxEventBehavior* provides the means to handle an event on server side via AJAX:

Java Code

```
Label label = new Label("label","Hello!");
label.setOutputMarkupId(true);

label.add(new AjaxEventBehavior("click") {
  @Override
  protected void onEvent(AjaxRequestTarget
                   target) {
    //Do my stuff...
  }
});
```

HTML Template

```
<html>
<body>
    <span wicket:id="message">
      Message goes here
    </span>
</body>
</html>
```

AjaxRequestTarget is the main entity for AJAX development.

Refresh component HTML and add JavaScript to AJAX response

- A common parameter for AJAX handler is *org.apache.wicket.ajax.AjaxRequestTarget*, which can be used to refresh component HTML:

```
protected void onEvent(AjaxRequestTarget
                       target) {
    target.add(panel);
}
```

- *AjaxRequestTarget* can be used also to enrich AJAX response with JavaScript code and header items:

```
protected void onEvent(AjaxRequestTarget
                       target) {
   target.appendJavaScript(";alert('hello!');");
   target.getHeaderResponse().render(headerItem);
}
```

Built-in AJAX components and behaviors

A number of ready-to-use components and behaviors are provided out of the box:

- AjaxLink
- AjaxButton
- AjaxCheckBox
- AutoCompleteTextField
- AjaxEventBehavior
- AjaxFormSubmitBehavior
- AbstractAjaxTimerBehavior
- ….

More examples at http://examples7x.wicket.apache.org/ajax/

Java lambdas: a match made in heaven

- Java 8 lambdas are quite suited for writing callback code, which is what we do to handle AJAX events.
- With the incoming Wicket 8 an AJAX link can be written leveraging lambdas in the following way:

```
AjaxLink.onClick("link", target -> target.add(component));
```

- Like it? Don’t miss Martijn’s talk!

Testing with Wicket

Test in isolation

- Test Driven Development (and unit testing) has become a fundamental activity in our everyday-job. Wicket offers a rich set of helper classes that allows us to test our applications in isolation using just JUnit.

- With “just JUnit” we mean:

1) We don’t need to have a running server
2) We don’t need to run tests for a specific browser (like we do with Karma)
3) No additional library required, just Wicket and JUnit (no need of browser automation tools like Selenium)

Test in isolation

- The central class in a Wicket testing is *org.apache.wicket.util.tester.WicketTester*. This utility class provides a set of methods to render a component, click links, check page content, etc...

```
public class TestHomePage {
    private WicketTester tester;

    @Before
    public void setUp() {
        tester = new WicketTester(new WicketApplication());
    }

    @Test
    public void testHomePageLink() {
            //start and render the test page
            tester.startPage(HomePage.class);
            //assert rendered page class
            tester.assertRenderedPage(HomePage.class);
            //move to an application link
            tester.executeUrl("./foo/bar");
            //test expected page for link
            tester.assertRenderedPage(AnotherHomePage.class);
    }
}
```

Testing the response

- *WicketTester* allows us to access to the last response generated during testing with method *getLastResponse*. Utility class *Mock-HttpServletResponse* is returned to extract informations from mocked request.

```
String responseContent = tester.getLastResponse().getDocument();
tester.assertContains("regExp");
```

- Resulting markup can be tested at tag-level with *TagTester*:

Testing the response

Response content:

```
<html xmlns:wicket="http://wicket.apache.org">
    <head>
        <meta charset="utf-8" />
        <title></title>
    </head>
    <body>
        <span class="myClass"></span>
        <div class="myClass"></div>
    </body>
</html>
```

JUnit code:

```
String responseContent = tester.getLastResponse().getDocument();
//look for a tag with 'class="myClass"'
TagTester tagTester = TagTester.createTagByAttribute(responseTxt,
                                "class", "myClass");

assertEquals("span", tagTester.getName());
List<TagTester> tagTesterList = TagTester.createTagsByAttribute(responseTxt,
                            "class", "myClass", false);

assertEquals(2, tagTesterList.size());
```

Testing AJAX events

- AJAX components can be tested as well “triggering” the JavaScript event they handle:

Page Code

```
Label label = new Label("label", "Hello World!");
Label otherLabel = new Label("otherLabel", "hola!");
label.setOutputMarkupId(true);

label.add(new AjaxEventBehavior("click") {
  @Override
  protected void onEvent(AjaxRequestTarget target) {
    target.add(otherLabel);
  }
});
```

Test Code

```
//simulate an AJAX "click" event
tester.executeAjaxEvent("label", "click");

//test other assertions…
```

Testing AJAX response

- The AJAX response can be tested with *WicketTester* to ensure that a specific component has been added (i.e. we want to refresh its markup):

Page Code

```
Label label = new Label("label", "Hello World!");
Label otherLabel = new Label("otherLabel", "hola!");
label.setOutputMarkupId(true);

label.add(new AjaxEventBehavior("click") {
  @Override
  protected void onEvent(AjaxRequestTarget target) {
    target.add(otherLabel);
  }
});
```

Test Code

```
//simulate an AJAX "click" event
tester.executeAjaxEvent("label", "click");
//test if AjaxRequestTarget contains a component (using its path)
tester.assertComponentOnAjaxResponse("otherLabel");
```

Testing AJAX behaviors

- AJAX behaviors can also be tested in isolation, relying only on *WicketTester:*

Test Code

```
AjaxFormComponentUpdatingBehavior ajaxBehavior =
                new AjaxFormComponentUpdatingBehavior("change"){
            @Override
            protected void onUpdate(AjaxRequestTarget target) {
                //...
            }
};

component.add(ajaxBehavior);

//execute AJAX behavior, i.e. onUpdate will be invoked
tester.executeBehavior(ajaxBehavior);
```

Other test goodies

WicketTester offers many more utilities for unit testing:

- Check component status (*assertEnabled*, *assertDisabled, assetVisible, assertInvisible*)
- Check component's model value (*assertModelValue*)
- Test forms with *FormTester*:

```
FormTester formTester = tester.newFormTester("form");
//set credentials
formTester.setValue("username", username);
formTester.setValue("password", password);
//submit form
formTester.submit();
```

- ...

Summary and references

- We had a “journey” through the life of Wicket.
- We have seen how it evolved in the very last years.
- As for any other Open Source project, the health of the community is fundamental.
- We have seen some of the most appealing features, still there is lot more to discover!
- Learn more and keep in contact with us!
  - Main site: http://wicket.apache.org/
  - Tweeter account: https://twitter.com/apache_wicket/
  - User guide: http://wicket.apache.org/learn/#guide
  - User guide live examples: http://examples-wickettutorial.rhcloud.com/

APACHECON Europe

Thank you!

THE LINUX FOUNDATION