

APACHE



WICKET

# APACHE WICKET

Jan Kriesten

Java User Group Hamburg, 17.10.2007



# sign of renitence

- ▶ Werbeagentur
- ▶ Standorte Nürnberg + Hamburg
- ▶ Kompetenz in Grafik und Technik
- ▶ Kunden

SIEMENS

KATTUS



warner | music | group

AKEMI®

VDW

VERBAND DEUTSCHER WERBEFILMPRODUZENTEN E.V.

RECHTSANWALTSKAMMER NÜRNBERG





# Agenda

- ▶ **Was ist Wicket?**
- ▶ Aufbau und Konzept
- ▶ Komponenten by Example
- ▶ Wicket testen
- ▶ Ausblick und Resümee



# Was ist Wicket?

- ▶ Die Idee hinter Wicket
- ▶ Features
- ▶ Vergleich mit anderen Frameworks



# Die Idee hinter Wicket

- ▶ leicht (einfach – konsistent – intuitiv)
- ▶ wiederverwendbar
- ▶ sauber
- ▶ sicher
- ▶ effizient und skalierbar
- ▶ vollständig



# Features (I)

- ▶ Objektorientiertes Seiten- und Komponenten-Modell  
Java-Objekte: Kapselung, Vererbung, Events
- ▶ Separation of Concerns  
Keine Logik im HTML – keine neuen Tags
- ▶ Status-Management
- ▶ Sicher  
Keine sensiblen Informationen in URLs
- ▶ Integration  
Spring – Guice – OSGi
- ▶ Clusterfähig



## Features (II)

- ▶ Komponenten – ready to use
  - sortier-, filter- und navigierbare Tabellen
  - Datepicker, Rich-Text-Editoren, Google Maps, etc.
  - Tabs, Breadcrumb, Trees, Wizard
- ▶ Ajax-Unterstützung und -Komponenten
- ▶ Header-Contribution  
JavaScript & CSS
- ▶ URL mounting  
Bookmarking, lesbare URLs



## Features (III)

- ▶ Wiederverwendbare Komponenten
- ▶ Geschachtelte Formulare
- ▶ Back-Button-Support  
Versionierung von Komponenten
- ▶ Einfache Lokalisierung  
HTML, Images und Resource-Strings anpaßbar
- ▶ Anpassung von HTML-Attributen aus Java
- ▶ Unit-Tests



# Vergleich mit Frameworks

- ▶ Traditionelle Frameworks  
(Struts, WebWork, Spring MVC, etc.)
  - Abstraktion basiert auf HTTP-Request-Mechanismus
  - Zusätzliche Technologie für HTML-Ausgabe  
(JSP/JSTL, etc.)
  - Kein Status-Management
  - Aufwändige Konfiguration (XML)



# Vergleich mit Frameworks

- ▶ Komponenten-orientierte Frameworks (JSF, Tapestry, etc.)
  - Aufwändig zu lernen
  - Komplexe Konfiguration (XML)
  - Ggf. zusätzliche Technologien (JSP, Facelets, Seam, etc.)
  - Wiederverwendung von Komponenten z.T. aufwändig
  - teils mit „domain specific language“ oder eingeschränktem Java-Sprachumfang



# Agenda

- ▶ Was ist Wicket?
- ▶ **Aufbau und Konzept**
- ▶ Komponenten by Example
- ▶ Wicket testen
- ▶ Ausblick und Resümee



# Aufbau und Konzept

- ▶ **Application**
- ▶ Session
- ▶ RequestCycle
- ▶ Components
- ▶ Behaviors
- ▶ Models



# Application

- ▶ Startpunkt der Web-Applikation
- ▶ Konfiguration
  - Homepage
  - Ausgabe von Wicket-Tags
  - File-Monitor für Änderungen
- ▶ Factories für
  - Session, RequestCycle, Security, etc.



# Application

## ► Konfiguration in der web.xml:

```
<filter>
  <filter-name>WicketFilter</filter-name>
  <filter-class>org.apache.wicket.protocol.http.WicketFilter</filter-class>
  <init-param>
    <param-name>applicationClassName</param-name>
    <param-value>wicket.TestApplication</param-value>
  </init-param>
  <init-param>
    <param-name>configuration</param-name>
    <param-value>development</param-value>
  </init-param>
</filter>

<filter-mapping>
  <filter-name>WicketFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```



# Aufbau und Konzept

- ▶ Application
- ▶ **Session**
- ▶ RequestCycle
- ▶ Components
- ▶ Behaviors
- ▶ Models



# Session

- ▶ Abstraktion einer User Session
- ▶ Speicherung in HttpSession
- ▶ Speichern von Session-spezifischen Daten
  - Locale, Browser-Info (Hersteller, Version)
  - Eigene Daten (angemeldeter Benutzer, Warenkorb)
  - Page-History (Back-Button-Unterstützung)



# Session

```
class MySession extends WebSession {  
    private ShoppingCart cart;  
    public ShoppingCart getCart() { ... }  
    public void setCart( ShoppingCart cart ) { ... }  
}
```

```
MySession.get().setCart( new ShoppingCart() );
```

```
...
```

```
ShoppingCart cart = MySession.get().getCart();  
cart.add( quantity, product );
```



# Aufbau und Konzept

- ▶ Application
- ▶ Session
- ▶ **RequestCycle**
- ▶ Components
- ▶ Behaviors
- ▶ Models



# RequestCycle

▶ Verarbeitet den Request:

- (1) Erstellen des RequestCycleProcessor-Objekts
- (2) Dekodieren des Requests
- (3) Request-Ziel identifizieren (Page, Component)
- (4) Event abarbeiten (onClick, onSubmit)
- (5) Respond (Page, Component, Image, PDF, ...)
- (6) Aufräumen



# RequestCycle

- ▶ Requests können folgender Art sein:
  - Stateful
    - gebunden an die User-Session
    - nicht als Bookmark nutzbar
  - Stateless
    - nicht zwingend an die User-Session gebunden
    - als Bookmark nutzbar



# Aufbau und Konzept

- ▶ Application
- ▶ Session
- ▶ RequestCycle
- ▶ **Components**
- ▶ Behaviors
- ▶ Models



# Components

- ▶ Komponenten kapseln die Umsetzung von Logik und Ausgabe
- ▶ Komponenten können Events erhalten (onClick, onSubmit)
- ▶ Komponenten „wissen“ wie und wo sie sich selbst darstellen müssen
- ▶ Komponenten können eigenes Markup haben (Page, Panel, Border)



# Components

- ▶ Referenziert wird eine Komponente im Markup über die `wicket:id`:

HTML:

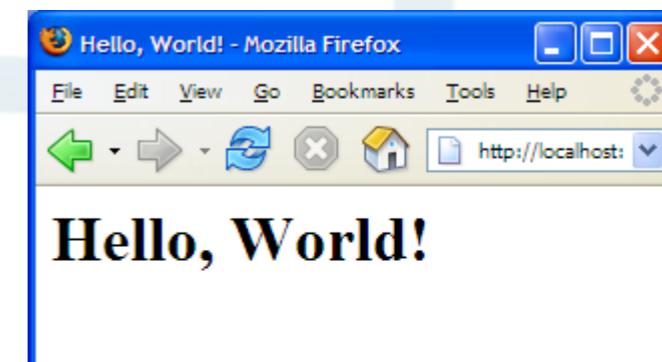
```
<h1 wicket:id="msg">Nachricht</h1>
```

JAVA:

```
new Label( "msg", "Hello, World" );
```

Ergebnis:

```
<h1>Hello, World</h1>
```





# Components – eine Seite

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:wicket="http://wicket.apache.org">
  <head>
    <title>Simple Page</title>
  </head>
  <body>
    <h1 wicket:id="msg">Nachricht</h1>
  </body>
</html>
```

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

public class HelloWorldPage extends WebPage
{
    public HelloWorldPage()
    {
        super();
        add( new Label( "msg", "Hello, World!" ) );
    }
}
```



# Aufbau und Konzept

- ▶ Application
- ▶ Session
- ▶ RequestCycle
- ▶ Components
- ▶ **Behaviors**
- ▶ Models



# Behaviors

- ▶ Schnittstellen zu Komponenten
- ▶ Behaviors können Markup verändern

```
item.add( new AbstractBehavior()
{
    public void onComponentTag( Component component, ComponentTag tag)
    {
        String css = (((Item)component).getIndex() % 2 == 0) ? "even" : "odd";
        tag.put("class", css);
    }
});
```

Ergebnis:

```
<tr class="odd">...</tr>
<tr class="even">...</tr>
```



# Behaviors

- ▶ Verändern/hinzufügen von Attributen im Markup
- ▶ Header-Contribution (CSS, Javascript-Libraries)  
(Interface [IHeaderContributor](#))
- ▶ Hinzufügen von Javascript-Events
- ▶ Hinzufügen von Ajax-Verhalten

```
component.add(  
    new AjaxSelfUpdatingBehavior(  
        Duration.seconds( 1 ) ) );
```



# Aufbau und Konzept

- ▶ Application
- ▶ Session
- ▶ RequestCycle
- ▶ Components
- ▶ Behaviors
- ▶ **Models**



# Models

- ▶ Models verbinden Komponenten mit POJO's

```
new Label( "name", model );
```

PropertyModel

```
public class Person  
{  
    String name;  
    String ort;  
    ...  
}
```



# Models

▶ Java mangelt es an „lazy binding“

- wird nicht aktualisiert:

```
new TextField( "txt", person.getName() );
```

- kann Null-Pointer-Exception (NPE) geben:

```
new Label( "strasse", person.getAddress().getStreet() );
```

▶ Lösung: Zugriff á la OGNL/EL

- PropertyModel: `new PropertyModel( person, "address.street" );`

- Null wird bei Aktualisierung aufgefangen



# Models

- ▶ Standard-Models für verschiedene Zwecke
  - CompoundPropertyModel  
„vererbbares“ Model
  - StringResourceModel  
zur Lokalisierung
  - LoadableDetachableModel  
Temporäres, transientes Model



# Agenda

- ▶ Was ist Wicket?
- ▶ Aufbau und Konzept
- ▶ **Komponenten by Example**
- ▶ Wicket testen
- ▶ Ausblick und Resümee



# Komponenten by Example

- ▶ Eigene Komponente erstellen:
  - „extends“
  - Fertig.
- ▶ Komponenten erweitern `org.apache.wicket.Component` (direkt oder indirekt)
- ▶ Komponenten sind verfügbar über den Anwendungs-Classpath.



# Komponenten by Example

- ▶ `org.apache.wicket.markup.html.panel.Panel`
  - Panels ermöglichen Gruppierung von Objekten
  - Panels haben eigenes Markup
  - Auf Seiten ersetzbar durch andere Komponenten
  - HeaderComponent
  - Panels können beliebige andere Komponenten beinhalten (auch Panels)

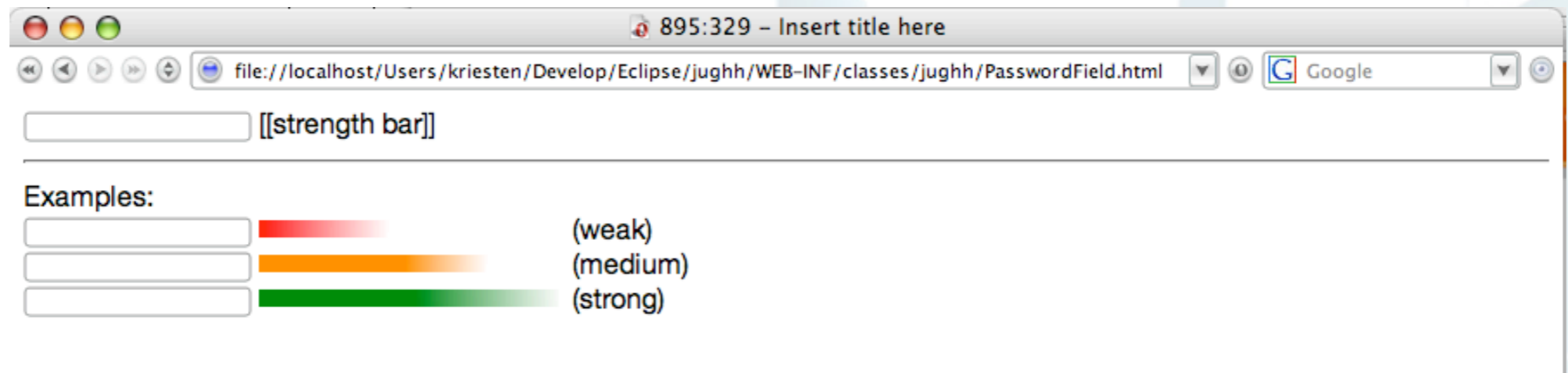


# Komponenten by Example

```

<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <title>Insert title here</title>
  <wicket:head>
    <wicket:link>
      <link rel="stylesheet" type="text/css" href="res/PasswordField.css"/>
    </wicket:link>
  </wicket:head>
</head>
<body>
  <wicket:panel>
    <input wicket:id="password" type="password" /> <span wicket:id="strength">[[strength bar]]</span>
  </wicket:panel>
  <div>
  <hr/>
  Examples:<br/>
  <input type="password" /> <span class="weak">&nbsp;</span> (weak)<br/>
  <input type="password" /> <span class="medium">&nbsp;</span> (medium)<br/>
  <input type="password" /> <span class="strong">&nbsp;</span> (strong)<br/>
  </div>
</body>
</html>

```





# Komponenten by Example

```
public class PasswordField
    extends Panel
{
    public final static String WEAK = "weak";
    public final static String MEDIUM = "medium";
    public final static String STRONG = "strong";

    public PasswordField( String id, IModel model ) {
        super( id, model );
        PasswordTextField passwordTextField = new PasswordTextField( "password", model );
        add( passwordTextField );

        final Label strength = new Label( "strength", " " );
        add( strength );
        strength.add( new AttributeModifier( "class", true, new Model() {
            private static final long serialVersionUID = 1L;

            public Object getObject( ) {
                return getPasswordStrength( (String) PasswordField.this.getModelObject() );
            }
        } ) );

        strength.setOutputMarkupId( true );
        passwordTextField.add( new OnKeyPausedAjaxBehavior() {
            private static final long serialVersionUID = 1L;

            protected void onUpdate( AjaxRequestTarget target ) {
                target.addComponent( strength );
            }
        } );
    }
}
```



# Komponenten by Example

```
public class TestPage
    extends WebPage
{
    private static final long serialVersionUID = 1L;

    private static class User
        implements Serializable
    {
        private static final long serialVersionUID = 1L;
        private String password;

        public String getPassword( )
        {
            return password;
        }

        public void setPassword( String password )
        {
            this.password = password;
        }
    }

    private User user = new User();

    public TestPage()
    {
        Form form = new Form( "form" );
        form.add( new PasswordField( "password", new PropertyModel( user, "password" ) ) );
        add( form );
    }
}
```



# Reusability by Example

- ▶ JAR erstellen mit
  - HTML, Javascript, Bilder, CSS
  - class-Datei
- ▶ JAR dem classpath hinzufügen
- ▶ Fertig.



# Agenda

- ▶ Was ist Wicket?
- ▶ Aufbau und Konzept
- ▶ Komponenten by Example
- ▶ **Wicket testen**
- ▶ Ausblick und Resümee



# Wicket testen

## ▶ WicketTester: Unit-Tests ohne Servlet-Container

```
public class MyPage extends WebPage {
    public MyPage() {
        add(new Label("myMessage", "Hello!"));
        add(new Link("toYourPage") {
            public void onClick() { setResponsePage(new YourPage("Hi!")); }
        });
    }
}

private WicketTester tester;
public void setUp() { tester = new WicketTester(); }
public void testRenderMyPage() {
    tester.startPage(MyPage.class);           // Seite starten und rendern
    tester.assertRenderedPage(MyPage.class);  // Class prüfen
    tester.assertLabel("myMessage", "Hello!"); // Label prüfen
    tester.clickLink("toYourPage");           // klick simulieren
    ...
}
```



# Agenda

- ▶ Was ist Wicket?
- ▶ Aufbau und Konzept
- ▶ Komponenten by Example
- ▶ Wicket testen
- ▶ **Ausblick und Resümee**



# Ausblick und Resümee

- ▶ Weitere Entwicklungen
- ▶ Resümee
- ▶ Literatur
- ▶ Interaktiv



# Weitere Entwicklungen

- ▶ Unterstützung von Java Generics
- ▶ Validierung auf Client-Seite (GWT-Javascript-Generierung)
- ▶ Ajax-Back-Button-Unterstützung
- ▶ Session-Inspektor zur Unterstützung bei der Fehlersuche



# Resümee

- ▶ Wicket ist gut zu erlernen
- ▶ Vielzahl fertiger und erweiterbarer Komponenten
- ▶ Hohe Produktivität bei der Umsetzung von Projekten
- ▶ Gute Unterstützung durch aktive Community
- ▶ „Be friends with your boss again!“



# Literatur – Bücher

- ▶ Pro Wicket, Karthik Gurumurthy, Springer-Verlag
  - erschienen 2006
  - behandelt Wicket 1.2
- ▶ Wicket in Action, Martijn Dashorst + Eelco Hillenius, Manning
  - erscheint 2008 – MEAP verfügbar
  - geschrieben aus der Praxis
  - Autoren entwickeln Wicket mit
  - Martijn + Eelco in der Community sehr aktiv



# Literatur – Online

- ▶ Wicket-Homepage:  
Erste Anlaufstelle für Informationen  
<http://wicket.apache.org/>
- ▶ WIKI:  
Etwas unübersichtlich, aber viele nützliche Hinweise  
<http://cwiki.apache.org/WICKET/>
- ▶ Wicket-Stuff:  
Interessante Projekte (z.B. Google Map-Integration),  
Komponenten-Übersicht/-Beispiele  
<http://wicketstuff.org/confluence/display/STUFFWIKI/Wiki>  
<http://wicketstuff.org/wicket13/>



# Interaktiv

- ▶ User-Mailinglist  
Archiv: <http://www.nabble.com/Apache-Wicket-f13974.html>  
sehr aktiv und schnelle Reaktion auf Fragen
- ▶ IRC: ##wicket@irc.freenode.net  
sehr aktiv, viele Wicket-Entwickler sind hilfreich zur Stelle
- ▶ Blogs  
Entwickler-Blogs mit interessanten Einsichten