FrOSCon
Free and Open Source Software Conference

anderScore
trust in competence

Einführung in Apache Wicket

20.08.2016

Daniel Krämer

Inhalt

- **Vorstellung und Organisation**
- **Einführung**
  - Grundlagen
  - Installation
  - Aufbau einer WebApp
- **Architektur**
  - Models
  - Views
  - Formulare
  - Tests
- **Einordnung**

Nachfolgendes Event mit Ralf Bommersbach:
Wicket – Formulareingaben validieren & testen

Vorstellung

anderScore trust in competence

Daniel Krämer (M.Sc. C.S.)

- Absolvent der H-BRS
- Software Engineer
- Schwerpunkte
  - Java
  - Web Engineering
  - Software-Architektur
- Organisator des DevRooms Java

Organisation

Unterlagen

- Präsentation

https://github.com/anderscore-gmbh/FrOSCon-2016/tree/wicket-basics

Einführung

Einführung

anderScore trust in competence

Grundlagen

- Komponentenorientiertes Java-Web-Framework
- Veröffentlichung: 2004
- Aktuelle Version: 7.4.0 (Juli 2016)
- Apache-Lizenz (Open Source)

https://wicket.apache.org

Philosophie

- Serverseitige Verarbeitung
  - Session hält Komponentenbaum
  - Loading und Detaching
  - HTTP Roundtrip
- Strikte Trennung von GUI und Logik
  - HTML + CSS
  - Java
- OO-Prinzipien
  - Abstraktion
  - Vererbung
  - Kapselung
  - Wiederverwendung

Installation

Installation

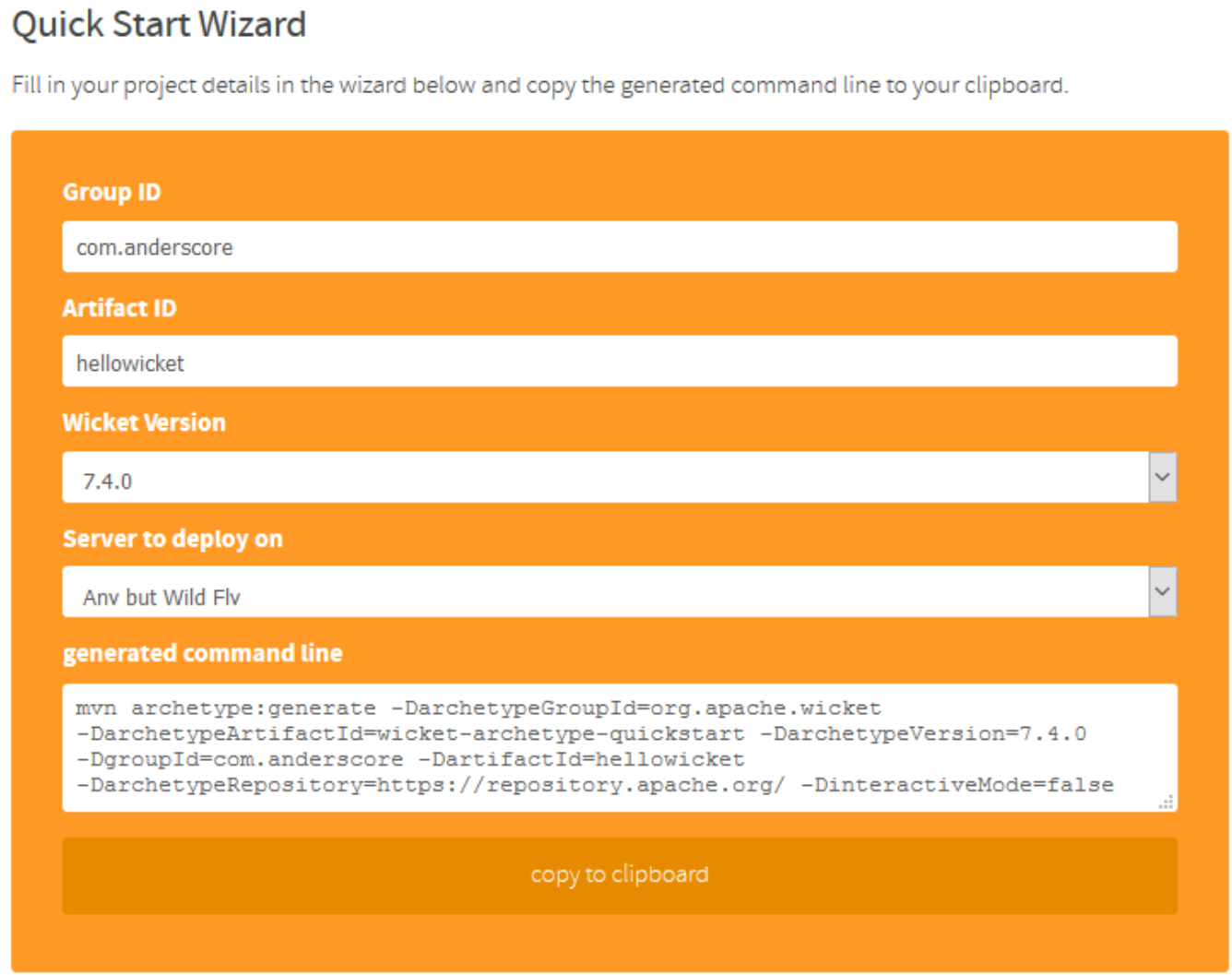
anderScore trust in competence

Empfehlung: Wicket Quick Start

- Generierung eines Maven-Projektes
  - Lauffähige Wicket-Applikation
  - Starter für Server Jetty
- Import in IDE eurer Wahl

http://wicket.apache.org/start/quickstart.html

Installation



Aufbau einer WebApp

Aufbau einer WebApp

anderScore trust in competence

Grundlegende Bestandteile

- Application-Klasse
  - Konfiguration und Initialisierung der Anwendung
  - Festlegung der Home Page
- Web Pages
  - Ein HTML View + eine Klasse pro Page
  - HTML: ausschließlich Markup
  - Java: Anwendungslogik

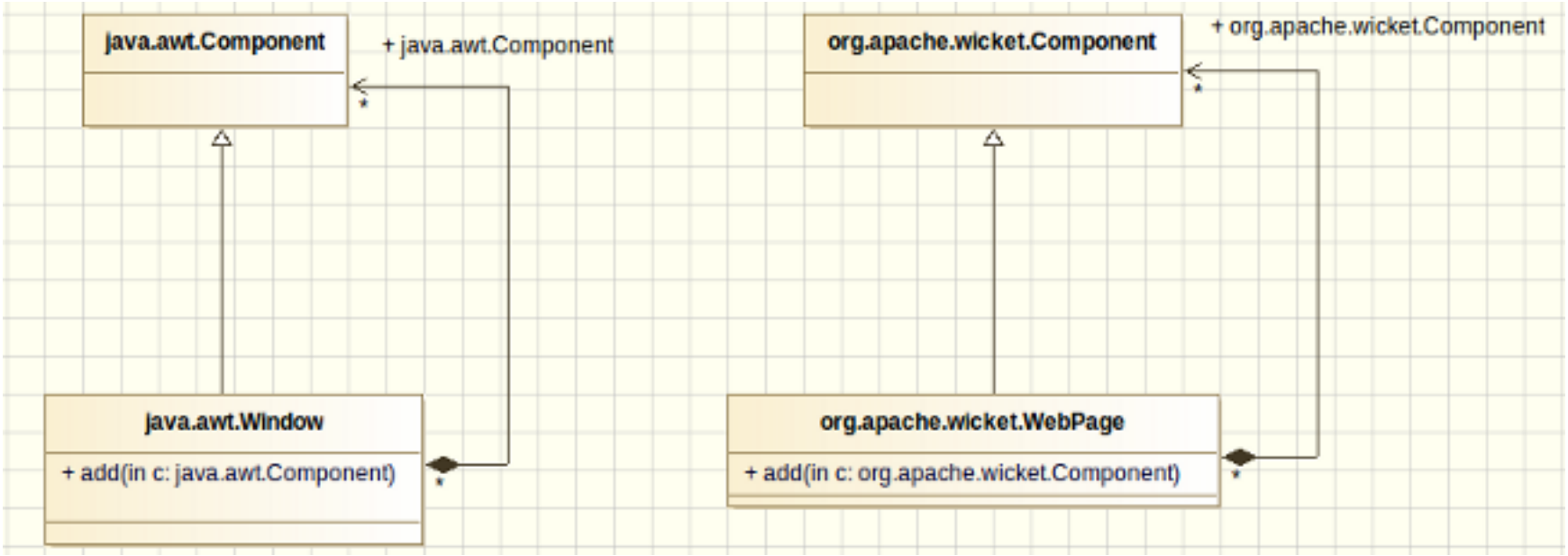
Aufbau einer WebApp

anderScore trust in competence

Grundlegende Bestandteile

- Komponenten
  - Bestandteile der UI (z.B. Formularfelder)
  - Einfügung in andere Komponenten und Web Pages
- Models
  - Kapseln fachliche Daten
  - Bindung an UI-Komponenten
- WEB.xml
  - WicketFilter

Vergleich mit AWT



Architektur

Architektur

anderScore trust in competence

Komponenten

- Wiederverwendbare UI-Bausteine
- HTML + Java
- Vorgefertigte Komponenten
  - Labels
  - Forms
  - Links
  - Buttons

Architektur

anderScore trust in competence

Pages

- HTML + Java-Klasse im selben Package
- Analoger Aufbau eines Komponentenbaums
  - HTML: Platzhalter mit **wicket:id**
  - Java: Programmatische Einfügung im Konstruktor
- Navigation über Link-Komponenten
  - Definition eines onClick-Handlers in Java

Pages

```
<!DOCTYPE html>
<html xmlns:wicket="http://wicket.apache.org">

<head>
    <title>Wicket Tutorial</title>
</head>

<body>

<p>Anbieter des Tutorials: <span wicket:id="company"></span></p>
<p><a wicket:id="start">Tutorial beginnen</a></p>

</body>

</html>
```

Pages

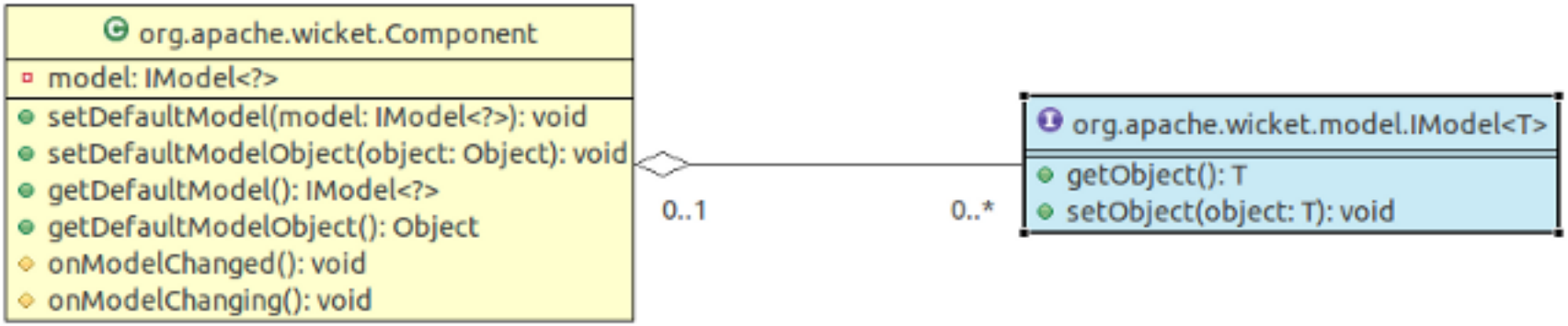
```
public class HomePage extends WebPage {
   private static final long serialVersionUID = 1L;

   public HomePage(final PageParameters parameters) {
      super(parameters);

      add(new Label("company", "anderScore"));
      add(new Link("start"){
         @Override
         public void onClick() {
            // Weiterleitung zu einer anderen Page
            setResponsePage(StartPage.class);
         }
      });

      // Weitere Komponenten einfügen...
    }
}
```

Models

Models

Grundlagen

- Kapselung von Datenobjekten (Strings, POJOs, …)
- **Data Binding**: UI-Komponenten ↔ fachliche Daten
- Höchstens ein Model pro Komponente



Models

anderScore trust in competence

Beispiel: PropertyModel

- Kapselt einzelne Property eines Fachobjektes
- Vorteile gegenüber statischen Werten
  - Immer aktueller Wert in UI
  - Aktualisierung der Property in Objekt

```
Person person = new Person();
//Personendaten laden...

Label label = new Label("name", new PropertyModel(person, "name"));
```

Views

Views

anderScore trust in competence

Labels

- Dynamische Ausgabe von Zeichenketten
- Convenience-Konstruktor ohne Model

- HTML

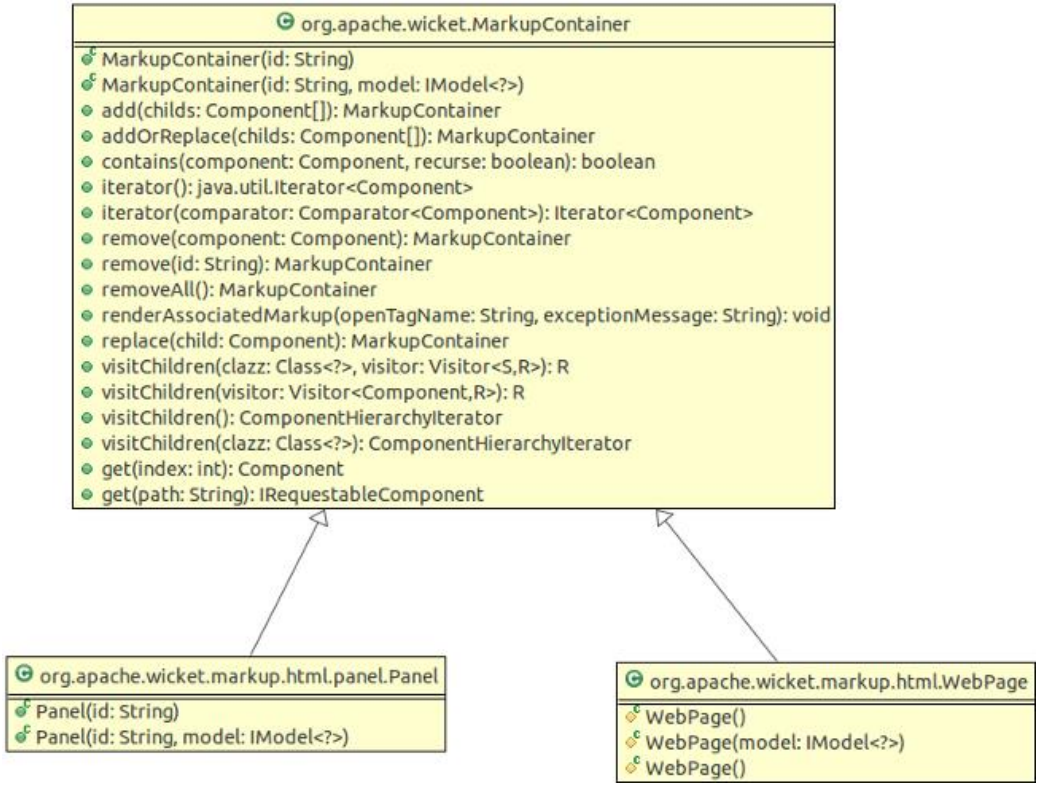
```
<p>Anbieter des Tutorials: <span wicket:id="company"></span></p>
```

- Java

```
add(new Label("company", "anderScore"));
```

Panels

- Wiederverwendbare Container für Komponenten
- HTML + Java



Panels

- HTML: Panel

```
<html>

<head></head>

<body>

<wicket:panel>
   <!-- Diverse Labels. Äußeres Gerüst wird ignoriert! -->
</wicket:panel>

</body>

</html>
```

Panels

- Java: Panel

```
public class PersonPanel extends Panel {

    public PersonPanel(String id, IModel<Person> person) {
        super(id);
        setDefaultModel(new CompoundPropertyModel(person));

        add(new Label("name"));
        add(new Label("surname"));
        add(new Label("address"));
        add(new Label("email"));
        add(new Label("spouse.name"))
    }
}
```

Panels

- Java: Page

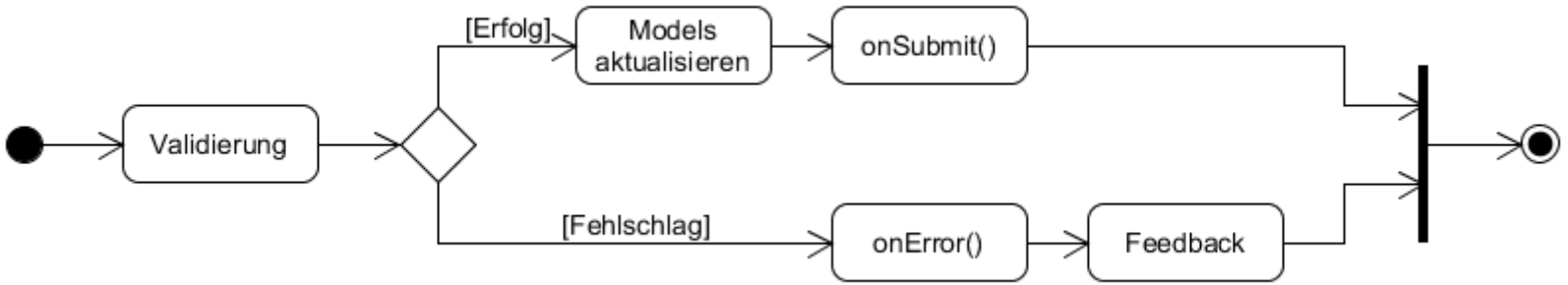
```
public class PersonPage extends WebPage {

    public PersonPage() {
        Person john = new Person("John", "Doe");
        IModel<Person> person = new Model<>(john);

        add(new PersonPanel("person", person));
    }
}
```

Formulare

Grundlagen

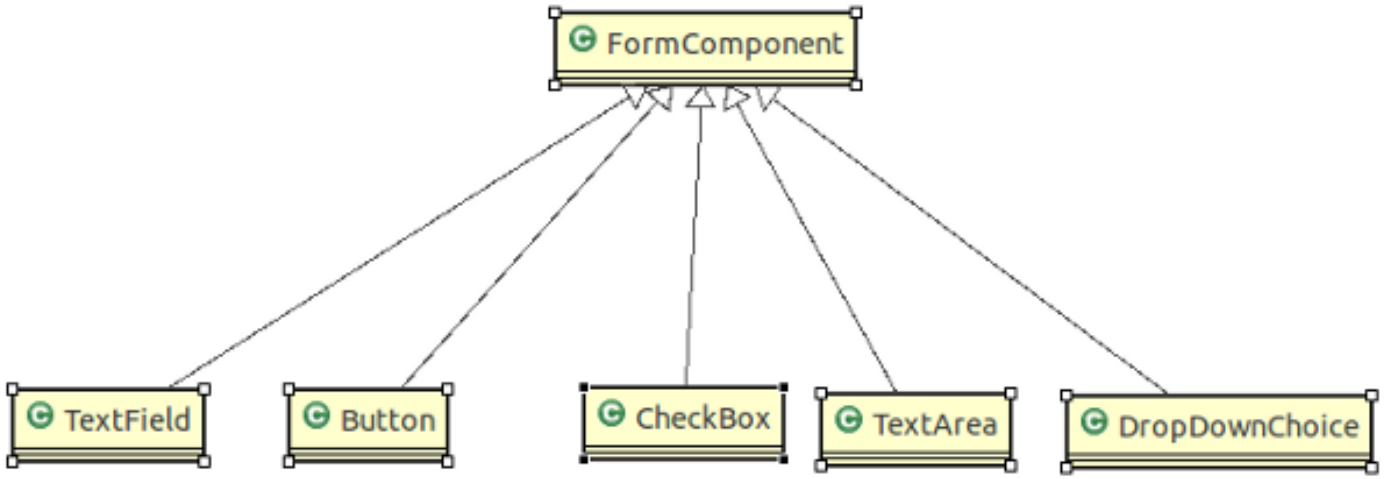
- Eigene Komponenten
- Handler auf Serverseite
  - onSubmit()
  - onError()

- Verarbeitung



Grundlagen

- Vorgefertigte Eingabekomponenten
  - Textfelder
  - CheckBoxen
  - DropDowns
  - Buttons
  - …



Grundlagen

- HTML: Page

```
<html>

<head><title>Person</title></head>

<body>

<form id="personForm" method="get" wicket:id="personForm">

  <span>Name: </span><input wicket:id="name" type="text" id="name"/>
  <span>Nachname: </span><input wicket:id="surname" type="text" id="surname"/>
  <input type="submit" name="Save" value="Speichern"/>

</form>

</body>

</html>
```

Grundlagen

- Java: Page

```
public class PersonPage extends WebPage {

    public PersonPage() {
        super();

        add(new PersonForm("personForm"));
    }
}
```

Grundlagen

- Java: Formular

```
public class PersonForm extends Form {

    private TextField nameField;
    private TextField surnameField;

    public PersonForm(String id) {
        super(id);

        nameField = new TextField("name", Model.of(""));
        surnameField = new TextField("surname", Model.of(""));

        add(nameField);
        add(surnameField);
    }

    public final void onSubmit() {
        String name = (String) nameField.getDefaultModelObject();
        String surname = (String) surnameField.getDefaultModelObject();

        System.out.println("Hallo " + name + " " + surname);
    }
}
```

Ausblick: Validierung

- Interface *IValidator*
- Vorgefertigte Validatoren
  - EmailAddressValidator
  - URLValidator
  - DateValidator
  - RangeValidator
  - …
- Eigene Validatoren

- **Mehr zu diesem Thema im nachfolgenden Beitrag von Ralf Bommersbach!**

Tests

Tests

anderScore trust in competence

Allgemeines

- Fokussierung des Frameworks auf Java
  - JUnit
  - TestNG
  - Mockito
  - …
- Unterstützung durch Wicket
  - WicketTester
  - FormTester
  - TagTester

Tests

anderScore trust in competence

WicketTester

- Utility-Klasse zur Unterstützung von Tests
- Kommunikation mit dem Framework
  - Rendern von Pages und Komponenten
  - Prüfung des Status von Komponenten
  - Isoliertes Testen von Komponenten
  - Klicken auf Links
  - …

WicketTester

```
public class TestHomePage {

    private WicketTester tester;

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

    @Test
    public void testComponents(){
        // Rendern der Homepage
        tester.startPage(HomePage.class);
        // Rendering der Homepage überprüfen
        tester.assertRenderedPage(HomePage.class);
        // Überprüfung auf Vorhandensein von Label
        tester.assertLabel("label", "First label");
        // Klick auf Link simulieren
        tester.clickLink("reload");
        // Überprüfung von Vorhandensein von Label
        tester.assertLabel("label", "Second label");
        // Überprüfung der Aktivierung eines DatePickers
        tester.assertEnabled("form:datepicker");
    }
}
```

Einordnung

Einordnung

Vergangenheit und Gegenwart

- 12 Jahre Bewährung in der Praxis
- Aktive Weiterentwicklung durch Community
- Nahezu feature-complete
- Weiterhin sehr nachgefragt
  - Banken
  - Versicherungen

Einordnung

Zukunft

- Neue Technologien: Java 8, HTTP/2, …
- Optimierung
- Gegen aktuellen Trend
  - Serverseitiger State
  - Schwergewichtig

Einordnung

Einsatzfelder

- Schwerpunkt auf serverseitiger Verarbeitung
- Anwendungslandschaften im Java-Umfeld
- Migrationen **zu** Java
- … und **von** Java!

Zusammenfassung

- Komponentenorientiertes Java-Web-Framework
- Starker Fokus auf OO-Prinzipien
- Strikte Trennung von GUI und Logik
- Serverseitige Verarbeitung
- Data Binding durch Models

- Praxiserprobt und nahezu feature-complete
- Alternative zu aktuellen FE-Technologien

Vielen Dank für eure Aufmerksamkeit!