3.– 6. September 2012
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Spielend leicht

Einführung in das Play!-Framework

Frank Goraus

MATHEMA Software GmbH

Gliederung

- Let's have a look
  - Schnellübersicht
  - Buzzword Bingo
  - Unter der Haube
- Let's Play! (Live Coding)
- A deeper Look
  - Weiterführende Features
  - Model, View, Controller
  - Validation
  - Tests

Schnellüberblick

- „Fix and reload“
- Einfache Fehlersuche
- „Groovy“ges Template System
- Full Stack (Hibernate, OpenID, MemCached)
- Stateless Model („share nothing“)
- Asynchron (→ RESTful architectures)
- 100% Java (Scala in 2.0)
- „Fun & Productive“

Buzzword Bingo

- Stateless MVC architecture
  - „share nothing“
  - Browser History
- HTTP-to-code mapping
  - Reines HTTP
- Efficient templating engine
  - Keine Taglibs
  - Voller Zugriff auf das Model
  - Groovy expression language
    - Elvis-Operator, dynamische Methoden, usw.

Buzzword Bingo 2

- „JPA on Steroids“
  - Keinerlei Konfiguration nötig
  - Entity Manager startet und synchronisiert automatisch
- Test driven development
  - Unit Tests
  - Function Tests
  - Acceptance Tests (Selenium)
- Modulares Pluginsystem

Unter der Haube

- Play ist nicht nur das Framework, „Play ist die Plattform“
- Play! Server
- MVC
  - Model
    - Daten Repräsentation & Bussiness Logik
  - View
    - HTML, XML, JSON, Bilder, …
  - Controller
    - Verarbeitet Events (=HTTP Requests)
    - Responsen und Modeländerungen

Hello Play!

Hello Play!
(Live coding)

Weiterführende Features

- Renderer
  - render(), renderText(), renderXML(), renderBinary(), renderJSON()
- Controller Annotations
  - @Before, @After, @Finally
- „Session Object“
  - = Cookie
  - „flash“, nur für einen Request gültig

View – Syntax

- Expressions - ${...}
- Tags - #{...}
  - #{a @Application.index()}Home#{/a}
  - #{list items:users as:'user'}
    ${user}
    #{/list}
- Actions - @{...} (relative URL), @@{...} (absolute URL)
- Messages - &{...}
- Kommentare - *{...}*
- Scriptlet Code - %{...}%

View - Templates

- Einbindung über spezielle Tags
  - #{extends 'main.html' /}
    #{set title:'Home' /}
- Custom Tags
  - app/views/tags → tagname.html anlegen
  - über #{tagname … /} einbinden
- Implizt verfügbare Objekte
  - session, flash, request, params, play (Konfiguration der App), lang, messages, out

View - Templates 2

- Java Object Expressions
  - dynamisch zur Laufzeit verfügbar
  - eigene Expressions erstellbar
    - public class MyExpression extends play.templates.JavaExpressions {

```
public static String myMethod( Object s, int aParam ) {
  …
}
public static String anotherMethod( String s ) {
  …
}
}
```

Models - JPA

- Einfach aktivierbar
  - conf/application.conf → DB anbinden
  - @javax.persistence.Entity
    public class Pojo extends play.db.jpa.Model
  - automatische ID am Objekt
  - Methoden wie findById(), save(), delete(), find(expression)
- direkter Datenbankzugriff auch über play.db.DB möglich
- Play Cache (memcached, ehCache)

ePlay

ePlay
(mehr Live coding)

Validation

- Variable validation im Controller
- validation.required(user);
  validation.email(email);
  validation.email(email).message(„Email ungültig“);
  validation.addError(„user“, „Username fehlt“);
- If (validation.hasErrors()) {
  params.flash();
  validation.keep();
  formAnzeigeMethode();
  }

Validation 2

- Verschachtelung von Validierungen
  - If (validation.required(user).ok) {
    validation.minSize(user, 6);
    }
- Auch als Annotation
  - public static void validate(@Requiered String user) { … }
  - public static void validate(@Valid User user) { … }

```
public class User {
  @Required
  public String name;
}
```

Validation 3

- Eigene Validatoren
  - public class User {

```
@CheckWith(MyValidation.class)
public String name;

static class MyValidation extends Check {
  public abstract boolean isSatisfied(Object user, Object
  name) {
    return … ;
  }
}
}
```

Tests

- Unit Tests
  - public class MyTest extends UnitTest {

```
private String play, fun;
@Before
public void setup() {
  play = fun = „true“;
}
@Test
public void testSomething() {
  assertEquals(play, fun);
}
```

}

Tests 2

- Funktionstests
  - Public class MySecondTest extends FunctionalTest {

```
@Test
public void testIndex() {
  Response res = GET(„/“);
  assertIsOk(res);
  assertContentType(„text/html“, res);
  assertStatus(404, res);
  assertTrue(getContent(res).contains(„Home“));
}
```

Tests 3

- Aktzeptanztest – Selenium
  - #{selenium 'Test'}
    open('/admin')
    waitForPageToLoad(1000)
    type('user', 'admin')
    type('password', 'admin')
    clickAndWait('login')
    assertTextPresent('Hallo Admin')
    #{/selenium}

Play! 2.0

- Teil des Typesafe Stack (2.0)
- SBT Integration (Build - früher über Python Skripte)
- Akka Integration
- Nativer Scala Support
  - Kern in Scala + Java API
  - Template Engine nun Scala statt Groovy basiert
- Nicht kompatibel zu Play 1.x
  - auch Plugins
- direkte Integration von dynamischen Web-Features (Comet, WebSockets)

Play! 2.0

Play 2.0
(Live Demo)

Vielen Dank!

Frank Goraus

MATHEMA Software GmbH