7363 - Web-basierte Anwendungen
4750 – Web-Engineering

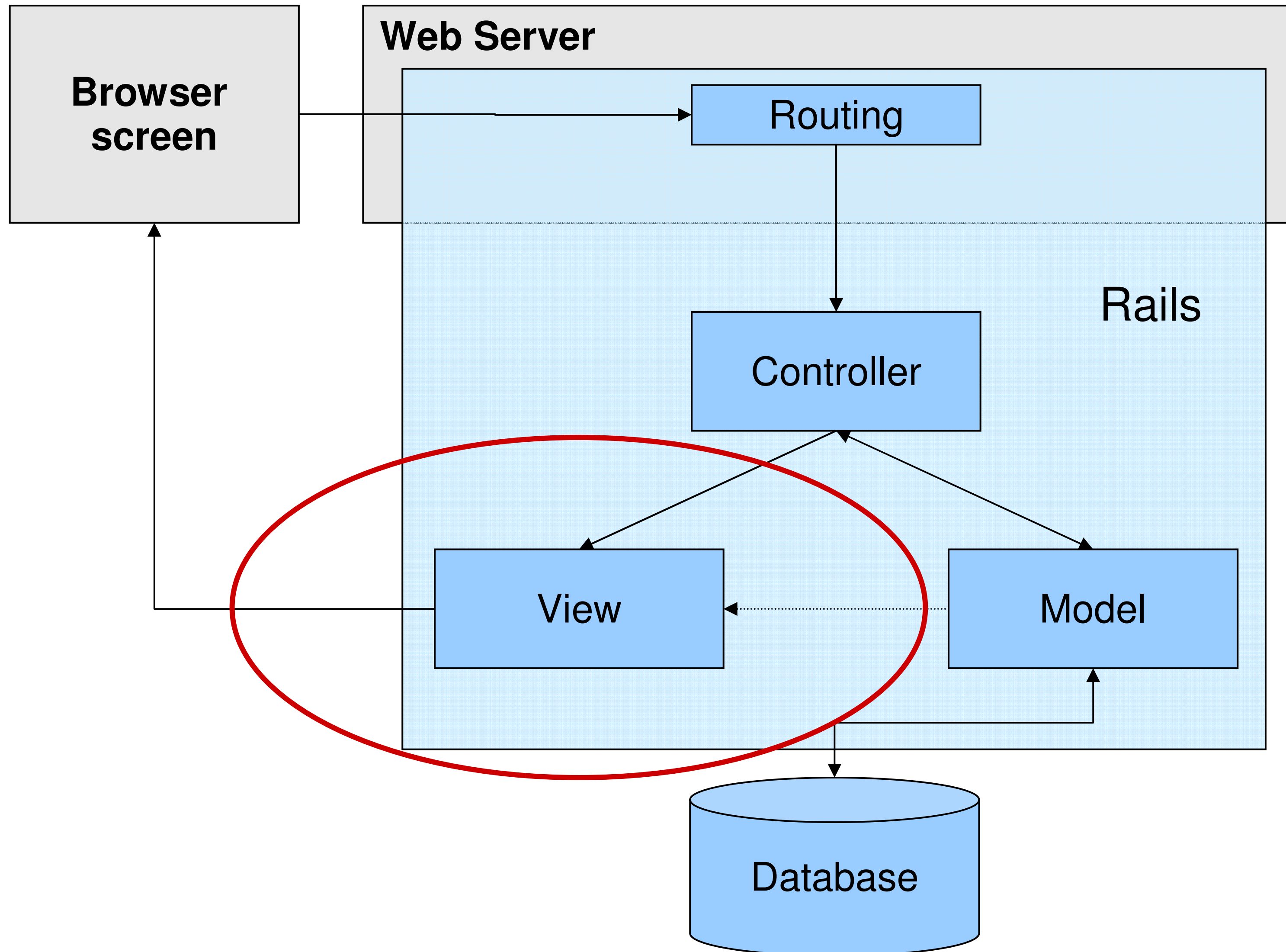
Eine Vertiefungsveranstaltung

© 2005, 2009 H. Werntges, Studienbereich Informatik, FB DCSM, FH Wiesbaden

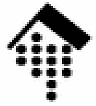
Views

© 2005, 2009 H. Werntges, Studienbereich Informatik, FB DCSM, FH Wiesbaden

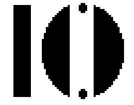
Views



© 2005, 2009 H. Werntges, Studienbereich Informatik, FB DCSM, FH Wiesbaden

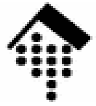


Views

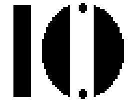


- Aufgaben eines Views (XHTML)
  - Organisation der Server-Antwort
  - Umsetzung des Schablonenkonzepts
    - Aufnahme konstanter Ausgabeteile
    - Befüllung der variablen Teile mit Daten aus den Modellen
  - Organisation von Eingabemasken / Formularen

- Aufgaben eines Views (andere Formate)
  - Im Prinzip ähnlich
  - JavaScript (JS): Genutzt im AJAX-Kontext oder Einbettung in HTML
  - XML: Genutzt insb. im REST-Kontext zur Anwendungskopplung
    - Mit der „Builder"-Bibliothek ist XML-Erzeugung vergleichsweise einfach möglich

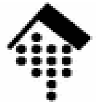


Views

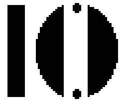


- Vorbemerkungen
  - Allein mit Views kann man sich viele Stunden beschäftigen
  - Wir konzentrieren uns auf ein paar grundlegende Gedanken und einen wichtigen Ausschnitt der Möglichkeiten

- Weitere Informationen:
  - API-Doc: ActionView::Helpers::*Helper (z.B. ::FormHelper)
  - Rails Guides: „Action View Form Helpers"
    - "http://guides.rubyonrails.org/form_helpers.html
  - Ruby on Rails 2: Kap. 8 „Templatesystem mit ActionView"
    - http://openbook.galileocomputing.de/ruby_on_rails/ruby_on_rails_08_001.htm
  - Agile Web Development With Rails (2nd ed.): Ch. 22 „Action View"

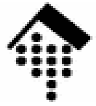


Views

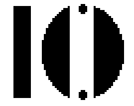


- Organisation der Abläufe (Erinnerung, Fall (X)HTML)
  - Das Routing bildet URL-Komponenten auf Controller-Methoden ab
  - Diese sog. „Aktionen" bewirken Ausgaben (meist mittels „render") oder leiten an andere Aktionen bzw. Seiten weiter
  - „render" bewirkt die Nutzung hinterlegter Views
    - Auch implizit, wenn zwar die Aktion nicht existiert, wohl aber die passende View-Datei
  - Eine View-Datei liefert nur den wesentlichen Inhalt eines vollständigen HTML-Dokuments (z.B. den Inhalt von „body") – der Rest stammt aus der gleichnamigen Datei in „app/views/layouts"
  - *Code sharing (DRY)*:
    - Ihre Controller-Klassen sind nicht direkt von ActionController::Base abgeleitet, sondern von ApplicationController (Code in Datei app/controllers/application.rb)
    - Analog wirkt „app/views/layouts/application.html.erb" als gemeinsame Layoutdatei. Modelle benötigen eigene Layout-Templates nur im Fall von Abweichungen.
    - Partials in Views: Nutzungsbeispiel für app/views/books (new & edit)

© 2005, 2009 H. Werntges, Studienbereich Informatik, FB DCSM, FH Wiesbaden

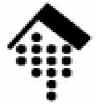


Views

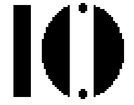


In app/views/books: **edit.html.erb**	In app/views/books: **new.html.erb**	**View-Datei**
In app/views/books: **_books_input.html.erb**		**Partial-Datei**
In app/views/layouts: **application.html.erb**	In app/views/layouts: **books.html.erb**	**Layout-Datei**
In public/stylesheets: **scaffold.css**	In public/stylesheets: **books.css**	**CSS-Datei**

© 2005, 2009 H. Werntges, Studienbereich Informatik, FB DCSM, FH Wiesbaden



Views



- Einfachstes XHTML-Beispiel
  - Rein statische Komponenten genügen:

```
<h1>Test 1</h1>
<p>Hello world!</p>
```

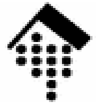
- Zusätzliche eRb-Anteile
  - Siehe Vorlesungsteil „wba-2-html-cgi“

```
<h1>Test 2</h1>
<p>Die Uhrzeit ist <%= Time.now %> </p>
```

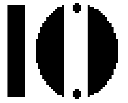
  - Einbettung von Code ohne direkte Ausgabe

```
<h1>Test 3</h1>
<% 3.times do |i| %>
<p>Paragraph <%= i %> </p>
<% end %>
```


© 2005, 2009 H. Werntges, Studienbereich Informatik, FB DCSM, FH Wiesbaden



Views



- Vorsicht vor *code insertion*!
  - Reichen Sie Benutzer-Eingaben nie ungefiltert an die Ausgabe oder einen Interpreter weiter – die Eingaben könnten zweckentfremdet werden!

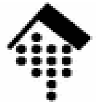
```
<h1>So nicht!</h1>
<p>Formularfeld "name" enthält: <%= params[:name] %></p>
```

  - Anwender könnten außer Nutztext auch HTML-Markup einschmuggeln!

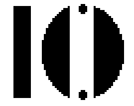
- Ausweg
  - Filterung/Maskierung kritischer Eingabesequenzen
    - Rails bietet dazu die Helper-Methode „h“.

```
<h1>So ist es besser!</h1>
<p>Formularfeld "name" enthält: <%= h(params[:name]) %></p>
```


© 2005, 2009 H. Werntges, Studienbereich Informatik, FB DCSM, FH Wiesbaden



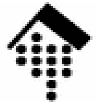
Views



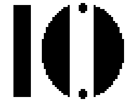
- Helper-Methoden
  - **Views sollen möglichst wenig Code enthalten**
    - Views bleiben dadurch lesbar und wartbar
    - Logik gehört nicht in Views!
    - *Stay DRY* - Code-Redundanzen vermeiden!
  - Wohin mit erforderlichem, aber länglichem Code?
    - **Auslagerung in Helper-Methoden (→ `app/helpers`)**
    - Möglichkeit zum *code sharing*
- Eingebaute Helper-Methoden
  - Rails bietet eine Vielzahl solcher Hilfsmethoden für Ausgabeformatierung, Verlinkung oder auch Seitenumbruch

```
<%= debug(params) %>           <!-- Bequemes Debugging -->
<%= number_with_precision(50.0/3) %>   <!-- 16.667 -->
<%= truncate(@txt, 10) %>            <!-- abcdefg... -->
```


© 2005, 2009 H. Werntges, Studienbereich Informatik, FB DCSM, FH Wiesbaden



Views



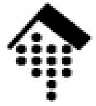
- Formular-Helper
  - Der Aufbau von Formularseiten ist ein zentrales Anliegen von WBA-Entwicklern. Rails bietet daher viele Methoden zur Kapselung und effizienten Erzeugung und Verwendung von Formularkomponenten
  - Erinnerung: Rails-Vorübung
  - Beispiele: „lib“-Projekt
- Unterscheide
  - Das Formular enthält i.w. die Daten eines Modells

```
<% form_for(@book) do |f| %>
…
  <%= f.text_field :pub_year %>
```

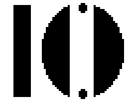
  - Das Formular enthält gemischte Daten

```
<% form_for(@book) do |f| %>
…
  <%= text_field :book, :pub_year %>
```


© 2005, 2009 H. Werntges, Studienbereich Informatik, FB DCSM, FH Wiesbaden



Views



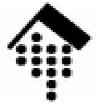
- Abläufe bei der Nutzung von Formularen, Bsp. „edit"-Zyklus
  - **Man beachte die Namenskonventionen!**
  - Index-Seite, „edit"-Link eines Eintrags anklicken
    → `GET /books/621/edit`
  - Routing setzt params[:id] auf „621", startet `BooksController#edit`

```
def edit
  @book = Book.find(params[:id])
end
```

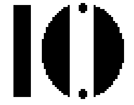
  - Rails liefert Ergebnis von *view template* „books/edit.html.erb" aus

```
…
<% form_for(@book) do |f| %>
…       <!-- Nur ein Auszug -->
 <p>
  <%= f.label :pub_year %><br />
  <%= f.text_field :pub_year %>
 </p>
<% end %>
…
```


© 2005, 2009 H. Werntges, Studienbereich Informatik, FB DCSM, FH Wiesbaden



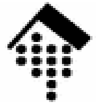
Views



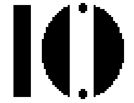
- Abläufe bei der Nutzung von Formularen, Bsp. „edit"-Zyklus
  - Entsprechender Ausschnitt aus dem ausgelieferten HTML-Code:

```
…
<form action="/books/621" class="edit_book" id="edit_book_621"
  method="post">
  <div style="margin:0;padding:0">
    <input name="_method" type="hidden" value="put" />
    <input name="authenticity_token" type="hidden"
      value="e5ba1ebb4dd2c333e0892a591b77129cf215c6fd" />
  </div>
  …
  <p>
    <label for="book_pub_year">Pub year</label><br />
    <input id="book_pub_year" name="book[pub_year]"
      size="30" type="text" value="2006" />
  </p>
  …
</form>
…
```

Pub year
2006



Views

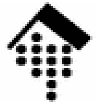


- Abläufe bei der Nutzung von Formularen, Bsp. „edit"-Zyklus
  - Anwender ändert die Jahreszahl auf „2007" und klickt Button „Update"
  - Browser sendet die Formulardaten per „POST" im HTTP body:

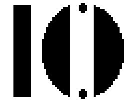
```
POST /books/621 HTTP/1.1
…
Content-Type: application/x-www-form-urlencoded
Content-Length: 277

_method=put&authenticity_token=e5ba1ebb4dd2c333e0892a591b7712
  9cf215c6fd&book%5Btitle%5D=Cluster+Computing&book%5Bauthor_i
  ds%5D%5B%5D=27&book%5Bpublisher_id%5D=17&book%5Bpub_year%5D=
  2007&book%5Bedition%5D=1&book%5Bisbn%5D=3-540-42299-
  4&book%5Bgtin%5D=9783540422990&commit=Update
```

  - Rails routet zur Aktion „update" von BooksController, extrahiert die Formularparameter und befüllt damit „params"

```
params[:book][:pub_year] == "2007"          # true
params["book"]["pub_year"] == "2007"        # auch: true
```



Views

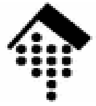


- Abläufe bei der Nutzung von Formularen, Bsp. „edit“-Zyklus
  - Die Aktion „update“ reicht params[:book] weiter zwecks Aktualisierung von DB-Spalten des Eintrags

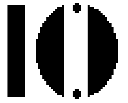
```
def update
  @book = Book.find(params[:id])
  respond_to do |format|
    if @book.update_attributes(params[:book])
      flash[:notice] = 'Book was successfully updated.'
      format.html { redirect_to(@book) }
      format.xml  { head :ok }
    else # …
    end
  end
end
```

  - ActiveRecord-Methode „update_attribute“ findet im Hash-artigen Objekt params[:book] die Spaltennamen der assoziierten DB-Tabelle „books“ als keys. Hier gilt z.B.: **`"pub_year" => "2007"`**. „save“ erfolgt implizit!

© 2005, 2009 H. Werntges, Studienbereich Informatik, FB DCSM, FH Wiesbaden



Views

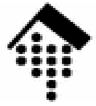


- **Generierung von Auswahl-Listen**
  - In der einfachen Form übergibt man einen iterierbaren Container von wählbaren Objekten mit lesbarer Stringdarstellung
  - Der ausgewählte String wird der Variablen zugewiesen. Beispiel:

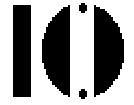
```
<% form_for(@book) do |f| %>
<%=  f.select :edition, 1..10 %> <!-- Integer erwartet -->
```

- Häufiger Spezialfall:
  - Auswahl von IDs gewünscht (z.B. von Modell-Objekten), aber in der Anzeige soll verständlicher Klartext statt Zahlen stehen
  - Dazu übergibt man ein Array von Arrays, etwa von Paaren [id, description]
  - Rails erzeugt dann option-Elemente mit Attribut „value“, dessen Wert dem ersten des Paares entspricht (Methode „first“). Die angezeigten Texte (ermittelt mit Methode „last“) bleiben Inhalt von „option“

```
<% pub_list = @publishers.map{|p| [p.id, p.name]} %>
<% form_for(@book) do |f| %>
<%=  f.select :publisher_id, pub_list %>
```



Views



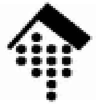
- Häufiger Spezialfall (Forts.):
  - Helper-Methode, die die Konstruktion des Hilfs-Arrays **pub_list** vermeidet:

```
<% form_for(@book) do |f| %>
<%=  f.collection_select :publisher_id, @publishers,
       :id, :to_s %>
```

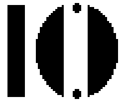
Name einer Methode, die die publisher_id liefert

Name einer Methode, die den anzuzeigenden Klartext liefert.

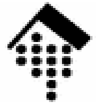
Hier ist die Stringdarstellung der Klasse Publisher dafür gut geeignet, daher „:to_s“



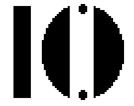
Views



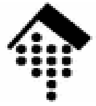
- Exkurs: *pagination*
  - Aufgabe:
    - Aufteilung vieler Treffer bei listenartigen Anzeigen in handhabbare Seitengrößen, etwa: 25 Einträge pro Seite
    - Generierung von Links zum „Blättern"
    - Effizienter Umgang mit DB und Speicher
  - Situation
    - Standard-Lösung von Rails 1.x wurde entfernt („ineffizient")
    - Heutige Empfehlung: „*Roll your own*" oder „Plugin / Gem nutzen"
    - Hier: Gem „mislav-will_paginate"
      - Quelle: `http://wiki.github.com/mislav/will_paginate`
    - Demo mit „books":
      - Installation, Konfiguration in config/environment, Controller, View



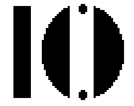
ActionMailer



- Exkurs: *ActionMailer*
  - Kein reines *View*-Thema!
    - Enthält eigene Model-Klassen
    - Benötigt Unterstützung in Controllern
  - Aufgaben:
    - Serverseitiges Versenden von E-Mails
      - *Plain text*, HTML, *attachments*, *multipart*-Nachrichten – alle MIME-Möglichk.!
    - Auch: Empfangen von E-Mails (hier nicht behandelt)
  - Nutzungsbeispiele:
    - Anmeldungsprozesse
      - *Dual opt-in*: Begrüßungs-E-Mail enthält Link, mit dem der Besitzer des E-Mail-Postfachs die Korrektheit einer Registrierung bestätigen kann
    - Admin-Benachrichtigungen
      - Diagnose- und Störmeldungen per E-Mail an Serverbetreuer
      - Kontaktaufnahme via Browser mit dem „*webmaster*"



ActionMailer



- Lösungsansatz (senden):
  - Konfiguration der Anbindungsdaten an einen SMTP-Server

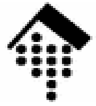
    → **./config/environments/development.rb** # test.rb, production.rb

  - Generierung einer Mailer-Klasse (Modell) mit speziellen Methoden zum Erzeugen von E-Mails sowie von passenden Views und Test-Code

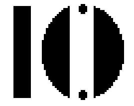
    → **./script/generate mailer *SomeMailer some_action***

  - Ausprogrammieren der Modell-Methoden
    - Insb. werden hier die variablen Daten in der E-Mail besorgt, etwa aus der DB
  - Ausgestaltung der Mail-Inhalte in Template-Technik, wie bei anderen Views
    - Dies schließt Text- als auch HTML-Mails ein.
  - Controller-Aktionen
    - Zunächst nur E-Mail-Erzeugung
    - Dann Versenden der erzeugten E-Mail
    - Kombination in einem Kommando möglich

© 2005, 2009 H. Werntges, Studienbereich Informatik, FB DCSM, FH Wiesbaden



ActionMailer

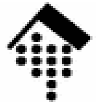


- Konfiguration
  - An `config/environments/development.rb` anfügen:

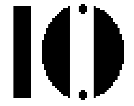
```
# Beispiele, es gibt weitere Parameter
config.action_mailer.raise_delivery_errors = false
config.action_mailer.perform_deliveries = true
config.action_mailer.default_charset = 'UTF-8'
config.action_mailer.smtp_settings = {
   :address =>        "smtp.informatik.fh-wiesbaden.de",
   :port =>           25,    # Darf fehlen (default)
   :domain =>         "informatik.fh-wiesbaden.de",
    # opt. weitere Angaben, etwa :user_name, :password
}
```

  - Ergänzende Dokumentation:
    - api.rubyonrails.org, ActionMailer::Base, Abschnitt „Configuration options“

© 2005, 2009 H. Werntges, Studienbereich Informatik, FB DCSM, FH Wiesbaden



ActionMailer

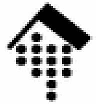


- Generator-Aufruf
  - Wir bereiten zwei Versendemethoden vor: *author_added, publisher_added*

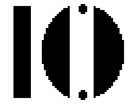
```
$ script/generate mailer DemoMailer author_added \
  publisher_added
```

```
      exists  app/models/
      create  app/views/demo_mailer
      exists  test/unit/
      create  test/fixtures/demo_mailer
      create  app/models/demo_mailer.rb
      create  test/unit/demo_mailer_test.rb
      create  app/views/demo_mailer/author_added.erb
      create  test/fixtures/demo_mailer/author_added
      create  app/views/demo_mailer/publisher_added.erb
      create  test/fixtures/demo_mailer/publisher_added
```


© 2005, 2009 H. Werntges, Studienbereich Informatik, FB DCSM, FH Wiesbaden



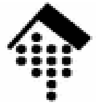
ActionMailer



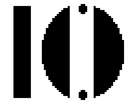
- Erzeugte Modell-Klasse in app/models/demo_mailer.rb (ergänzt)

```
class DemoMailer < ActionMailer::Base
  def author_added(author, sent_at = Time.now)
    subject    "Hi, #{author}" # "Author#to_s" nutzen!
    recipients 'werntges@informatik.fh-wiesbaden.de'
    from       'wba_demo@informatik.fh-wiesbaden.de'
    sent_on    sent_at
  end

  def publisher_added(publisher, sent_at = Time.now)
    subject    'DemoMailer#publisher_added'
    recipients 'werntges@informatik.fh-wiesbaden.de'
    from       'wba_demo@informatik.fh-wiesbaden.de'
    sent_on    sent_at
    body       :pub => publisher # Param-Übergabe an View
  end
end
```


© 2005, 2009 H. Werntges, Studienbereich Informatik, FB DCSM, FH Wiesbaden



ActionMailer

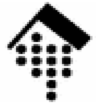


- *View* „app/views/demo_mailer/publisher_added.erb"

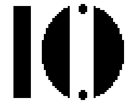
```
DemoMailer#publisher_added

Soeben wurde ein neuer Verleger hinzugefügt.
Sein Name: <%= @pub.name %>
Sein Ort:  <%= @pub.city %>
```

- *View* „app/views/demo_mailer/author_added.erb"

```
DemoMailer#author_added

Find me in app/views/demo_mailer/author_added.erb
```



ActionMailer



- In „app/controllers/publishers_controller.rb“

```
def create
  @publisher = Publisher.new(params[:publisher])
  DemoMailer.deliver_publisher_added(@publisher)
  # usw.
end
```

- In „app/controllers/authors_controller.rb“

```
def create
  @author = Author.new(params[:author])
  DemoMailer.deliver_author_added(@author)
  # usw.
end
```

- Wirkung:
  – Zwischen Speicherung und Rendern der Antwort wird E-Mail gesendet
  – Demo!

© 2005, 2009 H. Werntges, Studienbereich Informatik, FB DCSM, FH Wiesbaden