Fachhochschule Nordwestschweiz
Hochschule für Technik

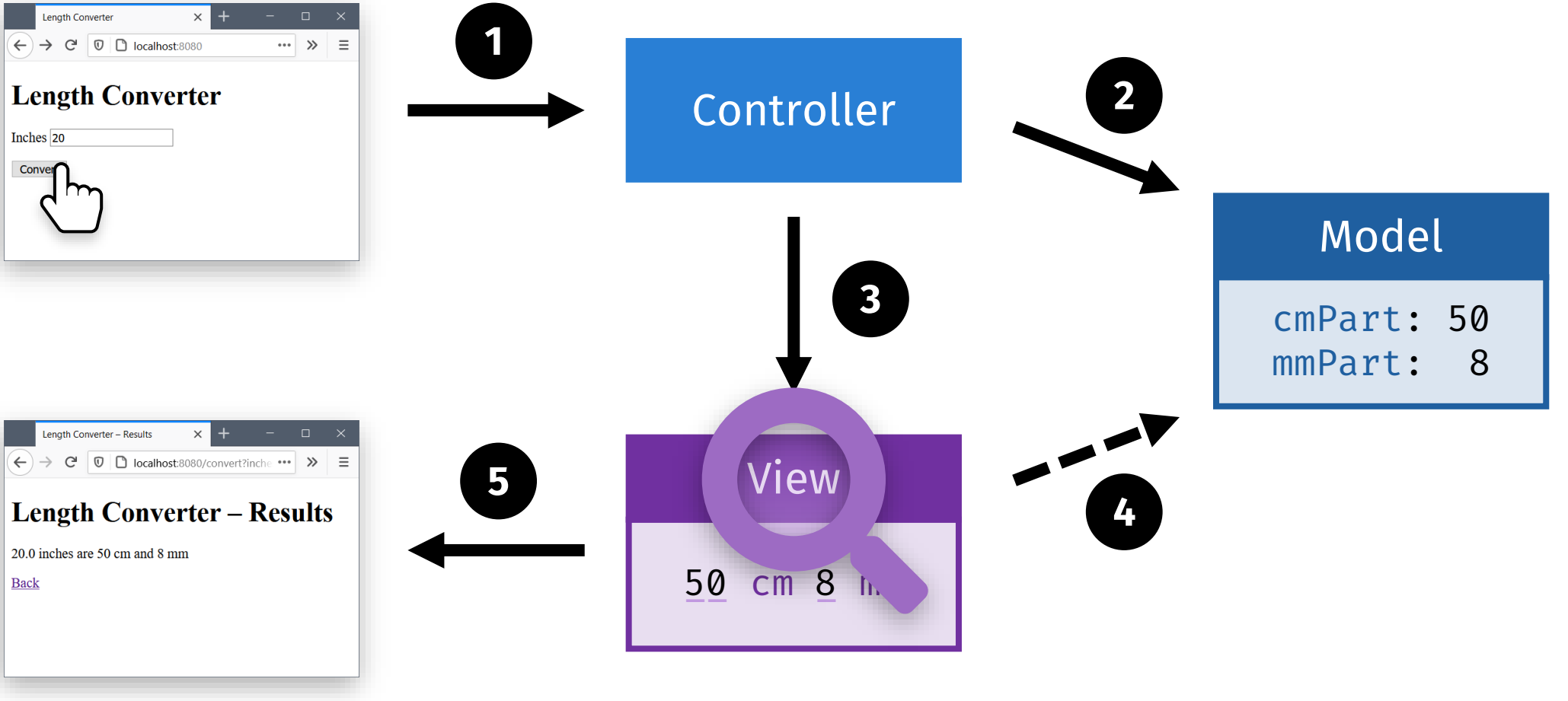
Web Engineering

Templates

Adrian Herzog

(basierend auf der Arbeit von Michael Faes, Michael Heinrichs & Prof. Dierk König)

Rückblick: MVC



Warum Templates?

MVC geht auch so:

```
@GetMapping("/hello")
public ResponseEntity<String> hello() {
    var name = ...
    return ResponseEntity.ok()
            .contentType(MediaType.TEXT_HTML)
            .body("""
                <!DOCTYPE html>
                <html>
                ...
                <p>Hello """ + name + """</p>
                </html>
                """);
}
```

Controller

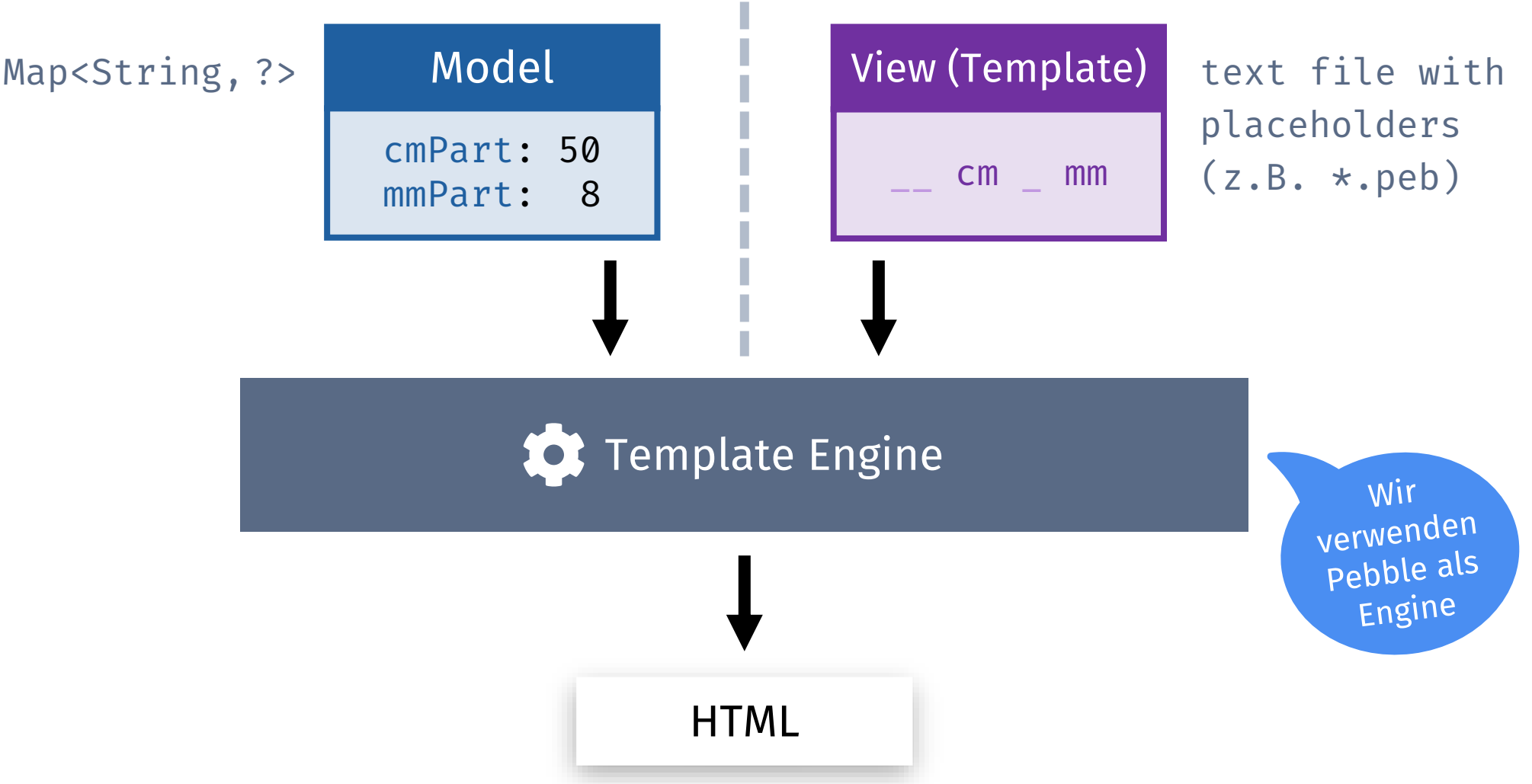
«View»

«Model»

Nachteile:

- Keine Trennung von Logik und Darstellung
  → Unübersichtlich, schlecht wartbar.
- String Concatenation ist umständlicher als Template-Syntax
  → Engines bieten geeignete Features für diese Aufgabe.

Mit Template Engine



HTML-Templates

HTML-Template: Mischung von HTML und *Anweisungen, wie Daten aus Model in Dokument dargestellt werden.*

Nicht nur Platzhalter!

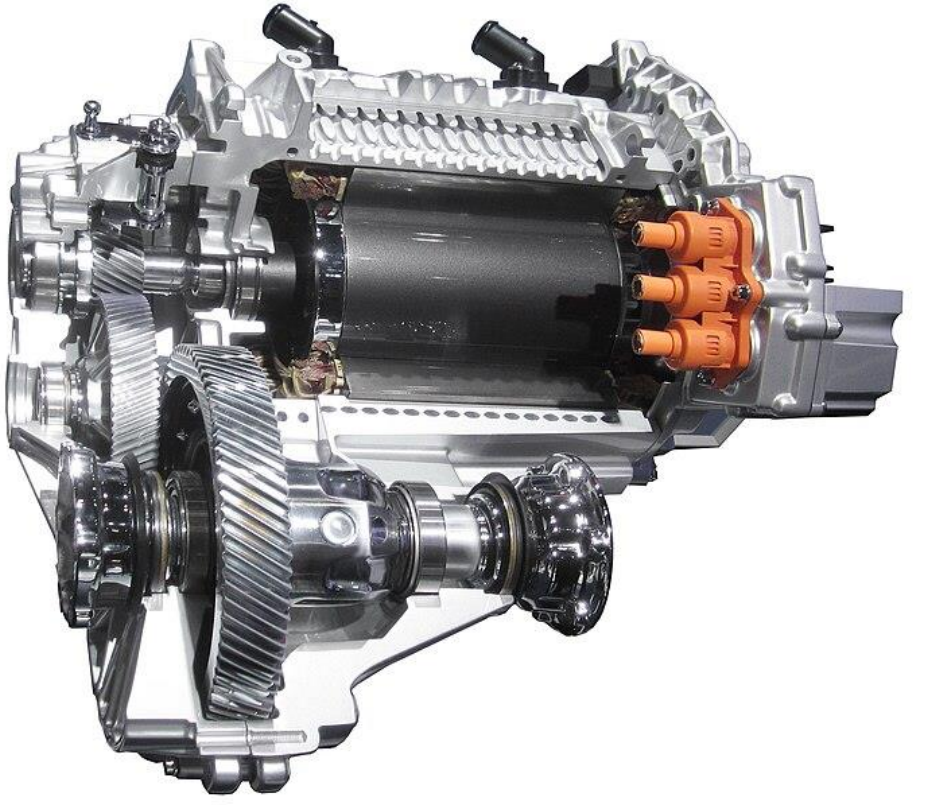
Beispiel JSP:

- Spezielle Tags `<%  …  %>` und `<%=  …  %>` grenzen HTML von Java-Code ab
- Normale Java-Kontrollstrukturen

```
<!DOCTYPE html>
<html>
<head><title>JSP</title></head>
  <%
    double num = Math.random();
    if (num > 0.95) {
  %>
      <h2>Lucky day!</h2>
      <p>(<%= num %>)</p>
  <%
    } else {
  %>
      <h2>Life goes on…</h2>
      <p>(<%= num %>)</p>
  <%
    }
  %>
</html>
```

Templating Engines in Spring Boot

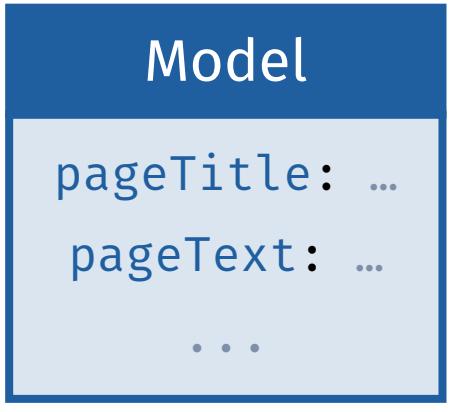
- Spring MVC ist modular aufgebaut und erlaubt uns, ***aus diversen Templating Engines auszuwählen*** (z.B. Thymeleaf, Pebble, FreeMaker).
- Zur Verwendung einer Templating Engine muss Spring korrekt konfiguriert werden. ***Spring Boot Starter*** Projekte übernehmen diese Konfiguration für uns.



Pebble

Template-Engine mit einfacher Syntax, aber mit vielen Features und Erweiterungsmöglichkeiten.

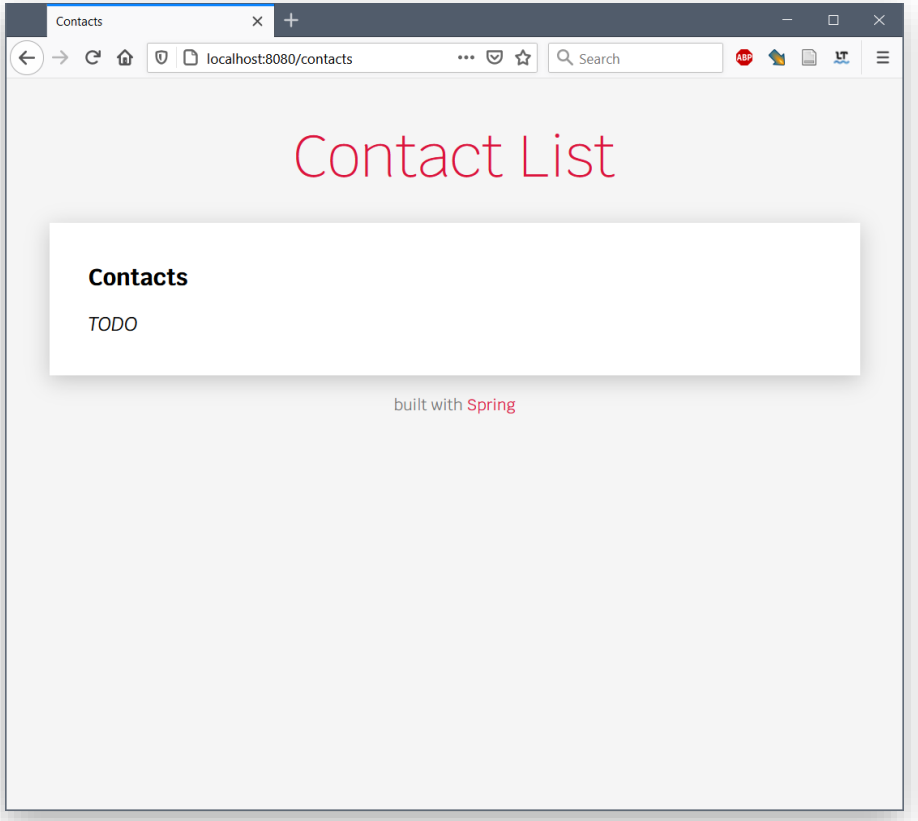
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>{{ pageTitle }}</title>
  </head>
  <body>
    <p>{{ pageText }}</p>
  </body>
</html>
```

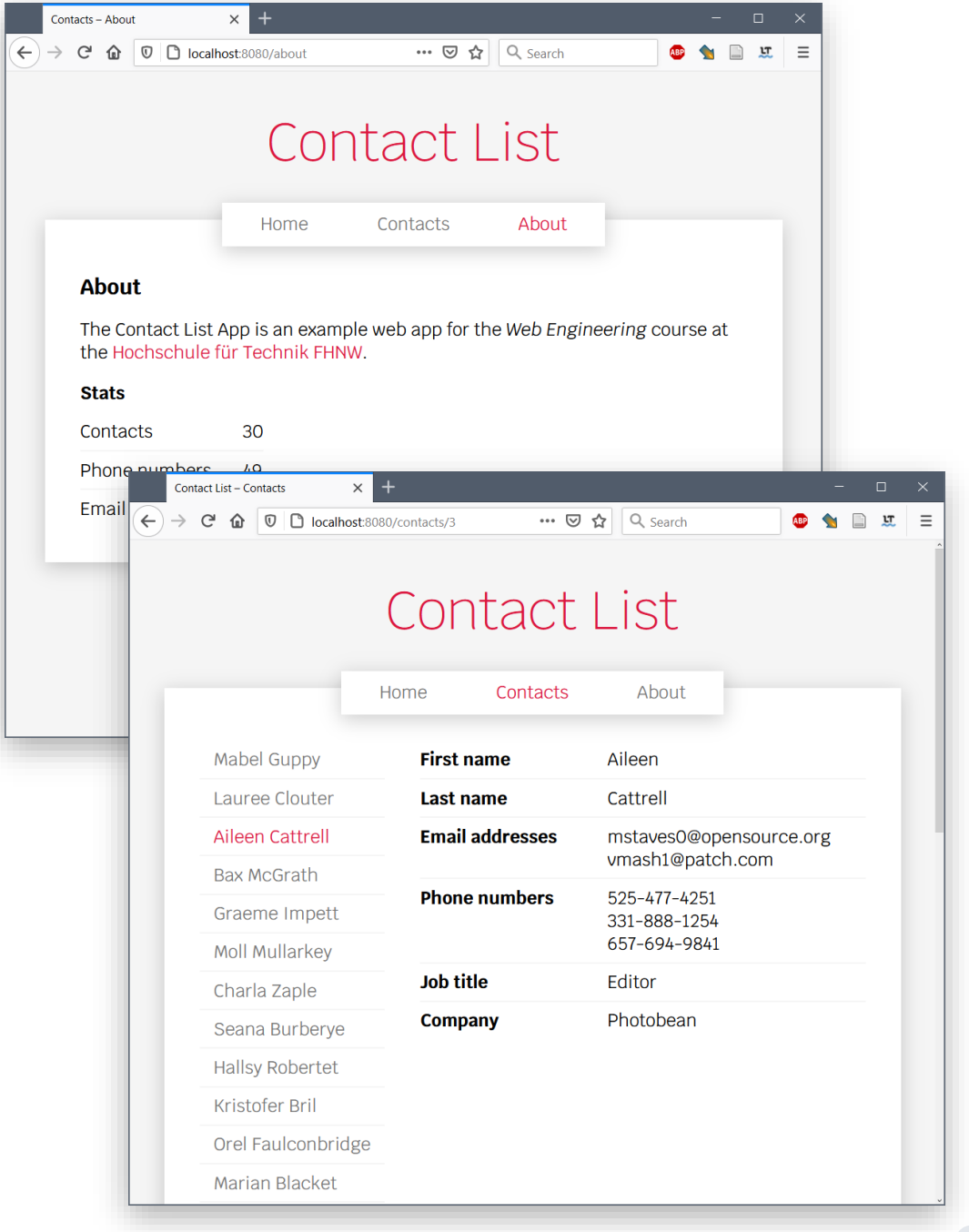


Der Starter konfiguriert Pebble für uns

https://pebbletemplates.io / Code vom Spring Boot Starter

Ziel heute:


Contacts
localhost:8080/contacts
Search
Contact List
Contacts
TODO
built with Spring


Contacts – About
localhost:8080/about
Search
Contact List
Home
Contacts
About
About
The Contact List App is an example web app for the Web Engineering course at the Hochschule für Technik FHNW.
Stats
Contacts 30
Contact List – Contacts
localhost:8080/contacts/3
Search
Contact List
Home
Contacts
About
Mabel Guppy
Lauree Clouter
Aileen Cattrell
Bax McGrath
Graeme Impett
Moll Mullarkey
Charla Zaple
Seana Burberye
Hallsy Robertet
Kristofer Bril
Orel Faulconbridge
Marian Blacket
First name Aileen
Last name Cattrell
Email addresses mstaves0@opensource.org vmash1@patch.com
Phone numbers 525-477-4251 331-888-1254 657-694-9841
Job title Editor
Company Photobean

Wichtigste Template-Elemente

Variablen und Ausdrücke

Zugriff auf Model innerhalb von {{ … }}:

```
<p>{{ pageText }}</p>
```

Zugriff auf Methoden und Attribute von Objekten (*fields*):

```
<strong>{{ person.name }}</strong><br>
{{ person.address.street }} {{ person.address.number }}
```

Ausdrücke werden interpretiert wie folgender Java-Code:

```
model.getAttribute("person").getName();
```

```
model.getAttribute("person").getAddress().getStreet();
```

```
model.getAttribute("person").getAddress().getNumber();
```

Berechnungen auch direkt in Template möglich…

```
Erreichte Credits: {{ stud.modules.size * 3 }} ECTS
```

… inklusive diverser Java-Operatoren, Zahlen und Strings …

```
Erreichte Credits:
<span class="{{ stud.credits >= 180 ? "done" : "" }}">
   {{ stud.credits }} ECTS
</span>
```

… aber oft eine schlechte Idee! Besser in Controller oder Model!

Warum?

- Pebble-Ausdrücke sind nicht statisch typisiert, sondern dynamisch
- Java-Code kann einfacher getestet werden
- *Separation of concerns*: Anwendungslogik gehört nicht in View

Filter

Die Ausgabe von Variablen und Ausdrücken kann mit *Filter* modifiziert werden:

```
<p>{{ text | upper }}</p>
```
→
```
<p>HELLO WORLD</p>
```

```
<p>{{ text | abbreviate(10) }}</p>
```
→
```
<p>Hello W...</p>
```

```
<p>{{ day | date("dd.MM.yy") }}</p>
```
→
```
<p>12.04.22</p>
```

Filter können auch aneinandergereiht werden:

```
<p>{{ text | abbreviate(10) | upper }}</p>
```

Kontrollstrukturen (if / for)

Damit können wir das Template dynamischer gestalten.

```
model.addAttribute(“student", service.getStudent());
```

im Controller

```
{% if student.modules.isEmpty %}
    <p>Noch keine Module belegt.</p>
{% else %}
    <ul>
        {% for module in student.modules %}
            <li>{{ module.name }}</li>
        {% endfor %}
    </ul>
{% endif %}
```

student.modules ist eine Liste

Tipp 1: Änderungen ohne Neustart

Um App nicht bei jeder Template-Änderung neustarten zu müssen, kann der Pebble-Cache deaktiviert werden:

```
pebble.cache=false
```

in die Datei `application.properties` einfügen

Tipp 2: Strikter Modus

Pebble ist standardmässig sehr tolerant und wirft keinen Fehler, wenn ein Feld nicht gefunden wird (z.B. wegen einem Schreibfehler). Das kann man ändern:

```
pebble.strict-variables=true
```

in die Datei `application.properties` einfügen

→ benötigt aber einen anderen Umgang mit Variablen die null sein können:

`{% if contact is defined %}` statt `{% if contact != null %}`

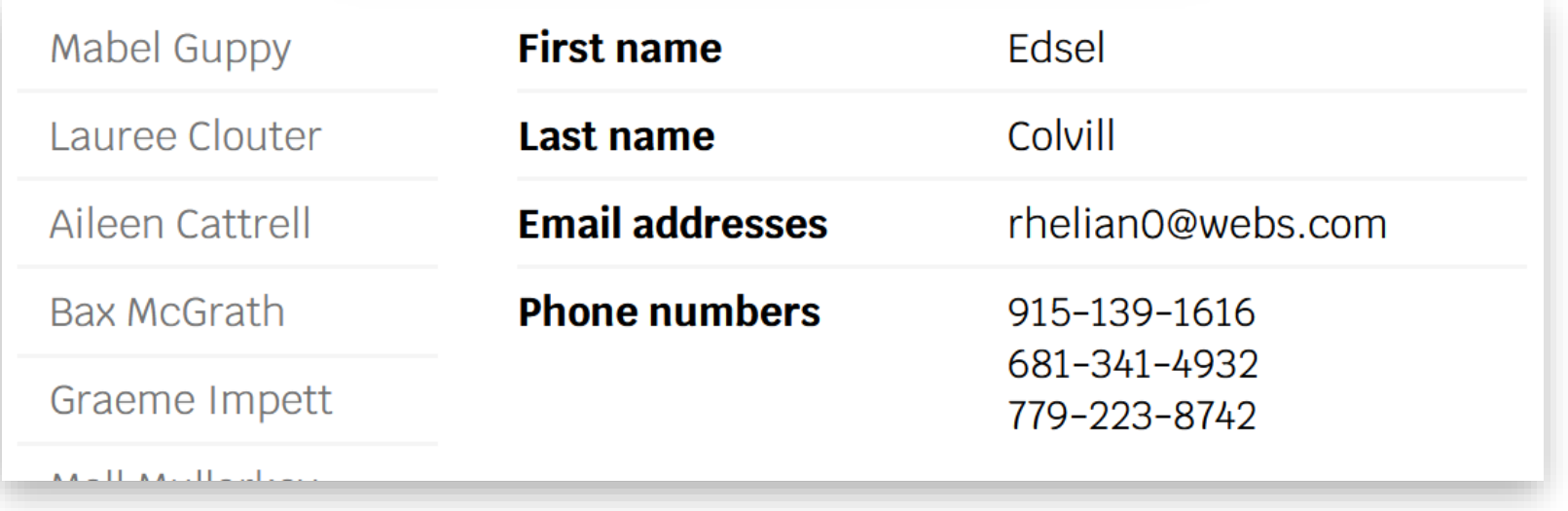
Übung 1: Schleifen in Templates

a) Mache dich mit dem Code im Projekt «contactlist-pebble» vertraut. Beachte insbesondere den vorhandenen Controller und das dazugehörige Template. Kompiliere und starte die App und prüfe, dass sie unter http://localhost:8080/contacts erreichbar ist (unter / gibt es einen Fehler).

b) Erweitere das Contacts-Template so, dass die Liste von Kontakten angezeigt wird. Dafür brauchst du `{% for … %}` und das `contactList`-Attribut aus dem Model. Jeder der Einträge soll ein Link sein, der auf /contacts/{id} verweist, wobei {id} die ID des entsprechenden Kontakts ist. Für diese URLs gibt es bereits ein Mapping im vorhandenen Controller, welches wieder dasselbe Template zurückgibt.

Übung 2: Verzweigungen in Templates

Erweitere das Template erneut, sodass rechts neben der Kontaktliste eine Tabelle mit den Details des ausgewählten Kontakts angezeigt wird (`contact`-Attribut aus dem Model).



Für fehlende Eigenschaften soll keine Zeile angezeigt werden. Falls kein Kontakt ausgewählt ist, soll eine Meldung zum Auswählen eines Kontakts eingeblendet werden. Verwende `{% if … %}`.

Komposition

Auch bei Templates möchten wir duplizierten Code vermeiden.

Eine Möglichkeit: **include**

fragments/module-list.peb:

```
{% if stud.modules.isEmpty %}
   <p>Noch keine Module belegt.</p>
{% else %}
   <ul>
      {% for module in stud.modules %}
         <li>{{ module.name }}</li>
      {% endfor %}
   </ul>
{% endif %}
```

student-details.peb:

```
<h2>Modul-Liste</h2>
{% include "fragments/module-list" %}
```

student-overview.peb:

```
<h3>Module von {{ stud.name }}</h3>
{% include "fragments/module-list" %}
```

include-te Templates haben Zugriff auf gleiche Variablen wie das aktuelle Template.

Manchmal praktischer: **macro**. Wiederverwendbares Template-Stück, mit *Parametern*.

```
{% macro input(type, name) %}
   <input type="{{ type }}" name="{{ name }}">
{% endmacro %}
...
{{ input("text", "firstName") }}
{{ input("text", "lastName") }}
```

form/stud.peb

Verwenden in anderem Template mit **import**:

```
{% import "form/stud" %}
...
{{ input("text", "firstName") }}
```

Layouts und Vererbung

Manchmal möchte man *inverse Komposition*: nicht Stücke in Seite einfügen, sondern Seite in ein Layout-Skelett einfügen.

Pebble: Templates können von anderen Templates erben:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>{% block title %}Demo{% endblock %}</title>
  </head>
  <body>
    {% block content %}{% endblock %}
  </body>
</html>
```

layout.peb

`block` definiert Template-Stück, das *überschrieben* werden kann.

Von einem anderen Template erben und Blöcke überschreiben:

```
{% extends "layout" %}

{% block title %}Modulübersicht{% endblock %}

{% block content %}
<main>
   <h1>Modulübersicht von {{ stud.name }}</h1>
   <p>...</p>
</main>
{% endblock %}
```

Fortgeschritten:

- Mehrstufige Vererbung: erbendes Template kann weitere Blöcke definieren, die wieder überschrieben werden können.
- Dynamische Vererbung: `extends` akzeptiert beliebige Ausdrücke!

Übrigens: URLs & Links

In HTML können drei Arten von URLs verwendet werden:

- Absolute URL: `https://fhnw.ch/de/studium/technik`
- Relative URL von *Root* aus: `/de/studium/technik`
- Relative URL von aktuellem Pfad aus: `studium/technik`

Auflösung von relativen Links ohne *Root-/*:

`https://fhnw.ch/de/die-fhnw` : Ordner «de», Dokument «die-fhnw»

Relative URL wird an Pfad von Ordner angehängt, d. h. *alles links von dem letzten /*:

`https://fhnw.ch/de/~~die-fhnw~~`

\+ `studium/technik`

= `https://fhnw.ch/de/studium/technik`

Ressourcen wie Stylesheets, Bilder, usw. werden sinnvollerweise mit relativer URL, aber von Root aus verlinkt, z. B. `/css/base.css`

- 👍 Es kann von allen Dokumenten aus die gleiche URL verwenden (wichtig für Templates!)
- 👍 Man muss Domain, unter der die App erreichbar ist, nicht kennen

Spring-Apps können aber auch in Unterpfad laufen, z. B.

https://www.cs.technik.fhnw.ch**/grading-server/**

«context path»

Best Practice mit Pebble: Immer `href`-Funktion verwenden, welche den Context Path automatisch hinzufügt, wenn nötig:

```
<link href="{{ href("/css/base.css") }}">
```

Fragen?