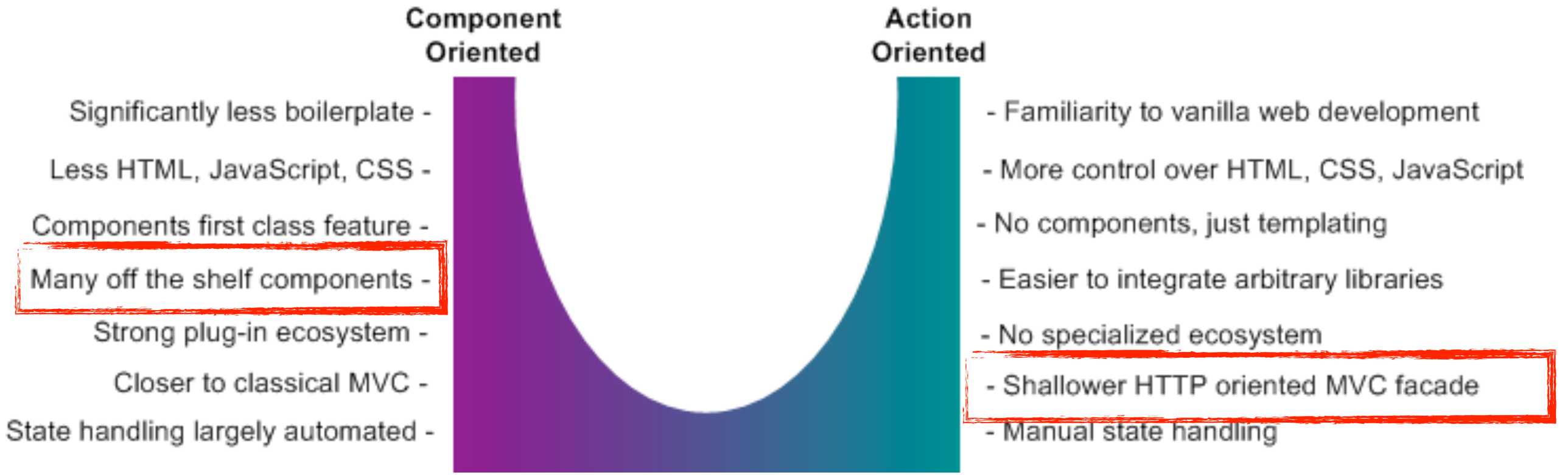
JavaServer Faces

Einordnung JavaEE

- In JavaEE gibt es zwei MVC-Frameworks
- Unterschied
  - **MVC**: aktionsorientiert (action oriented), vgl. Struts, SpringMVC, ASP.NET MVC
  - **JSF**: komponentenbasiert (component oriented), vgl Wicket, Vaadin, ASP.NET



- JavaServer Faces (JSF)
- Die Requests und Responses sind abstrahiert

- Java Server Pages (JSP)
- orientiert sich am Http-Lebenszyklus

https://blogs.oracle.com/theaquarium/entry/why_another_mvc_framework_in

Was ist JSF?

- JavaServer Faces (kurz JSF) ist ein Framework-Standard zur Entwicklung von **grafischen Benutzeroberflächen für Webapplikationen**. Basierend auf Servlets und JSP-Technik, gehört JSF zu den Webtechnologien der Java Platform, Enterprise Edition (Java EE). Mit Hilfe von JSF kann der Entwickler auf einfache Art und Weise Komponenten für Benutzerschnittstellen in Webseiten einbinden und die Navigation definieren. Voraussetzungen für die Entwicklung von JSF-Content sind das JDK, ein Servlet-Container (z. B. Apache Tomcat) und Grundlagenverständnis von HTML, HTTP und der Programmiersprache Java. Zur Vereinfachung der Entwicklung kann eine Integrierte Entwicklungsumgebung verwendet werden.

Wikipedia

Specification

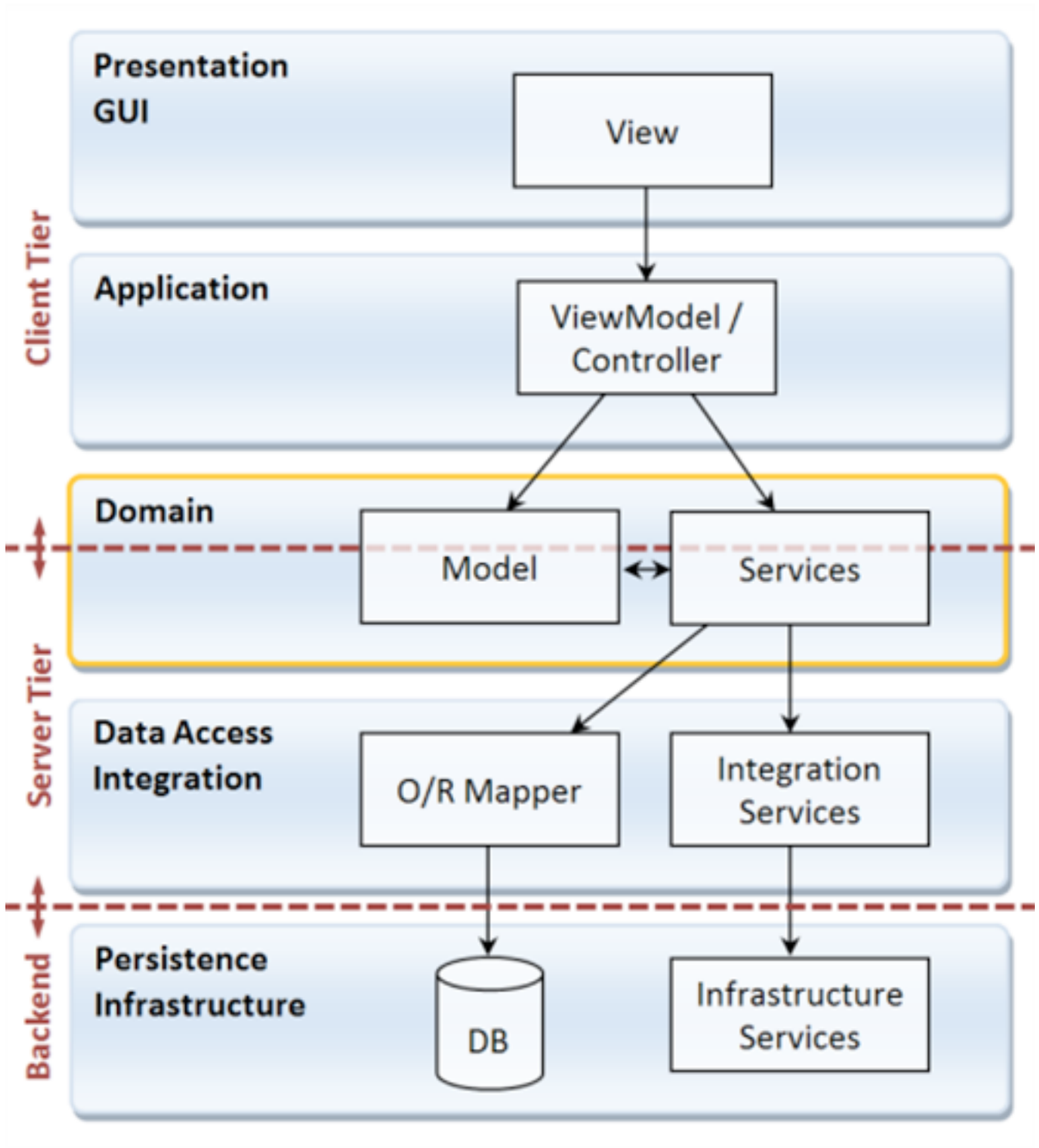
- JSR-000344 JavaServer Faces 2.2 Specification for Evaluation

- http://download.oracle.com/otndocs/jcp/jsf-2_2-fr-eval-spec/index.html

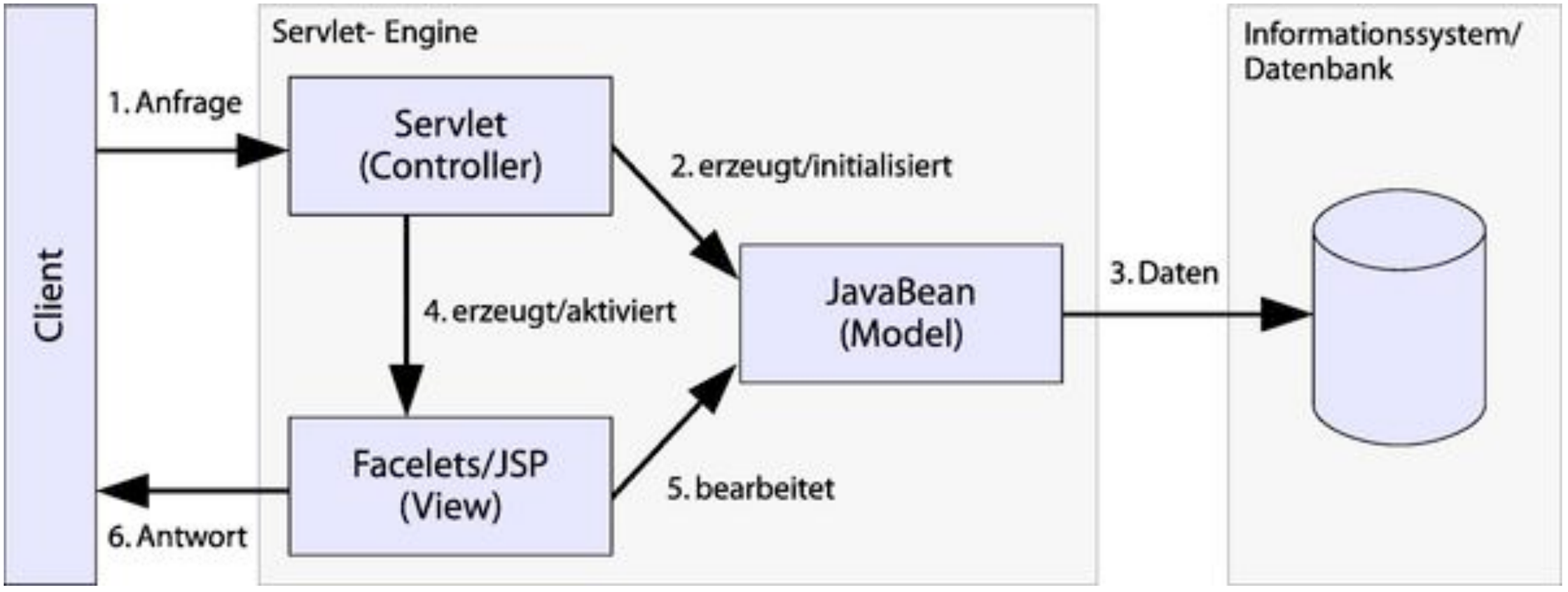
Implementations

- Mojarra (Referenzimplementierung)
- PrimeFaces http://www.primefaces.org/
- RichFaces (JBoss) http://richfaces.jboss.org/
- MyFaces (Apache): Tomahawk, Trinidad, Tobago, Orchestra
- Oracle ADF
- ICEfaces
- …

Overview

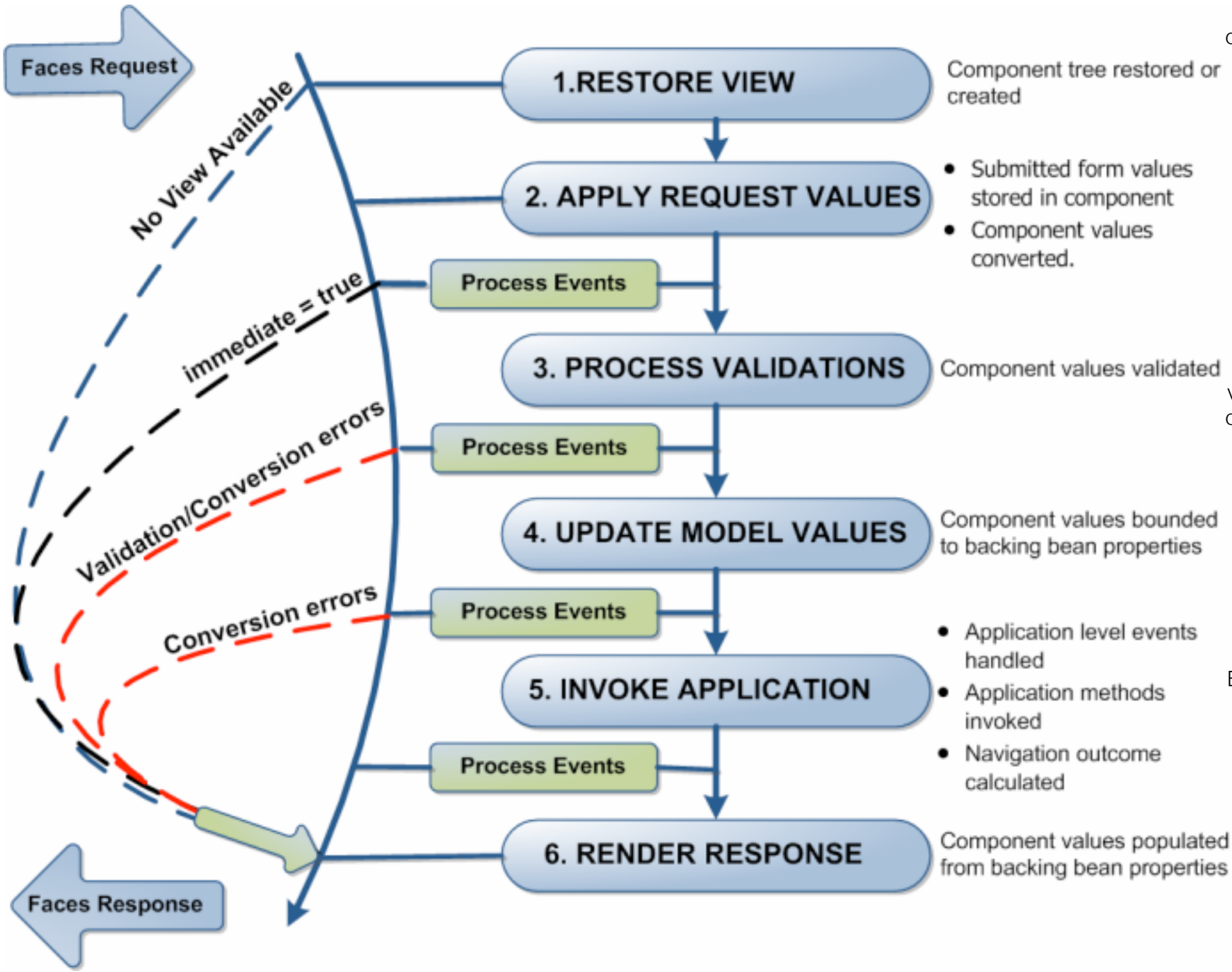

Client Tier
Presentation
GUI
View
Application
ViewModel /
Controller
Domain
Model
Services
Server Tier
Data Access
Integration
O/R Mapper
Integration
Services
Backend
Persistence
Infrastructure
DB
Infrastructure
Services

Ablauf



http://jsfatwork.irian.at/book_de/introduction.html#!idx:/introduction.html:1.1.1

JSF - Lebenszyklus



Ensures that the tree of the components and their states match the form when the view was originally generated at the response. JSF has to rebuild the view accurately before it can start inserting the changes and apply the form property values from the Faces request.

The framework iterates over the component objects in the tree. Each component retrieves the data from the request object, which are usually the request parameters but can be cookies, session attributes, or even header parameters.

The submitted string values, which are stored with the component, are converted to local values. These local values can be Java objects of any type. It is in this phase that the validators that are associated with the component can verify the value of the local values.

JSF updates the beans that are referenced by the component.

Business logic is invoked. JSF calls the method that is named by the action method of the command button or link component.

Encodes the Faces response and the JSF framework sends this output to the requesting user agent, which is usually a web browser.

https://bansihaudakari.wordpress.com/frameworks/jsf/jsf-lifecycle/

Rendering

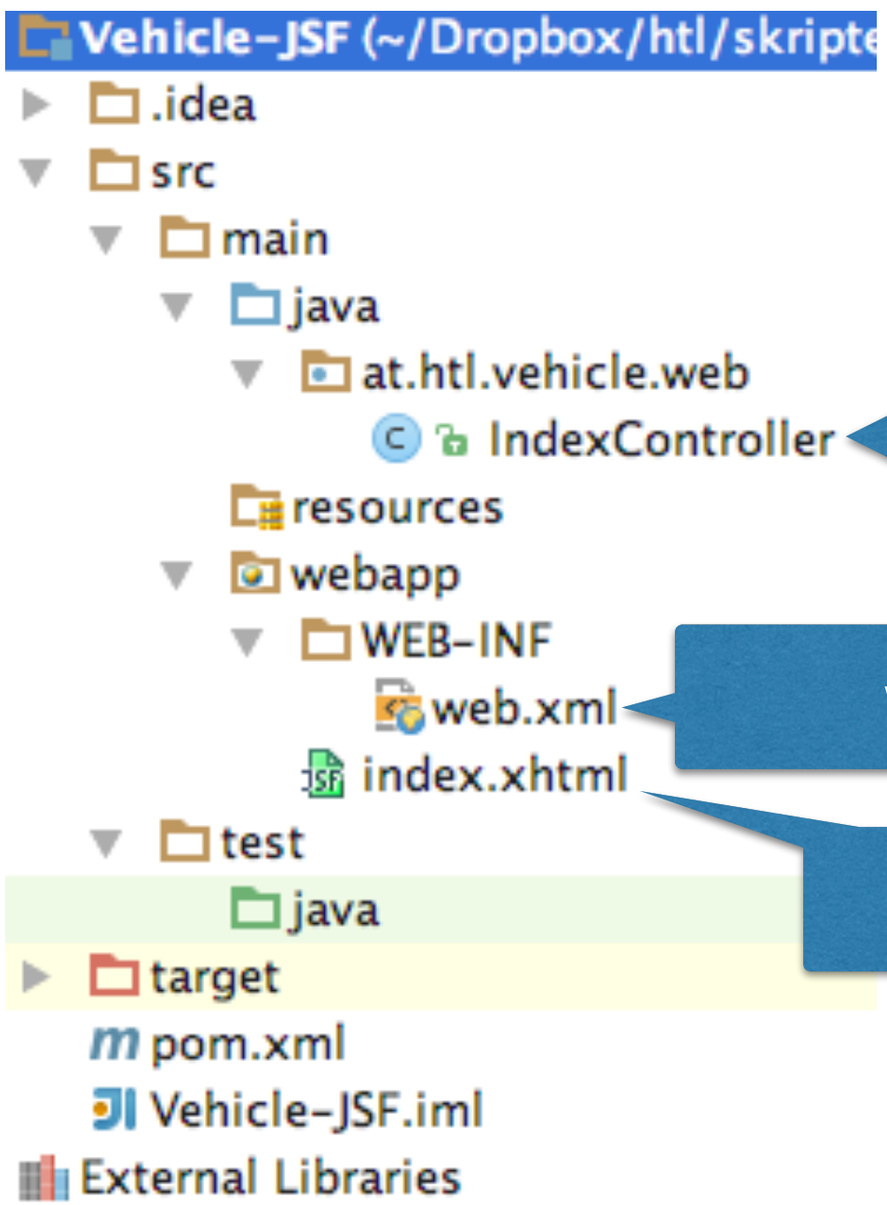
- JSP

- Facelets

TABLE 10-1 Comparison of Facelets and JSP

Feature Name	JSP	Facelets
Pages are compiled to...	A Servlet that gets executed each time the page renders. The UIComponent hierarchy is built by the presence of custom tags in the page.	An abstract syntax tree that, when executed, builds a UIComponent hierarchy.
Handling of tag attributes	All tag attributes must be declared in a TLD file. Conformance instances of components in a page with the expected attributes can be enforced with a taglibrary validator.	Tag attributes are completely dynamic and automatically map to properties, attributes and ValueExpressions on UIComponent instances
Page templating	Not supported, must go outside of core JSP	Page templating is a core feature of Facelets
Performance	Due to the common implementation technique of compiling a JSP page to a Servlet, performance can be slow	Facelets is simpler and faster than JSP
EL Expressions	Expressions in template text cause unexpected behavior when used in JSP	Expressions in template text operate as expected.
JCP Standard	Yes, the specification is separate from the implementation for JSP	No, the specification is defined by and is one with the implementation.

Struktur einer Anwendung



Das Model

web.xml: Konfiguration der themes

Webseiten

faces-config.xml dient zum Festlegen der Navigation zwischen Seiten. Ist aber nicht mehr notwendig.

pom.xml

Falls nur ein Theme benötigt wird:

```xml
<dependency>
    <groupId>org.primefaces.themes</groupId>
    <artifactId>cupertino</artifactId>
    <version>1.0.8</version>
    <scope>compile</scope>
</dependency>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-inst
         xsi:schemaLocation="http://maven.apache.org/POM/
    <modelVersion>4.0.0</modelVersion>

    <groupId>at.htl</groupId>
    <artifactId>vehicle-jsf</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>war</packaging>

    <dependencies>
        <dependency>
            <groupId>javax</groupId>
            <artifactId>javaee-web-api</artifactId>
            <version>7.0</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.primefaces</groupId>
            <artifactId>primefaces</artifactId>
            <version>5.3</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>org.primefaces.extensions</groupId>
            <artifactId>all-themes</artifactId>
            <version>1.0.8</version>
            <scope>compile</scope>
        </dependency>
    </dependencies>

    <build>
        <finalName>vehicle</finalName>
    </build>

</project>
```

web.xml

PROJECT_STAGE
Production: weniger Debug Statements, höhere Performance

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://
         version="3.1">
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</param-name>
        <param-value>Development</param-value>
    </context-param>
    <context-param>
        <param-name>primefaces.THEME</param-name>
        <param-value>cupertino</param-value>
    </context-param>
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>/faces/*</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>30</session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>faces/index.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

Welche Funktion haben wohl die einzelnen Einträge?

https://gist.github.com/ThomasStuetz/fa0b7e4b05d4359ebdea

IndexController.java

```java
package at.htl.vehicle.web;

import javax.enterprise.context.RequestScoped;
import javax.inject.Named;

@Named
@RequestScoped
public class IndexController {

    String name;

    public IndexController() { this.name = " HTL Leonding"; }

    public String getName() { return name; }

}
```

Scopes:
- RequestScope
- SessionScoped
- ConversationScoped
- ApplicationScoped

Es gibt auch noch andere Scopes.
Recherchieren Sie welche!

http://blog.triona.de/development/jee/ein-kurzer-uberblick-uber-scopes-in-java-ee7.html

IndexController.java

```java
package at.htl.vehicle.web;

import javax.enterprise.inject.Model;

@Model
public class IndexController {

    String name;

    public IndexController() { this.name = " HTL Leonding"; }

    public String getName() { return name; }

}
```

Sehen Sie sich die Definition von Model an

http://stackoverflow.com/questions/26168312/how-is-the-bean-accessed-first-time

IndexController.java

```
failure:WELD-000072: Bean declaring a passivating scope must be passivation
capable.  Bean:  Managed Bean [class at.htl.vehicle.web.IndexController]
with qualifiers [@Default @Any @Named]. Please see server.log for more
details.
```

```java
package at.htl.vehicle.web;

import javax.faces.view.ViewScoped;
import javax.inject.Named;
import java.io.Serializable;

@Named
@ViewScoped
public class IndexController implements Serializable {

    String name;

    public IndexController() {
        this.name = " HTL Leonding";
    }

    public String getName() {
        return name;
    }

}
```

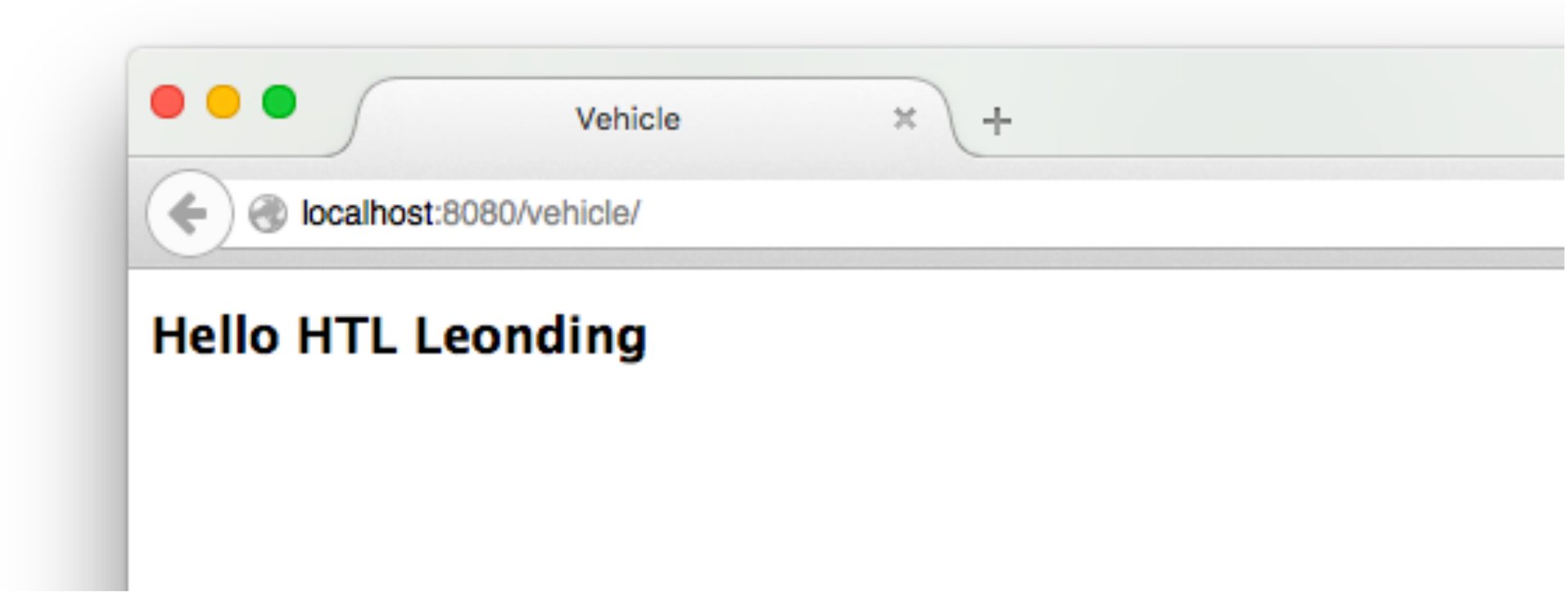
http://blog.triona.de/development/jee/ein-kurzer-uberblick-uber-scopes-in-java-ee7.html

index.xhtml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:p="http://primefaces.org/ui"
      xml:lang="en" lang="en">
<h:head>
    <title>Vehicle</title>
</h:head>
<h:body>

    <h1 class="ui-widget">Hello #{indexController.name}</h1>
    <!--
    <h1 class="ui-widget ui-widget-header ui-corner-all">Hello #{indexController.name}</h1>
    -->

</h:body>
</html>
```

Wie sieht der Output ohne Class-Attribut oder dem erweiterten (auskommentierten) Attribut aus?

Ergebnis



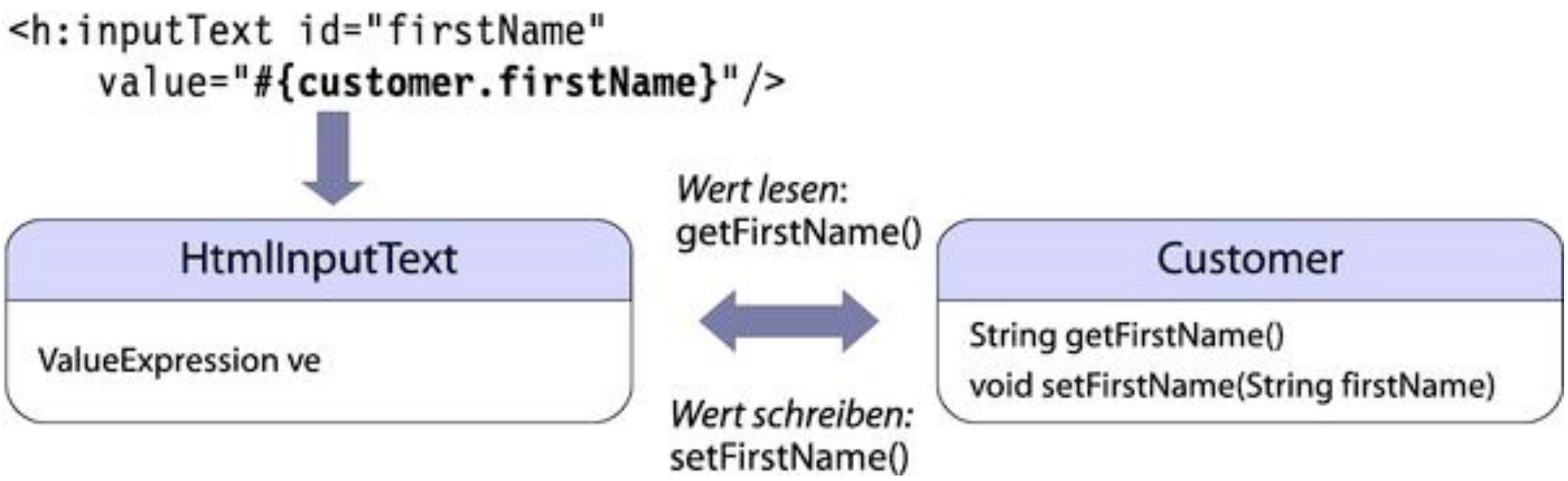
Scopes

Scope Name	Lebensdauer	Anwendungsbeispiel	Verfügbar ab Spezifikation
Singleton	Programmlaufzeit	—	CDI 1.0
Dependent	Abhängig vom aufrufender Bean	—	CDI 1.0
None	Referenz	—	JSF 1.0
Request	http-Request	GET-Requests	JSF 1.0
Session	Dauer der Anwendersession	Nutzereinstellungen	JSF 1.0
Application	Programmlaufzeit	Konstanten, locale	JSF 1.0
Conversation	Methodenaufruf	Wizards	CDI 1.0
Flow	Definitionsabhängig	Wizards	JSF 2.2
View	Bildschirmanzeige	Dynamische Formulare	JSF 2.0/JSF 2.2
Transaction	JTA-Transaktion	Transaktion	CDI 1.1
Flash	Zwei http-Requests	Redirect	JSF 2.0
Custom	Frei wählbar	Frei wählbar	JSF 2.0

http://blog.triona.de/development/jee/ein-kurzer-uberblick-uber-scopes-in-java-ee7.html
https://docs.jboss.org/weld/reference/latest/en-US/html/scopescontexts.html

Unified-EL

http://jsfatwork.irian.at/book_de/jsf.html#!idx:/jsf.html:2.5



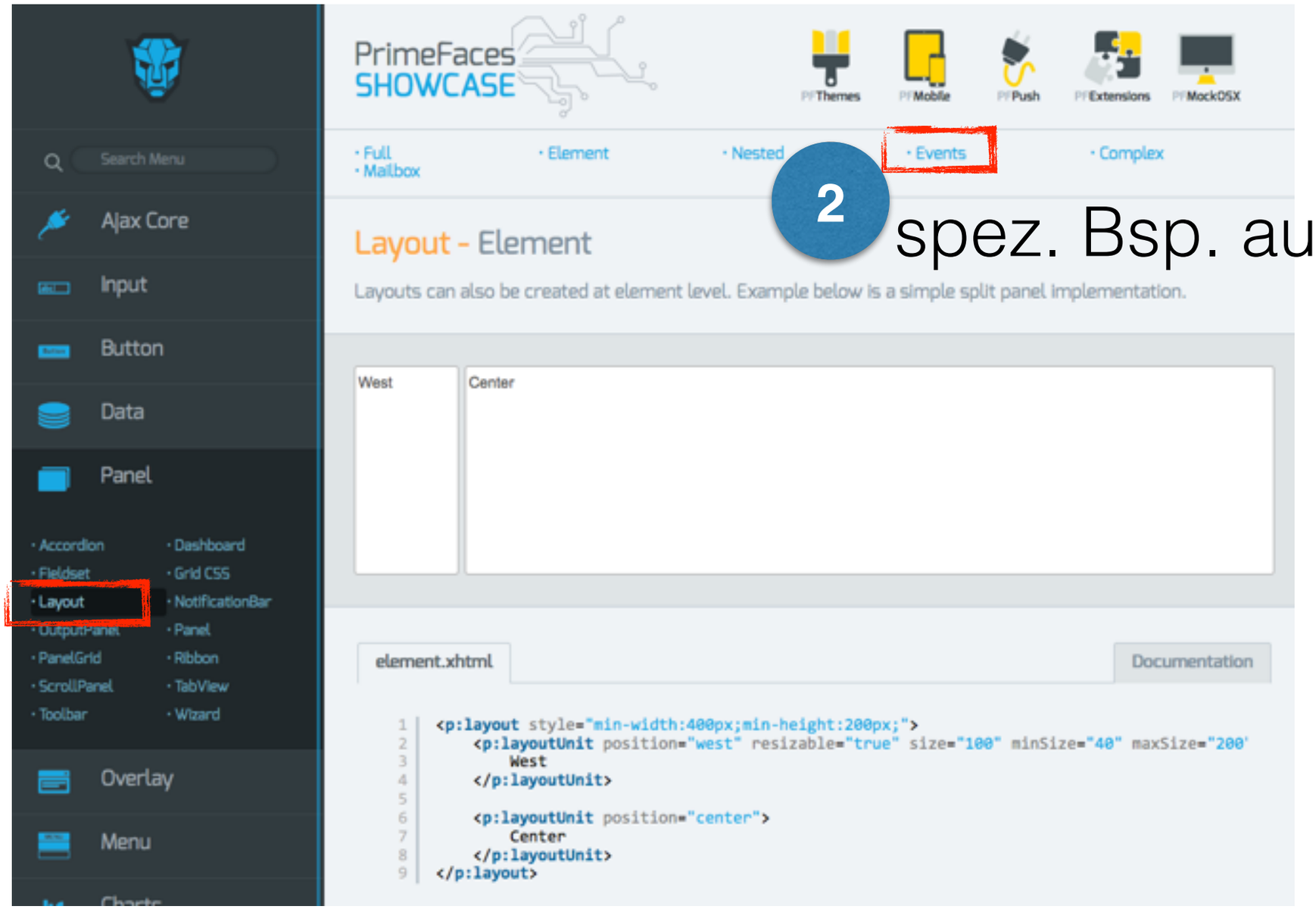
Welche Info-Quellen gibt es?

- Grundsätzlich gibt es Infos für Standard-JSF (Mojarra) und auch für das hier verwendete Primefaces
- Mojarra
  - http://jsfatwork.irian.at
  - JSR-344: http://download.oracle.com/otndocs/jcp/jsf-2_2-fr-eval-spec/index.html
- Primefaces
  - Showcase: http://www.primefaces.org/showcase/index.xhtml
  - ShowCase-Sources: https://github.com/primefaces/showcase/releases/tag/5_3
  - User-Guide: http://www.primefaces.org/documentation
  - Primefaces-Sources: https://github.com/primefaces
  - Themes: http://primefaces.org/themes
- und bei beiden natürlich Foren

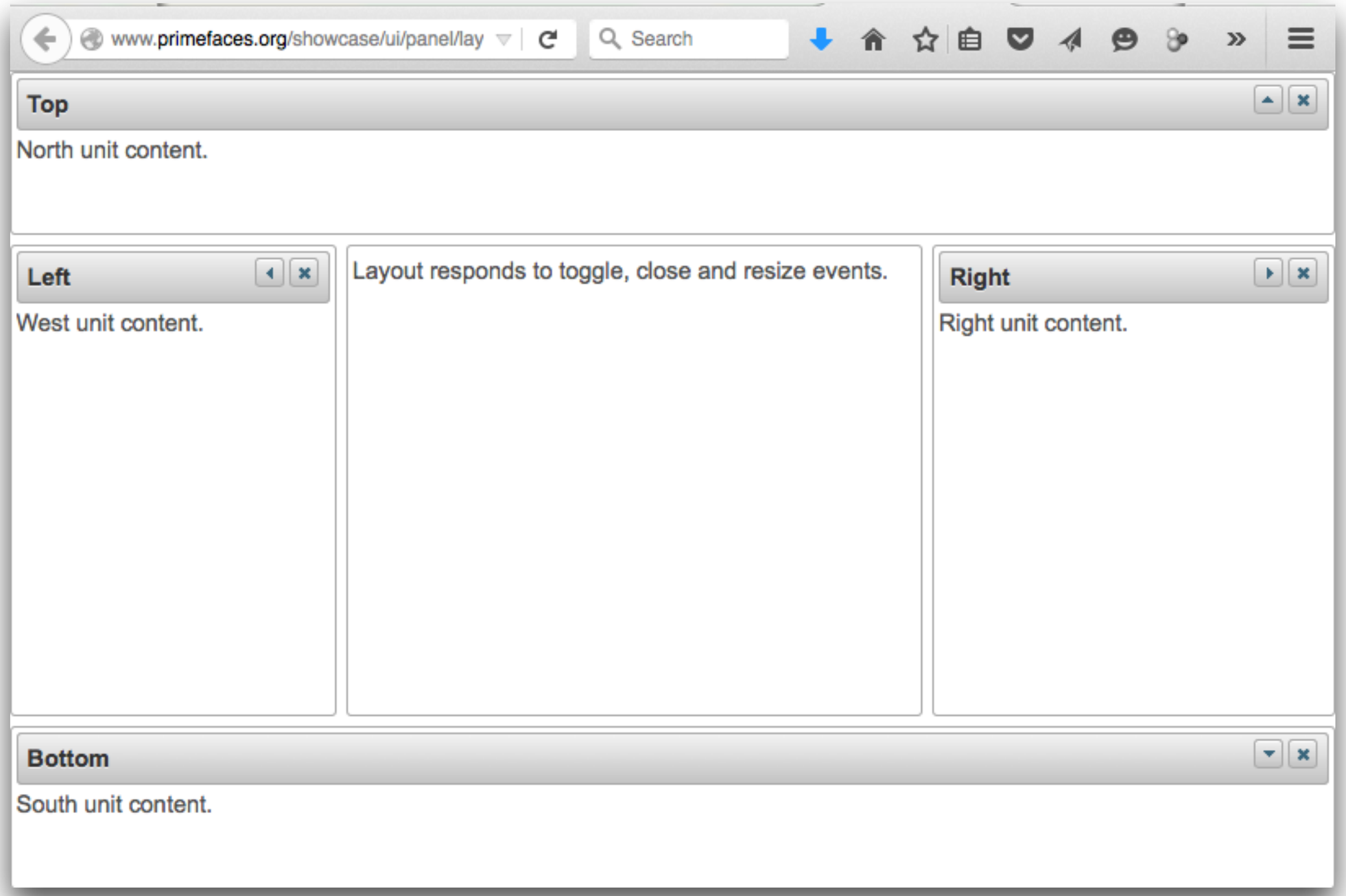
Beispiel Primefaces Layout Komponente

1 Im Showcase Komponente auswählen

2 spez. Bsp. auswählen



Bsp ansehen


www.primefaces.org/showcase/ui/panel/lay
Search
Top
North unit content.
Left
West unit content.
Layout responds to toggle, close and resize events.
Right
Right unit content.
Bottom
South unit content.

Source des Beispiels

PrimeFaces SHOWCASE

Ready For PrimeTime

PrimeFaces is a popular open source framework for JavaServer Faces featuring over 100 components, touch optimized mobilekit, push framework, client side validation, theme engine and more.

1

Download

Documentation

www.primefaces.org/downloads

Search

PRIME FACES

Overview Demos Development

Showcase

Version	Binary	Tag
5.3	showcase-5.3.war	showcase-5.3
5.2	showcase-5.2.war	showcase-5.2

2

primefaces / showcase

Releases Tags

Latest release

5.3.

cagataycivici released this 9 days ago · 3 commits to

5_3

b949a79

5_3

Set version

Downloads

3

Source code (zip)

Source code (tar.gz)

- push
- resources
- ui
  - ajax
  - button
  - chart
  - csv
  - data
  - df
  - dnd
  - file
  - input
  - menu
  - message
  - misc
  - multimedia
  - overlay
  - panel
    - accordionPanel.xhtml
    - dashboard.xhtml
    - fieldset.xhtml
    - grid.xhtml
    - layout
      - complex.xhtml
      - events.xhtml
      - mailbox.xhtml
      - nested.xhtml
      - template.xhtml
    - notificationBar.xhtml

4

Nun kann der Quellcode im eigenen Bsp verwendet werden.

simple dataTable

Bsp mit Entity Vehicle

```java
package at.htl.vehicle.entity;

public class Vehicle {

    private String brand;
    private String type;

    public Vehicle() {
    }

    public Vehicle(String brand, String type) {
        this.brand = brand;
        this.type = type;
    }

    public String getBrand() { return brand; }

    public void setBrand(String brand) { this.brand = brand; }

    public String getType() { return type; }

    public void setType(String type) { this.type = type; }

    @Override
    public String toString() { return String.format("%s %s", brand, type); }
}
```

v003

Vehicle-Repository

```java
package at.htl.vehicle.at.htl.vehicle.business;

import at.htl.vehicle.entity.Vehicle;

import javax.enterprise.context.ApplicationScoped;
import javax.inject.Singleton;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;

@ApplicationScoped
public class VehicleRepository {

    private List<Vehicle> vehicles;

    public VehicleRepository() {
        vehicles = new LinkedList<>();
    }

    public void add(Vehicle vehicle) {
        if (vehicle != null) {
            vehicles.add(vehicle);
        }
    }

    public List<Vehicle> getVehicles() { return vehicles; }
}
```

entspricht einer VehicleFacade bzw. VehicleDao allerdings werden die Fahrzeuge direkt im Repo gespeichert

InitBean.java

```java
package at.htl.vehicle.at.htl.vehicle.business;

import at.htl.vehicle.entity.Vehicle;

import javax.annotation.PostConstruct;
import javax.ejb.Singleton;
import javax.ejb.Startup;
import javax.inject.Inject;

@Startup
@Singleton
public class InitBean {

    @Inject
    VehicleRepository repo;

    public InitBean() {
    }

    @PostConstruct
    public void init() {
        repo.add(new Vehicle("Alfa Romeo", "2000 Berlina"));
        repo.add(new Vehicle("Alfa Romeo", "Spider 1967"));
        repo.add(new Vehicle("Opel", "Rekord 1700P1"));
    }

}
```

Wir brauchen auch ein paar Daten

v003

IndexController.java

```java
package at.htl.vehicle.web;

import at.htl.vehicle.at.htl.vehicle.business.VehicleRepository;
import at.htl.vehicle.entity.Vehicle;

import javax.enterprise.inject.Model;
import javax.inject.Inject;
import java.io.Serializable;
import java.util.List;

@Model
public class IndexController implements Serializable {

    @Inject
    private VehicleRepository repo;

    public IndexController() {
    }

    public List<Vehicle> getVehicles() {
        System.out.println(repo.getVehicles());
        return repo.getVehicles();
    }

}
```

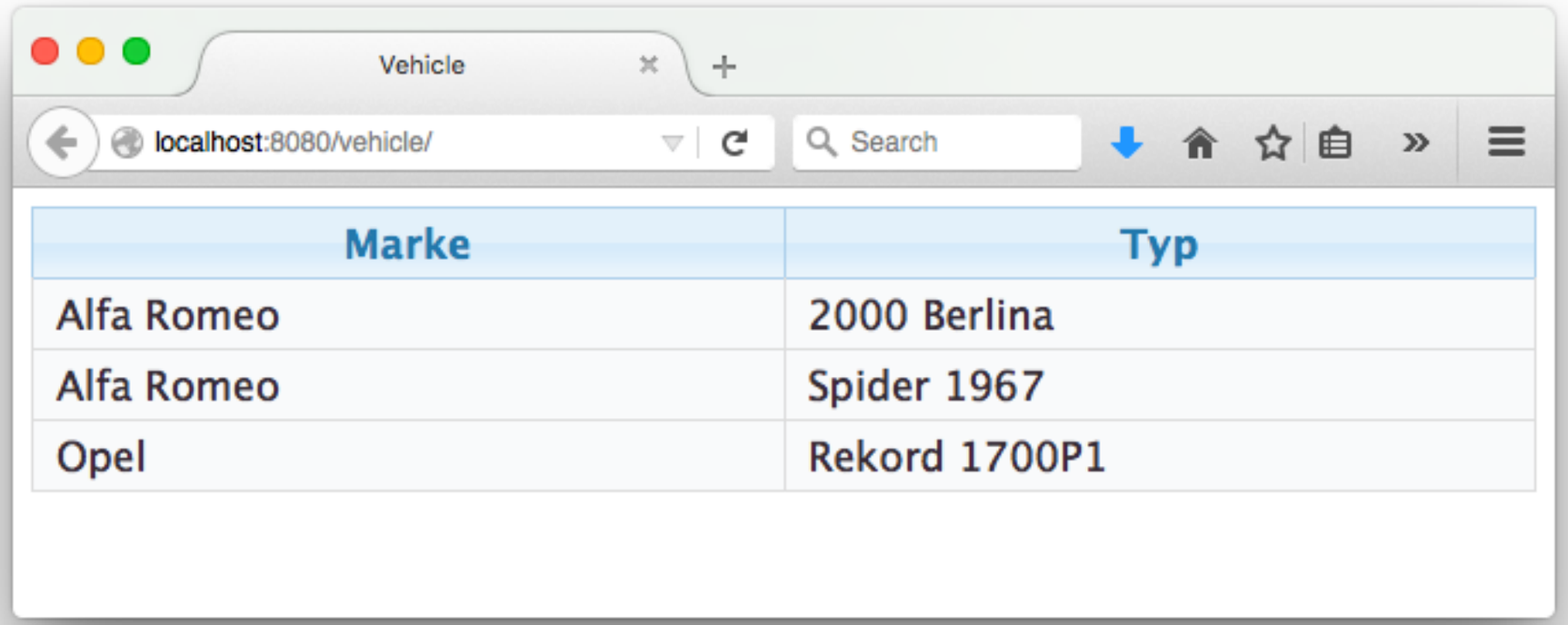
v003

index.xhtml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:p="http://primefaces.org/ui"
      xml:lang="en" lang="en">
<h:head>
    <title>Vehicle</title>
</h:head>
<h:body>

    <p:dataTable var="vehicle" value="#{indexController.vehicles}">

        <p:column headerText="Marke">
            <h:outputText value="#{vehicle.brand}" />
        </p:column>

        <p:column headerText="Typ">
            <h:outputText value="#{vehicle.type}" />
        </p:column>

    </p:dataTable>

</h:body>
</html>
```

v003

Ergebnis


Vehicle
localhost:8080/vehicle/
Search
Marke
Typ
Alfa Romeo
2000 Berlina
Alfa Romeo
Spider 1967
Opel
Rekord 1700P1

form und commandButton

IndexController.java

```java
@Model
public class IndexController implements Serializable {

    @Inject
    private VehicleRepository repo;

    private Vehicle currVehicle;

    public IndexController() { currVehicle = new Vehicle(); }

    public List<Vehicle> getVehicles() {...}

    public Vehicle getCurrVehicle() { return currVehicle; }

    public void setCurrVehicle(Vehicle currVehicle) {
        this.currVehicle = currVehicle;
    }

    public void doSaveVehicle() {
        if (currVehicle.getBrand() != null && currVehicle.getType() != null) {
            repo.add(currVehicle);
            currVehicle = new Vehicle();
            RequestContext.getCurrentInstance().reset("form_detail");
        }
    }
}
```

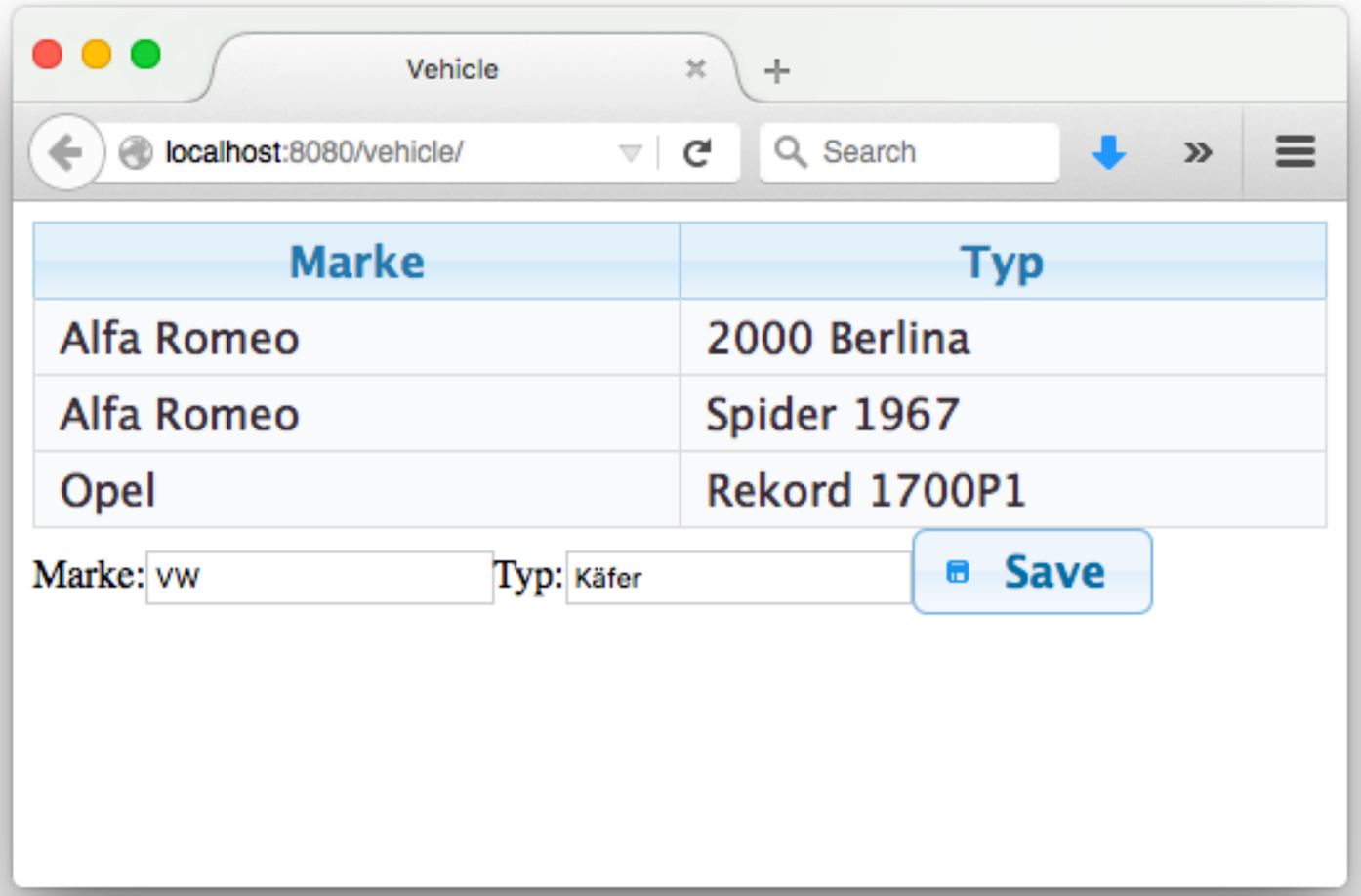
v004

index.xhtml

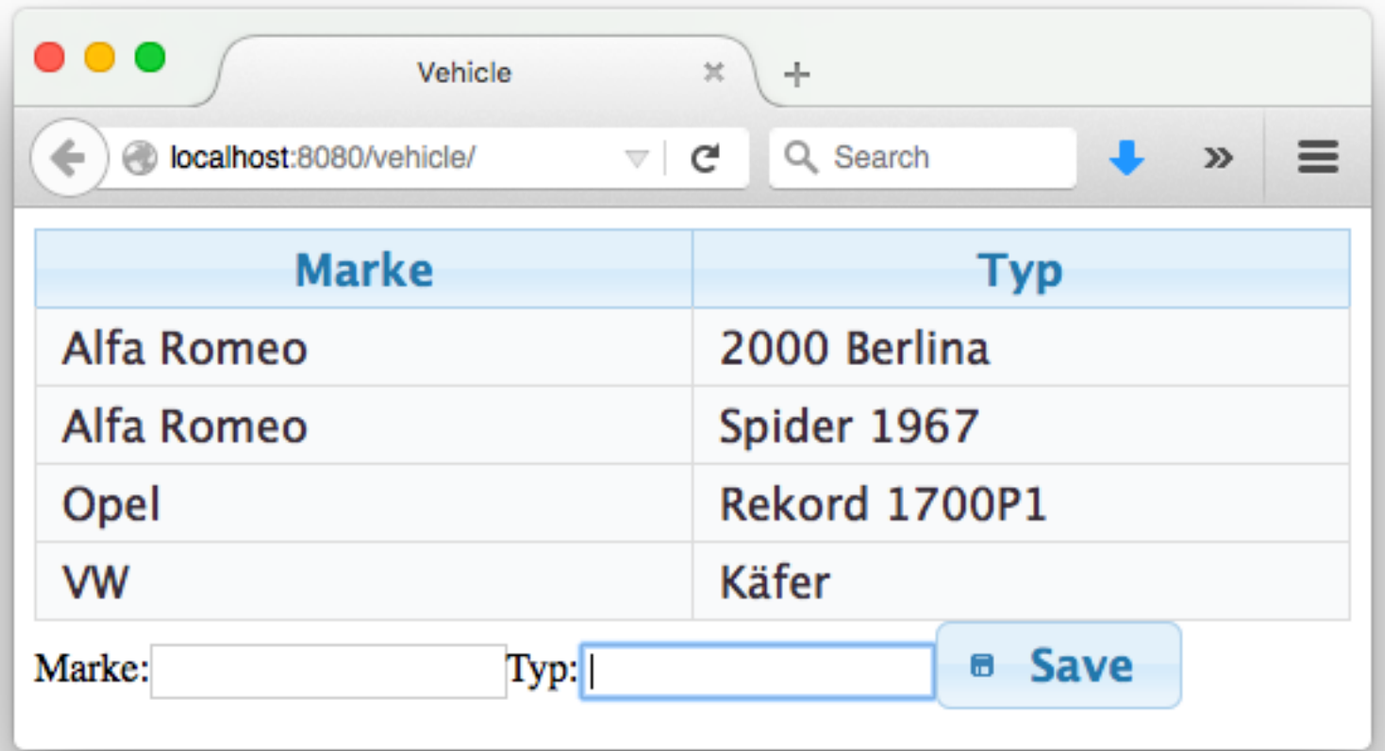
```
<h:body>

    <p:dataTable id="vehicle_table" var="vehicle" value="#{indexController.vehicles}">

        <p:column headerText="Marke">
            <h:outputText value="#{vehicle.brand}"/>
        </p:column>

        <p:column headerText="Typ">
            <h:outputText value="#{vehicle.type}"/>
        </p:column>

    </p:dataTable>
    <h:form id="form_detail">
        <h:outputLabel for="brand" value="Marke:"/>
        <h:inputText id="brand" value="#{indexController.currVehicle.brand}"/>
        <h:outputLabel for="type" value="Typ:"/>
        <h:inputText id="type" value="#{indexController.currVehicle.type}"/>
        <p:commandButton value="Save"
                         update="vehicle_table form_detail"
                         icon="ui-icon-disk"
                         actionListener="#{indexController.doSaveVehicle}"/>
    </h:form>
</h:body>
```

http://www.petefreitag.com/cheatsheets/jqueryui-icons/

Ergebnis


Vehicle
localhost:8080/vehicle/
Search
Marke
Typ
Alfa Romeo
2000 Berlina
Alfa Romeo
Spider 1967
Opel
Rekord 1700P1
Marke: VW
Typ: Käfer
Save

Ergebnis 2

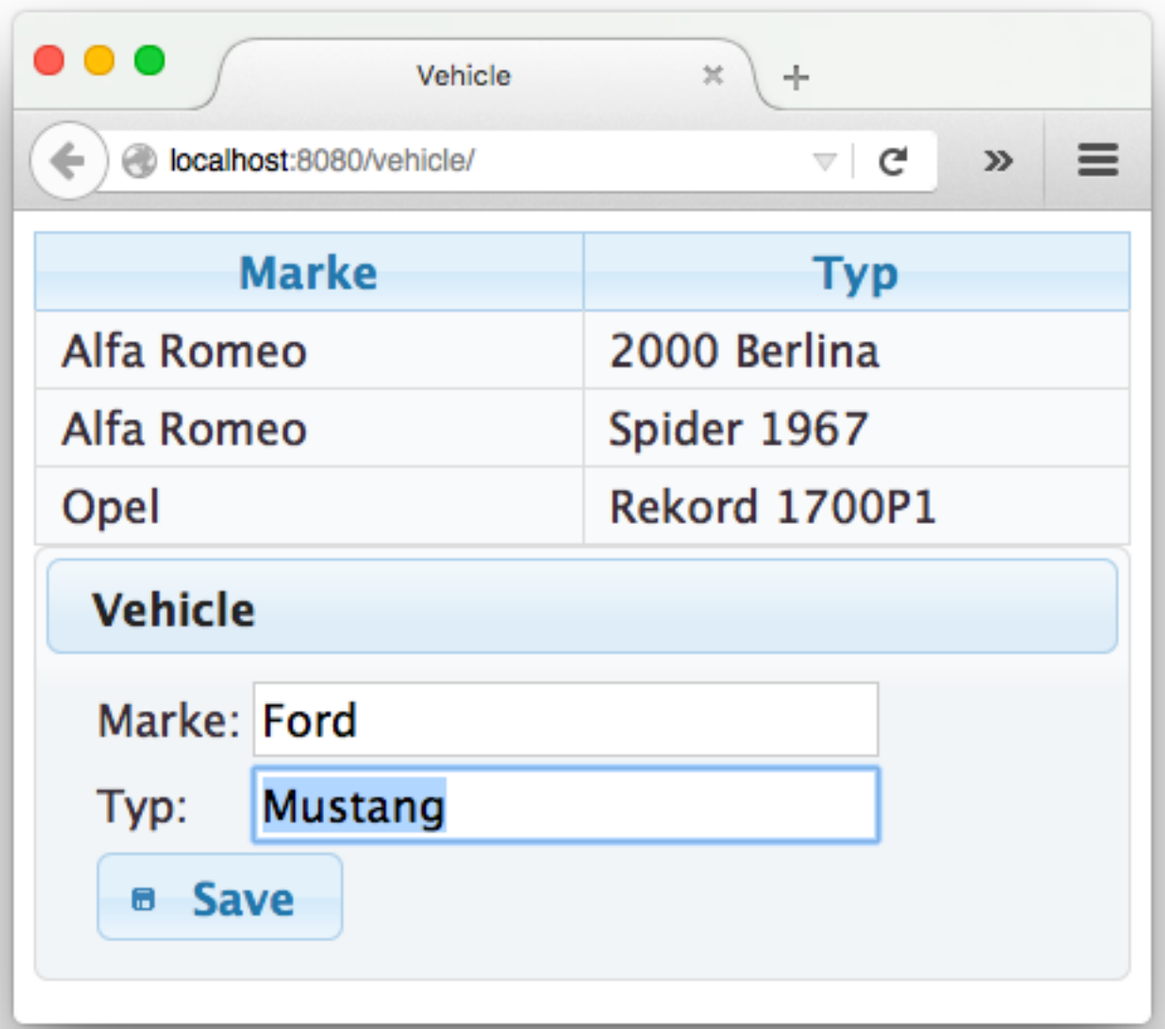

Vehicle
localhost:8080/vehicle/
Search
Marke
Typ
Alfa Romeo
2000 Berlina
Alfa Romeo
Spider 1967
Opel
Rekord 1700P1
VW
Käfer
Marke:
Typ:
Save

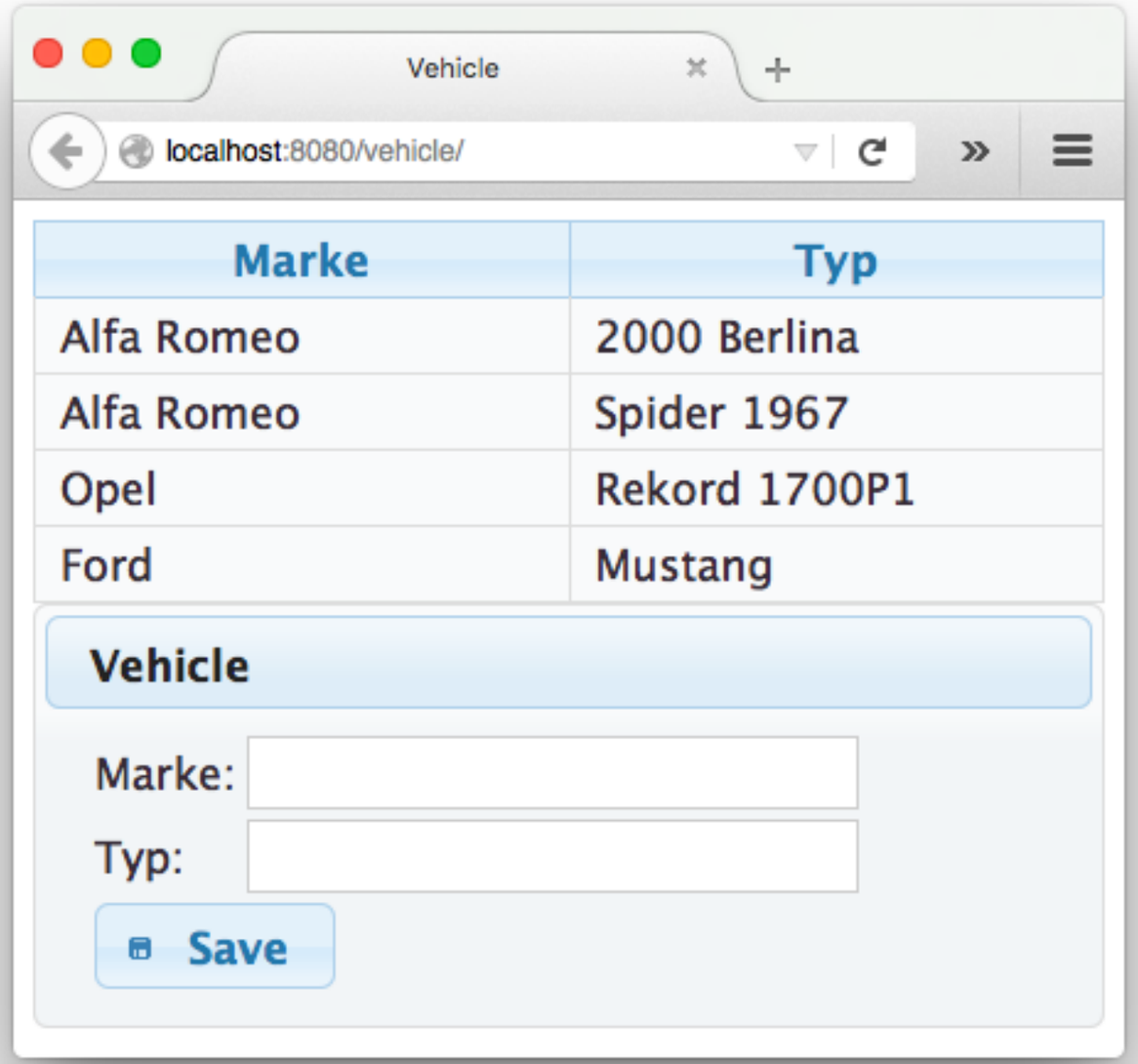
panel und panelGrid

panel und panelGrid

```
<h:form id="form_detail">
    <p:panel id="panel" header="Vehicle" style="margin-bottom: 10px;">
        <h:panelGrid id="panelgrid" columns="2">
            <h:outputLabel for="brand" value="Marke:"/>
            <h:inputText id="brand" value="#{indexController.currVehicle.brand}"/>
            <h:outputLabel for="type" value="Typ:"/>
            <h:inputText id="type" value="#{indexController.currVehicle.type}"/>
            <f:facet name="footer">
                <p:commandButton value="Save"
                                 update="vehicle_table form_detail"
                                 icon="ui-icon-disk" styleClass="ui-priority-primary"
                                 actionListener="#{indexController.doSaveVehicle}"/>
            </f:facet>
        </h:panelGrid>
    </p:panel>
</h:form>
```

Ergebnis





v004

Fragen

- Wie kann ein Bild im Dateisystem gespeichert und angezeigt werden

- Wie kann ein Bild in der DB gespeichert und angezeigt werden

Aufgabenstellung

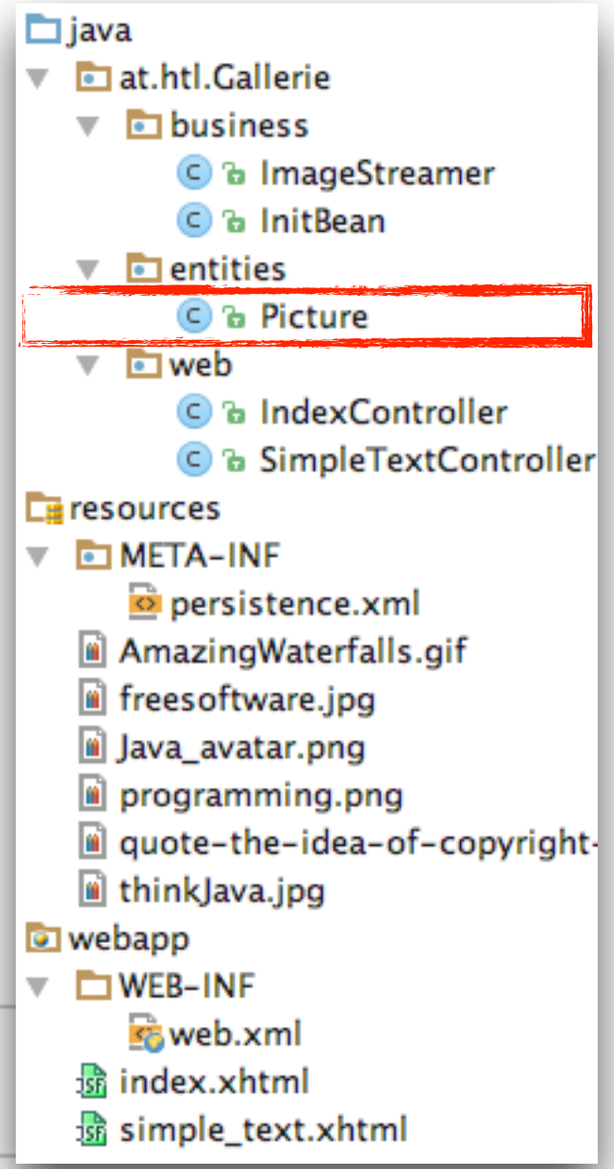
- Eine Tabelle mit Id und Blob wird erstellt, um die Bilder zu speichern

- Die Bilder werden in einer primefaces-galleria dargestellt

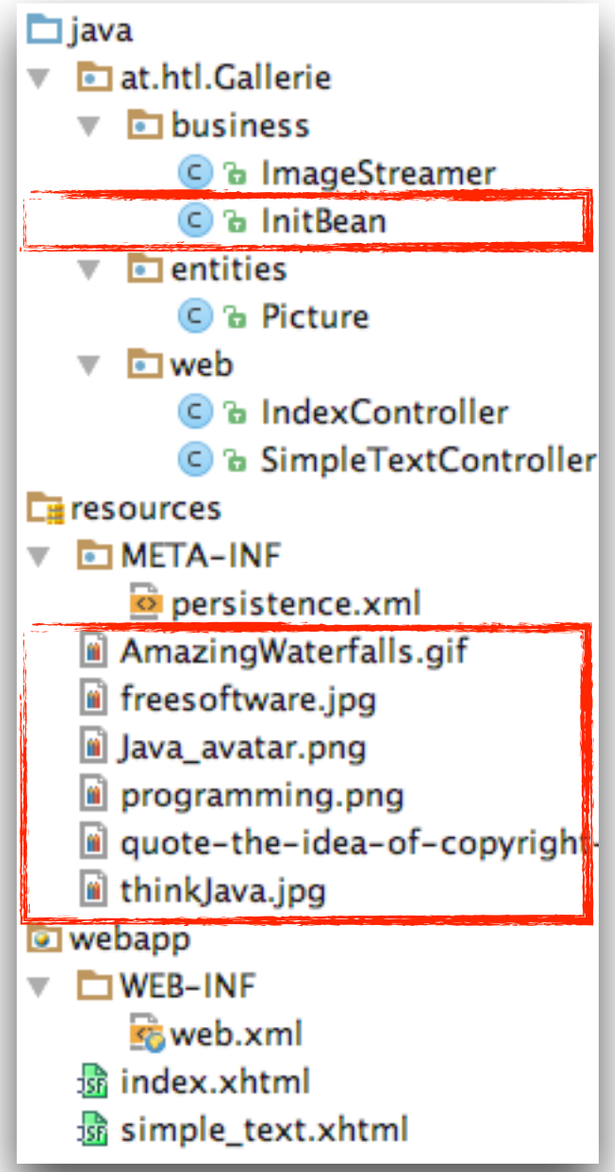
Darstellen von Bildern in JSF

Entität

```java
@Entity
@Table(name = "GA_PICTURE")
public class Picture implements Serializable{
    //region Fields
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Lob
    private byte[] image;
    //endregion

    //region Constructors
    public Picture(byte[] image) { this.image = image; }

    public Picture() {
    }
    //endregion

    //region Getters and Setters
    public Long getId() { return id; }

    public byte[] getImage() { return image; }

    public void setImage(byte[] image) { this.image = image; }
    //endregion
}
```



```java
@Startup
@Singleton
public class InitBean {
    @PersistenceContext
    EntityManager em;

    @PostConstruct
    private void init(){

        em.persist(getPicture("quote-the-idea-of-copyright-did-not-exist-in-a
        em.persist(getPicture("thinkJava.jpg"));
        em.persist(getPicture("Java_avatar.png"));
        em.persist(getPicture("freesoftware.jpg"));
        em.persist(getPicture("programming.png"));
        em.persist(getPicture("AmazingWaterfalls.gif"));

    }

    private Picture getPicture(String fileName) {
        ClassLoader classLoader = getClass().getClassLoader();
        File file = new File(classLoader.getResource(fileName).getFile());

        byte[] bFile = new byte[(int) file.length()];

        try {
            FileInputStream fileInputStream = new FileInputStream(file);
            //convert file into array of bytes
            fileInputStream.read(bFile);
            fileInputStream.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return new Picture(bFile);
    }

}
```



Zugriff auf Bilder im Ressource Ordner

HTML-Seite

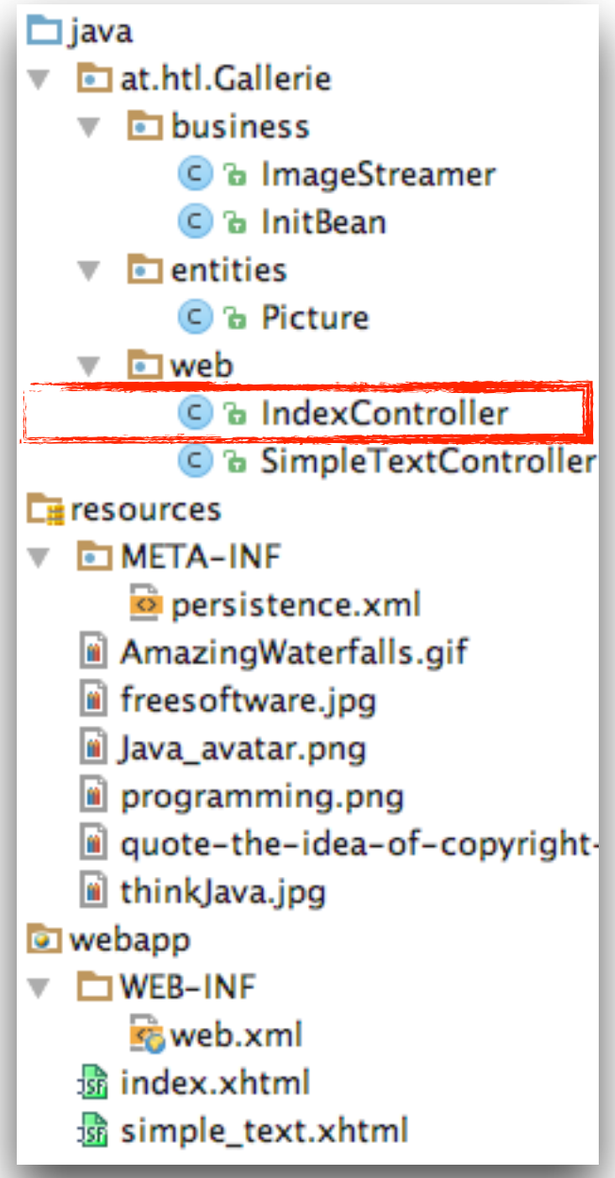
```
<p:graphicImage id="bild"
                value="#{fileUploadController.image}"
                cache="false"/>
```

Darstellung eines Bildes

```
<p:galleria id="galleria"
            value="#{fileUploadController.imageIds}"
            var="imageId"
            panelWidth="700"
            panelHeight="313"
            showCaption="false"
            effect="slide"
            effectSpeed="500">
    <p:graphicImage value="#{imageStreamer.image}"
                    title="Gallerie"
                    height="313"
                    cache="false">
        <f:param name="id" value="#{imageId}"/>
    </p:graphicImage>
</p:galleria>
```

Darstellung mehrerer Bilder in einer Gallerie unter Verwendung eines ImageStreamers

```java
@Named(value = "fileUploadController")
@SessionScoped
public class IndexController implements Serializable {

    @PersistenceContext
    private EntityManager em;

    @Resource
    private UserTransaction transaction;

    List<Picture> pictures;
    private List<Long> imageIds;

    @PostConstruct
    private void init() {
        pictures = em
                .createQuery("Select p from Picture p", Picture.class)
                .getResultList();
        imageIds = new LinkedList<>();
        try {
            for (int i = 0; i < pictures.size(); i++) {

                imageIds.add(pictures.get(i).getId());
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    //region Methods
    public StreamedContent getImage() throws IOException {...}

    public void handleFileUpload(FileUploadEvent event) {...}

    public List<Long> getImageIds() { return imageIds; }

    public void setImageIds(List<Long> imageIds) { this.imageIds = imageIds; }

    //endregion
}
```



IndexController

```
public StreamedContent getImage() throws IOException {
    return new ByteArrayContent(pictures.get(pictures.size()-1).getImage());
}
```

```
public void handleFileUpload(FileUploadEvent event) {...}
```

```
public List<Long> getImageIds() {
    return imageIds;
}
```

```
public void setImageIds(List<Long> imageIds) {
    this.imageIds = imageIds;
}
```

Um aus einer Datenbank für eine Galleria Bilder zu laden, muss man einen Kunstgriff durchführen. Eine Bean mit ApplicationScope muss an die Galleria ein Bild liefern, wobei mit <f:param> der notwendige Parameter - die ImageId - übergeben wird.

Bilder mit <p:graphicImage …> können einfach dargestellt werden (siehe HTML-Code).

```
@Named
@ApplicationScoped
public class ImageStreamer {

    @PersistenceContext
    EntityManager em;

    public StreamedContent getImage() throws IOException {
        FacesContext context = FacesContext.getCurrentInstance();

        if (context.getCurrentPhaseId() == PhaseId.RENDER_RESPONSE) {
            // So, we're rendering the view.
            // Return a stub StreamedContent so that it will generate right URL.
            return new DefaultStreamedContent();
        } else {
            // So, browser is requesting the image. Get ID value from actual request param.
            String id = context.getExternalContext().getRequestParameterMap().get("id");

            Picture picture = em
                    .createQuery("Select p from Picture p where p.id = :ID", Picture.class)
                    .setParameter("ID", Long.valueOf(id))
                    .getSingleResult();

            return new DefaultStreamedContent(new ByteArrayInputStream(picture.getImage()));
        }
    }

}
```

http://stackoverflow.com/questions/10944673/how-to-use-pgraphicimage-with-defaultstreamedcontent-in-an-uirepeat/10951201#10951201

FileUpload

web.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http:/
         version="3.1">
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>/faces/*</url-pattern>
    </servlet-mapping>
    <context-param>
        <param-name>primefaces.THEME</param-name>
        <param-value>casablanca</param-value>
    </context-param>
    <context-param>
        <param-name>primefaces.UPLOADER</param-name>
        <param-value>auto</param-value>
    </context-param>
    <welcome-file-list>
        <welcome-file>faces/index.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

pom.xml

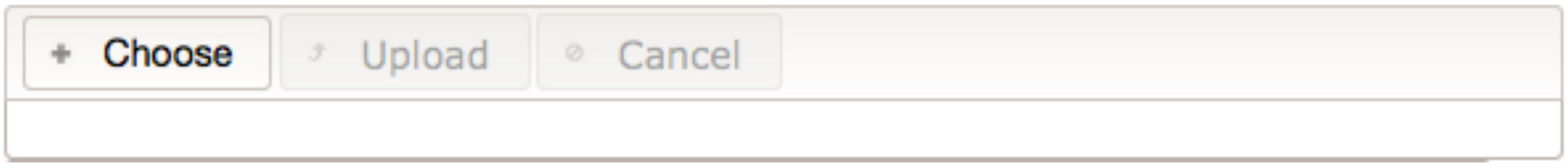
```xml
<dependencies>
    <dependency>
        <groupId>javax</groupId>
        <artifactId>javaee-web-api</artifactId>
        <version>7.0</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.primefaces</groupId>
        <artifactId>primefaces</artifactId>
        <version>5.3</version>
    </dependency>
    <dependency>
        <groupId>commons-io</groupId>
        <artifactId>commons-io</artifactId>
        <version>2.4</version>
    </dependency>
    <dependency>
        <groupId>org.primefaces.extensions</groupId>
        <artifactId>all-themes</artifactId>
        <version>1.0.8</version>
    </dependency>
</dependencies>
```

Für die Methode toByteArray(…)

HTML-Seite

Advanced Mode

```
<h:form enctype="multipart/form-data">
    <p:fileUpload mode="advanced"
                  multiple="true"
                  fileUploadListener="#{fileUploadController.handleFileUpload}">
    </p:fileUpload>
</h:form>
```

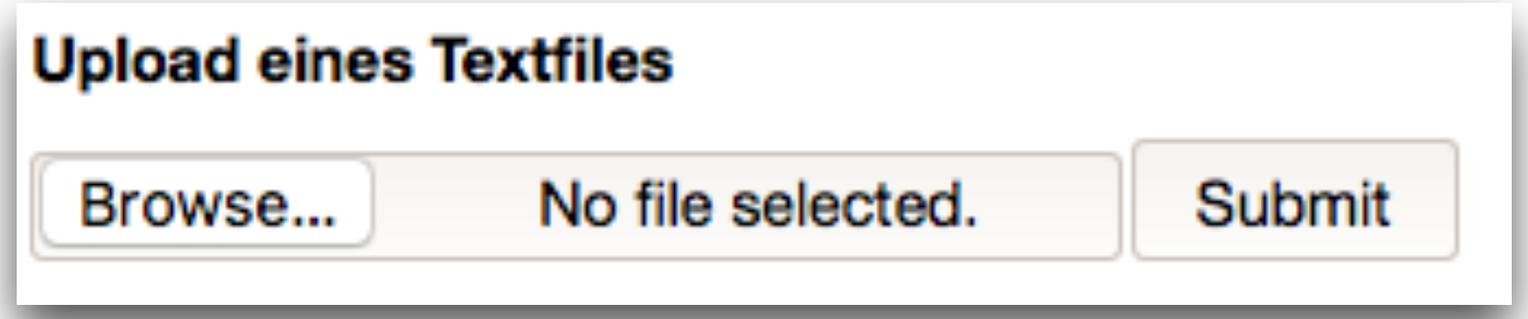


```
public void handleFileUpload(FileUploadEvent event) {
    try {
        InputStream inputStream = event.getFile().getInputstream();
        byte[] bytes = IOUtils.toByteArray(inputStream);
        transaction.begin();
        Picture newpic = new Picture(bytes);
        em.persist(newpic);
        transaction.commit();
        inputStream.close();

        // Liste der anzuzeigenden Pic aktualisieren
        pictures.add(newpic);
        imageIds.add(newpic.getId());
        System.out.println(pictures);
        System.out.println(imageIds);

        // Updaten der beiden Image-Komponenten
        FacesContext.getCurrentInstance().getPartialViewContext().getRenderIds().add("galleria");
        //FacesContext.getCurrentInstance().getPartialViewContext().getRenderIds().add("bild");
        RequestContext.getCurrentInstance().update("bild");
        /*
        //Using standard JSF API, add the client ID to PartialViewContext#getRenderIds().
        FacesContext.getCurrentInstance().getPartialViewContext().getRenderIds().add("foo:bar");

        //Using PrimeFaces specific API, use RequestContext#update().
        RequestContext.getCurrentInstance().update("foo:bar");
        */
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

FileUpload eines Text-Files

HTML-Seite

```
<h:form enctype="multipart/form-data">
    <p:growl id="messages"
             showDetail="true"/>

    <p:fileUpload value="#{simpleTextController.file}"
                  mode="simple"
                  styleClass="ui-button ui-widget ui-state-default
                              ui-corner-all ui-button-text-icon-left"
    />

    <p:commandButton value="Submit"
                     ajax="false"
                     actionListener="#{simpleTextController.upload}"
    />
</h:form>
```



```java
@Named
@RequestScoped
public class SimpleTextController {

    private String uploadedText;
    private UploadedFile file;


    public SimpleTextController() {
    }

    public void upload() {
        if(file != null) {
            FacesMessage message = new FacesMessage("Successful", file.getFileName() + " is uploaded.");
            FacesContext.getCurrentInstance().addMessage(null, message);

            uploadedText = new String(file.getContents());
            // http://stackoverflow.com/questions/1096621/read-string-line-by-line-in-java
            String[] lines = uploadedText.split(System.getProperty("line.separator"));
            for (String line : lines) {
                System.out.println(line);
            }

        }
    }

    public String getUploadedText() { return uploadedText; }

    public void setUploadedText(String uploadedText) { this.uploadedText = uploadedText; }

    public UploadedFile getFile() { return file; }

    public void setFile(UploadedFile file) { this.file = file; }
}
```

Zugriff auf den Textinhalt

Noch Fragen?