JAKARTA SERVER FACES

ZIELE

- Requestübergreifendes Management des States der UI Komponenten
- Stark typisiertes Eventmodel um serverseitig Client Events zu behandeln
- Kapselung der Unterschiede innerhalb der Clients und Browsern

- Handling der Navigation sowie Error und Exception Events
- Typ Konvertierung
- Validierung der Daten sowie entsprechendes Errorhandling

ABGRENZUNG ZU JSP

JSF

ist ein MVC Framework, welches zu dem ein Lifecycle enthält

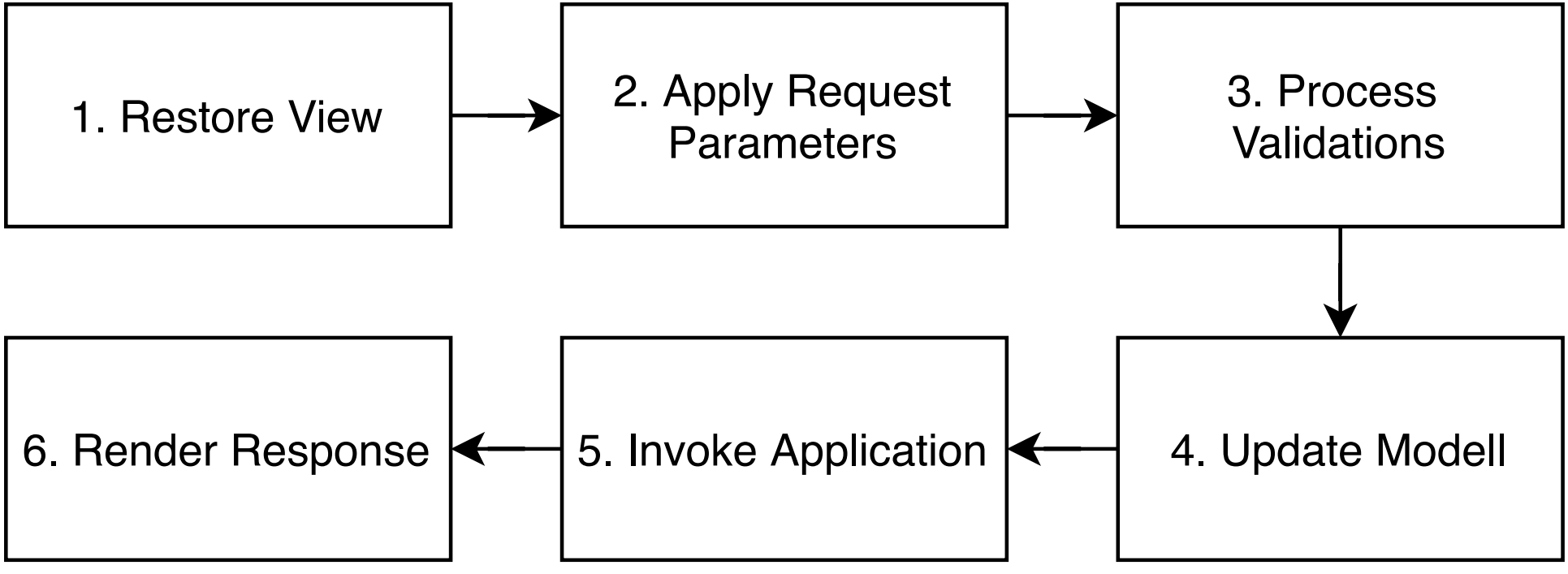
JSP

ermöglicht das dynamische Erzeugen von HTML Pages

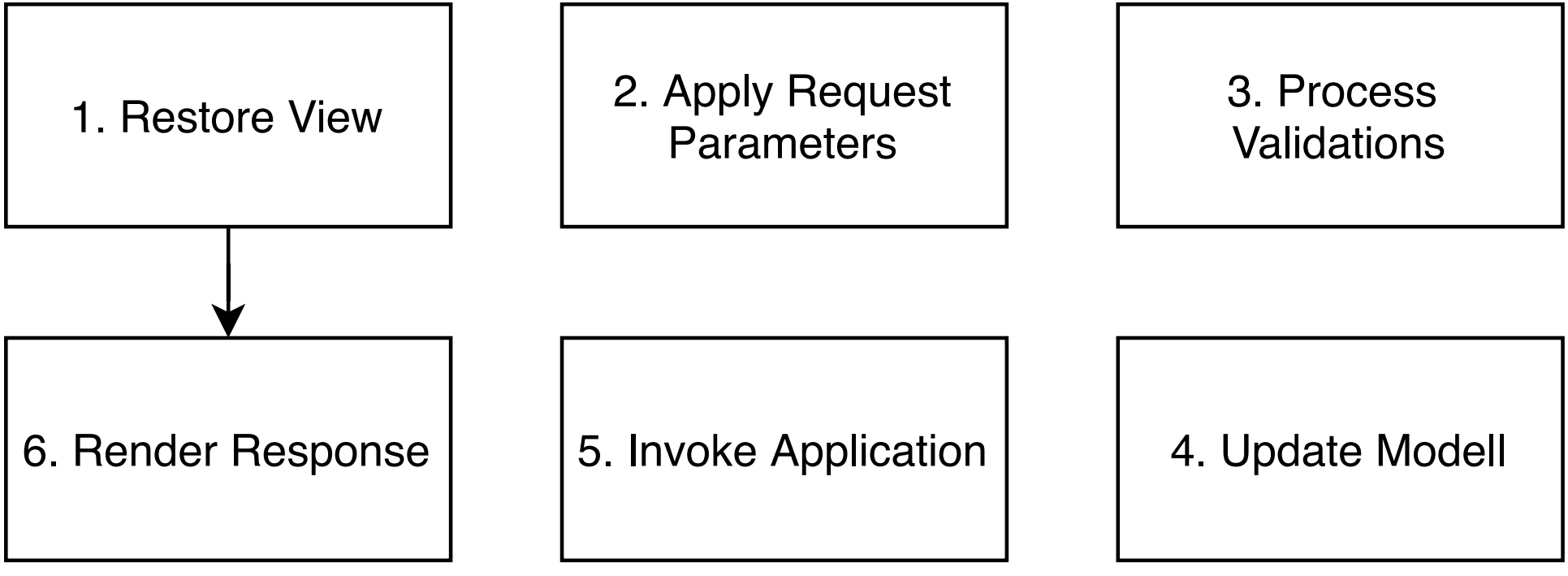
JSF KONFIGURATION

- Konfiguration in faces-config.xml im WEB-INF Ordner
- hier gibt es Definitionen zu:
  - Navigation
  - Listeners
  - Beans
  - etc.

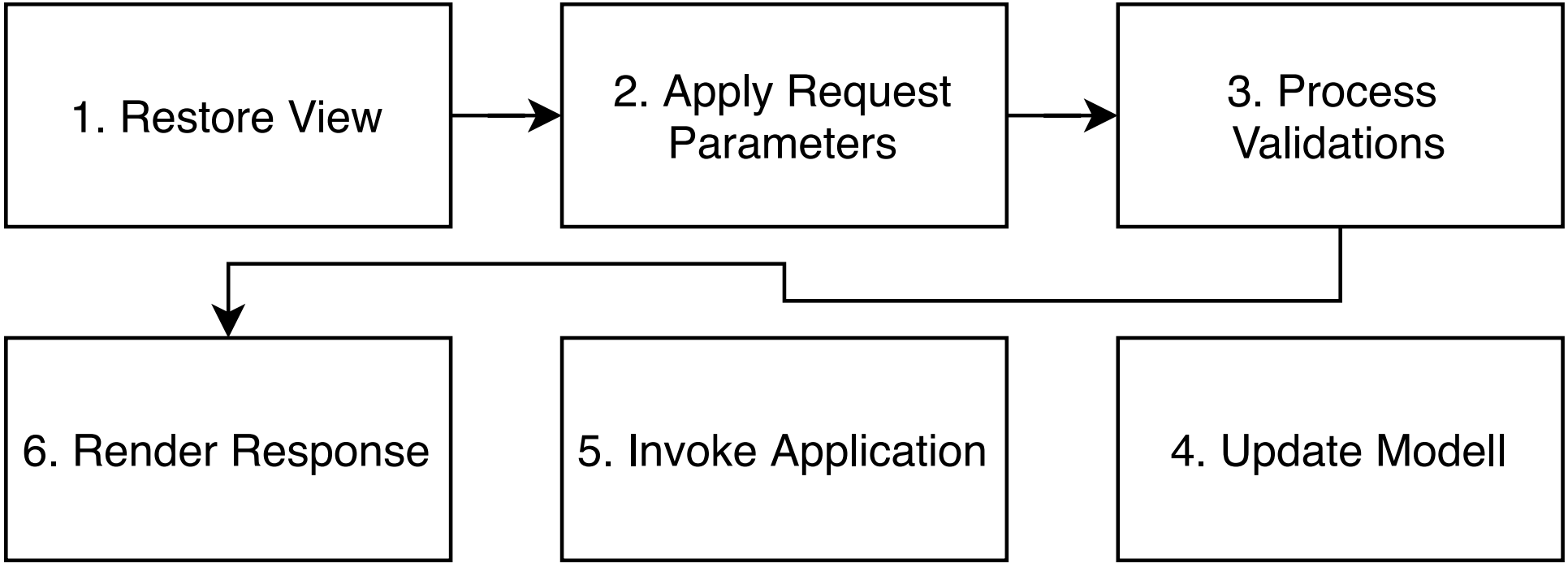
LIFECYCLE


1. Restore View
2. Apply Request Parameters
3. Process Validations
4. Update Modell
5. Invoke Application
6. Render Response

LIFECYCLE: INITIALER AUFRUF



LIFECYCLE: VALIDATION ERROR



ERSTE AUSGABE

```
<!DOCTYPE html>
<html lang="de"  xmlns="http://www.w3.org/1999/xhtml"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html">
    <head>
        <meta charset="UTF-8"/>
        <title>Hello World of JSF</title>
    </head>
    <body>
        <h:outputText value="Hello World"/>
    </body>
</html>
```

ERSTE AUSGABE

```
1  <!DOCTYPE html>
2  <html lang="de"  xmlns="http://www.w3.org/1999/xhtml"
3      xmlns:f="http://java.sun.com/jsf/core"
4      xmlns:h="http://java.sun.com/jsf/html">
5      <head>
6          <meta charset="UTF-8"/>
7          <title>Hello World of JSF</title>
8      </head>
9      <body>
10         <h:outputText value="Hello World"/>
11     </body>
12 </html>
```

ERSTE AUSGABE

```
<!DOCTYPE html>
<html lang="de"  xmlns="http://www.w3.org/1999/xhtml"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html">
    <head>
        <meta charset="UTF-8"/>
        <title>Hello World of JSF</title>
    </head>
    <body>
        <h:outputText value="Hello World"/>
    </body>
</html>
```

UNIFIED EXPRESSION LANGUAGE

- Grundsyntax

```
${expression} // immediate
#{expression} // deferred
```

LITERALE

```
${'hello ' + 'world'} // hello world
${21*2} // 42

${'\${' + '\#{'} // $ und # müssen escaped werden
```

ATTRIBUTZUGRIFF AUF BEANS

Rein lesend:

```
${bean.attribute} // => bean.getAttribute
```

ATTRIBUTZUGRIFF AUF BEANS

Lesend und schreibend:

```
#{bean.attribute} // Funktion phasenabhängig
```

- Lesender Zugriff in den Phasen: `process validations` und `render response`
- Schreibender Zugriff in den Phasen: `update Model`
- Beispiel (Two-Way-Binding, lesend+schreiben):

```
<h:InputText value="#{bean.name}"/>
```

BEANS

BEANS DEFINIEREN

- Annotations:
  - Zugriff über Klassenname mit kleinem Anfangsbuchstabe

```
1 @ManagedBean
2 @ViewScope
3 public class SayHello
```

- Faces-config.xml:
  - Zugriff über Attributwert managed-bean-name

```
1 <managed-bean>
2   <managed-bean-name>sayHello</managed-bean-name>
3   <managed-bean-class>demo.SayHello</managed-bean-class>
4   <managed-bean-scope>request</managed-bean-scope>
5 </managed-bean>
```

SCOPES

- Request Scope `RequestScoped`
- View Scope `ViewScoped`
  - während der Nutzer auf dem gleichen View bleibt
  - gilt auch über mehrere Page-Reloads
- Session `SessionScoped`
  - gilt für die Session des Nutzers
- Application Scope `ApplicationScoped`
  - Gilt über die komplette Laufzeit

BAKING BEANS

- JSF ruft die Getter und Setter in jeder Phase neu auf
  - unnötige Resourcenverschwendung
- Aufwendige Berechnungen oder Aufrufe externer Dienste aus Beans sind teuer
- Füge Baking Beans zwischen View und Logik ein
  - Pufferung und Interpretation der Daten

NAVIGATION

- explizit über in `faces-config.xml` definierte Regeln
- implizit wenn keine explizite Regel greift

NAVIGATION RULE (EXPLIZITE NAVITION)

```
 1 <navigation-rule>
 2     <from-view-id>/login.xhtml</from-view-id>
 3     <navigation-case>
 4         <from-action>#{LoginForm.login}</from-action>
 5         <from-outcome>success</from-outcome>
 6         <to-view-id>/storefront.xhtml</to-view-id>
 7     </navigation-case>
 8     <navigation-case>
 9         <to-view-id>/default.xhtml</to-view-id>
10     </navigation-case>
11 </navigation-rule>
```

NAVIGATION RULE (EXPLIZITE NAVITION)

```
<navigation-rule>
    <from-view-id>/login.xhtml</from-view-id>
    <navigation-case>
        <from-action>#{LoginForm.login}</from-action>
        <from-outcome>success</from-outcome>
        <to-view-id>/storefront.xhtml</to-view-id>
    </navigation-case>
    <navigation-case>
        <to-view-id>/default.xhtml</to-view-id>
    </navigation-case>
</navigation-rule>
```

NAVIGATION RULE (EXPLIZITE NAVITION)

```
1  <navigation-rule>
2      <from-view-id>/login.xhtml</from-view-id>
3      <navigation-case>
4          <from-action>#{LoginForm.login}</from-action>
5          <from-outcome>success</from-outcome>
6          <to-view-id>/storefront.xhtml</to-view-id>
7      </navigation-case>
8      <navigation-case>
9          <to-view-id>/default.xhtml</to-view-id>
10     </navigation-case>
11 </navigation-rule>
```

NAVIGATION RULE (EXPLIZITE NAVITION)

```
1  <navigation-rule>
2      <from-view-id>/login.xhtml</from-view-id>
3      <navigation-case>
4          <from-action>#{LoginForm.login}</from-action>
5          <from-outcome>success</from-outcome>
6          <to-view-id>/storefront.xhtml</to-view-id>
7      </navigation-case>
8      <navigation-case>
9          <to-view-id>/default.xhtml</to-view-id>
10     </navigation-case>
11 </navigation-rule>
```

NAVIGATION RULE (EXPLIZITE NAVITION)

```
<navigation-rule>
    <from-view-id>/login.xhtml</from-view-id>
    <navigation-case>
        <from-action>#{LoginForm.login}</from-action>
        <from-outcome>success</from-outcome>
        <to-view-id>/storefront.xhtml</to-view-id>
    </navigation-case>
    <navigation-case>
        <to-view-id>/default.xhtml</to-view-id>
    </navigation-case>
</navigation-rule>
```

NAVIGATION RULE (EXPLIZITE NAVITION)

```
 1 <navigation-rule>
 2     <from-view-id>/login.xhtml</from-view-id>
 3     <navigation-case>
 4         <from-action>#{LoginForm.login}</from-action>
 5         <from-outcome>success</from-outcome>
 6         <to-view-id>/storefront.xhtml</to-view-id>
 7     </navigation-case>
 8     <navigation-case>
 9         <to-view-id>/default.xhtml</to-view-id>
10     </navigation-case>
11 </navigation-rule>
```

IMPLIZITE NAVIGATION

Greift keine Navigation Regel dann wird der Rückgabewert der jeweiligen Action als relative URL genutzt.

ZUSÄTZLICHE INTERFACES

- Converters
- Events and Listeners
- Validators

COMMON EVENT PROCESSING

EVENT TYPES

- Application Events
  - ActionEvent
  - ValueChangeEvent
- System Events

APPLICATION EVENTS

- extends `jakarta.faces.event.FacesEvent`
- enthält immer die triggernde Komponenten
- werden an eine Lifecycle Phase gebunden
- können gequeued werden

LISTENER CLASSES APPLICATION EVENTS

- für jeden Application Event Type eine Listener
  - ActionListener
  - ValueChangeListener

SYSTEM EVENTS

- extends `SystemEvent`
- alle spezifizierte Events unter `jakarta.faces.event`
- Beispiel: `PostConstructApplicationEvent`

COMPONENT SYSTEM EVENTS

- spezielle System Events
- beinhalten einen Verweis auf eine Komponente
- Beispiel: `PreRenderViewEvent`

LISTENER CLASSES

- nur zwei Listener Classes nötig
- für System Events:
  `jakarta.faces.event.SystemEventListener`
- für Component System Events:
  `jakarta.faces.event.ComponentSystemEventListene`

VALIDATOREN

- Validatoren für Beans und Input Fields
- Standardvalidatoren für gängige Datentypen
  - einfache Einschränkungen für Standard-Datentypen
- eigene komplexe Validatoren möglich

STANDARDVALIDATOREN

```
<f:validateLength minimum="6" maximum="15"/>
<f:validateLongRange minimum="6" maximum="15"/>
<f:validatePattern pattern="<myPattern>"/>
```

VERWENDUNG

```
<h:inputText value="#{bean.variable}"
             validator="validatorMethod()" />
<h:inputText value="#{bean.variable}">
    <f:validateRequired/>
</inputText>
```

KONVERTER

- Validatoren für Beans und Input Fields
- Konverter für gängige Datentypen
  - BooleanConverter, BigIntegerConverter, DateTimeConverter etc.

VERWENDUNG

- wird automatisch zugeordnet durch den Typ der Beanvariable

```
<h:inputText value="#{bean.variable}"/>
```

JSF-TAGS

FORM

```
<h:form id="id">
    ...
</h:form>
```

CHECKBOX

```
<h:selectBooleanCheckbox value="#{item.done}"
                         disabled="true" />
```

INPUT TEXT

```
<h:inputText value="#{item.title}"/>
```

INPUT HIDDEN

```
<h:inputHidden value="#{item.title}"/>
```

COMMANDBUTTON

```
<h:commandButton value="Speichern"
                 action="#{manager.save()}"/>
```

REQUIREMENTS

- JDK
- Servlet-Container (bspw. Apache Tomcat)

PRAXIS

- erstellen Todo Listen Anwendung

REQUIREMENTS

- Todo Liste
- erkennbar welche Todo bereits abgeschlossen
- Todos sollen hinzugefügt und editiert werden können
- Todos enthalten ID, Titel und Description