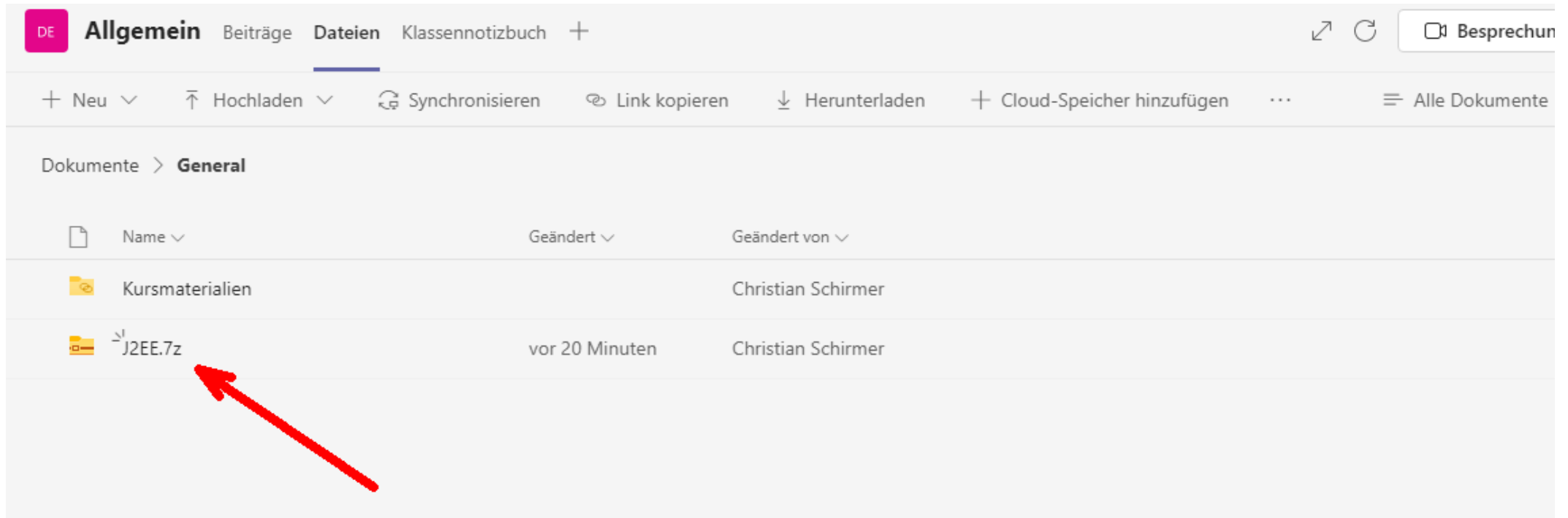




Java EE 7 Application Developer

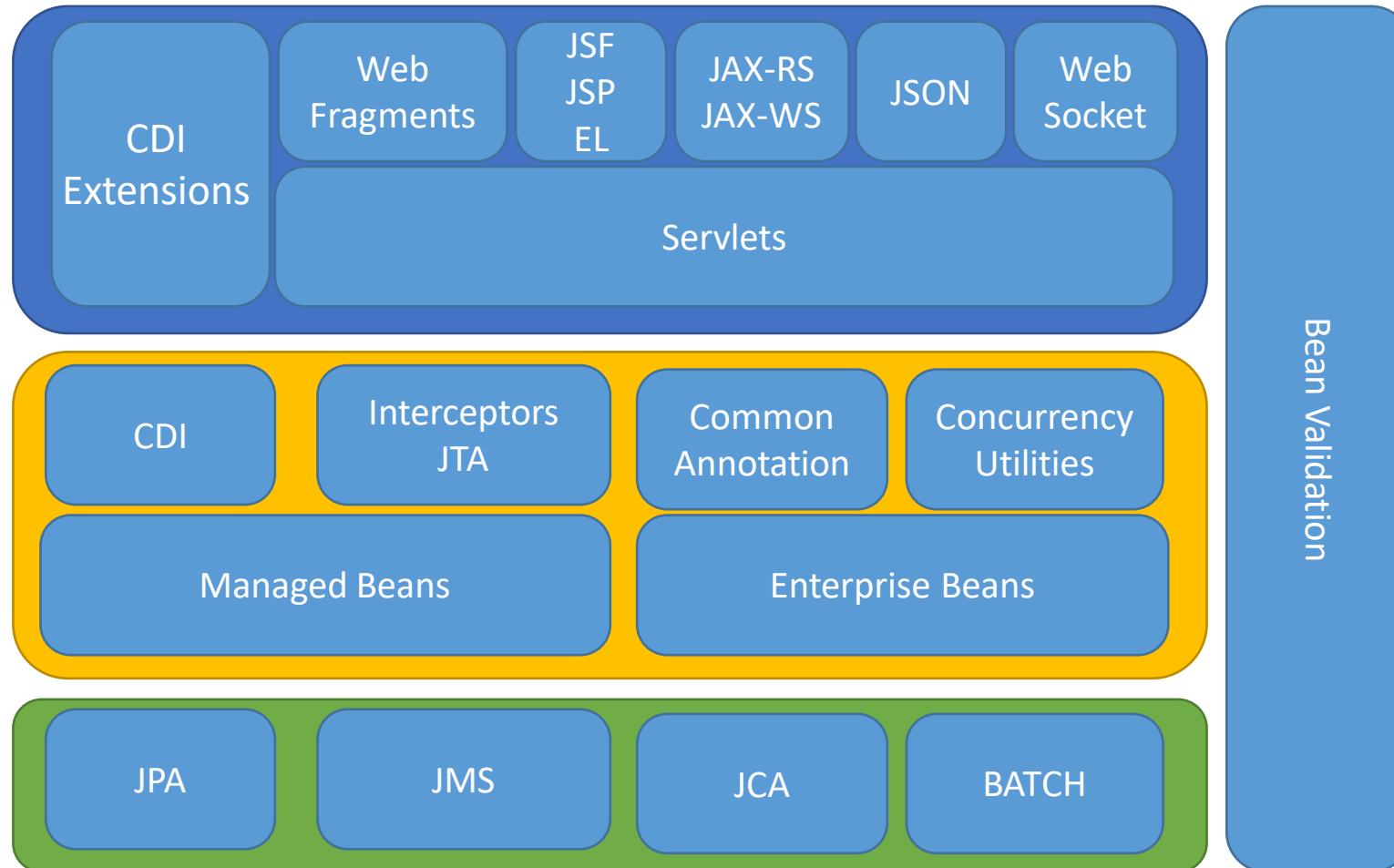
Vorbereitung

- Bitte die Datei „J2EE.7z“ von den Dateien in Teams herunterladen.
- Legen Sie den Ordner C:\J2EE an und verschieben Sie die Datei in dorthin
- Lassen Sie die Datei in den Ordner entpacken
 - „Hier entpacken“ bei 7Zip auswählen.



Java EE Komponenten Übersicht

Synonyme : Tier – Schicht – Ebene



Die verschiedenen Komponenten lassen sich logisch in drei Ebenen unterteilen:

- Backend-Tier
- Middle-Tier
- Web-Tier

Ab und an wird auch von einem

- Client-Tier gesprochen

Dies ist nur eine logische Darstellung, und die Komponenten können je nach den Anforderungen der Anwendung auf eine andere Ebene beschränkt werden.

J2EE unterschiedliche Profile

Web Profile required components

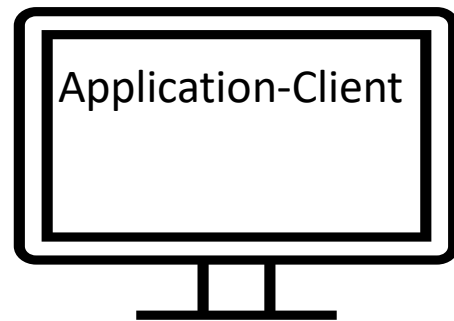
- Servlet 3.1
- JavaServer Pages (JSP) 2.3
- Expression Language (EL) 3.0
- Debugging Support for Other Languages (JSR-45) 1.0
- Standard Tag Library for JavaServer Pages (JSTL) 1.2
- JavaServer Faces (JSF) 2.2
- Java API for RESTful Web Services (JAX-RS) 2.0
- Java API for WebSocket (WebSocket) 1.0
- Java API for JSON Processing (JSON-P) 1.0
- Common Annotations for the Java Platform (JSR-250) 1.2
- Enterprise JavaBeans (EJB) 3.2 Lite
- Java Transaction API (JTA) 1.2
- Java Persistence API (JPA) 2.1
- Bean Validation 1.1
- Managed Beans 1.0
- Interceptors 1.2
- Contexts and Dependency Injection for the Java EE Platform 1.1
- Dependency Injection for Java 1.0

Full Profile

- Web Profile
- Enterprise JavaBeans (EJB) 3.2 – Full
- JMS 2.0
- JavaMail 1.5
- Connector 1.7
- Web Services 1.4
- JAX-WS 2.2
- Concurrency Utilities 1.0
- Batch 1.0
- JAXB 2.2
- Java EE Management 1.1
- JACC 1.5
- JASPIC 1.1
- JSP Debugging 1.0
- Web Services Metadata 2.1

Das Web Profile ist ein
SubSet vom Full Profile

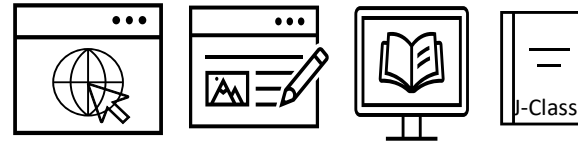
Clients



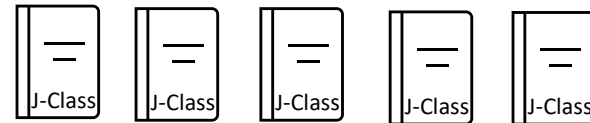
Server / Dienste

Applikationsserver

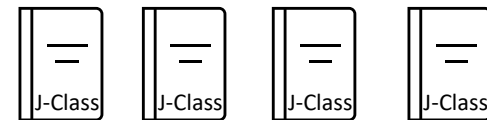
Web



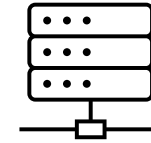
CDI



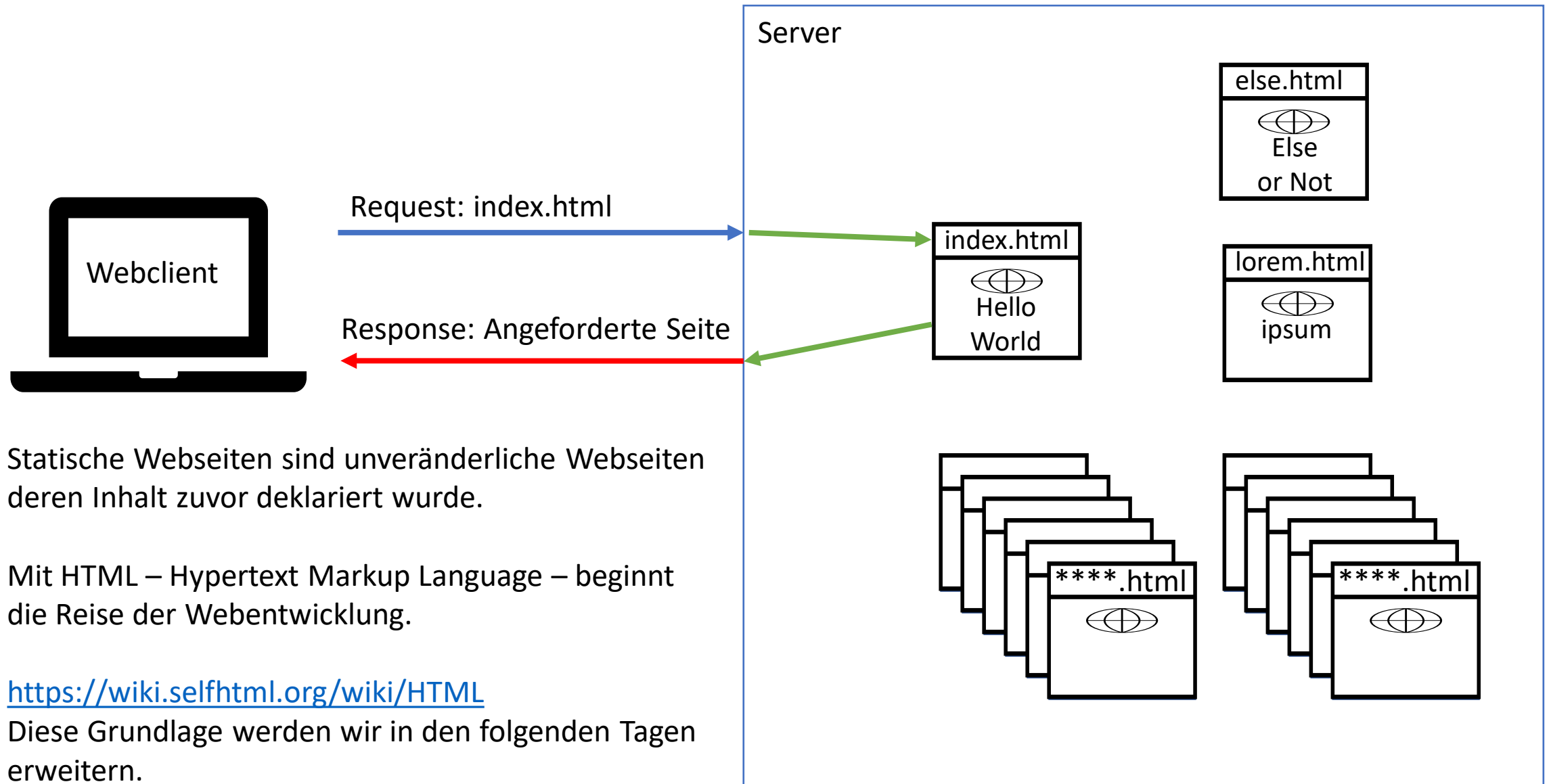
EJB



Datenbanken /
andere Systeme



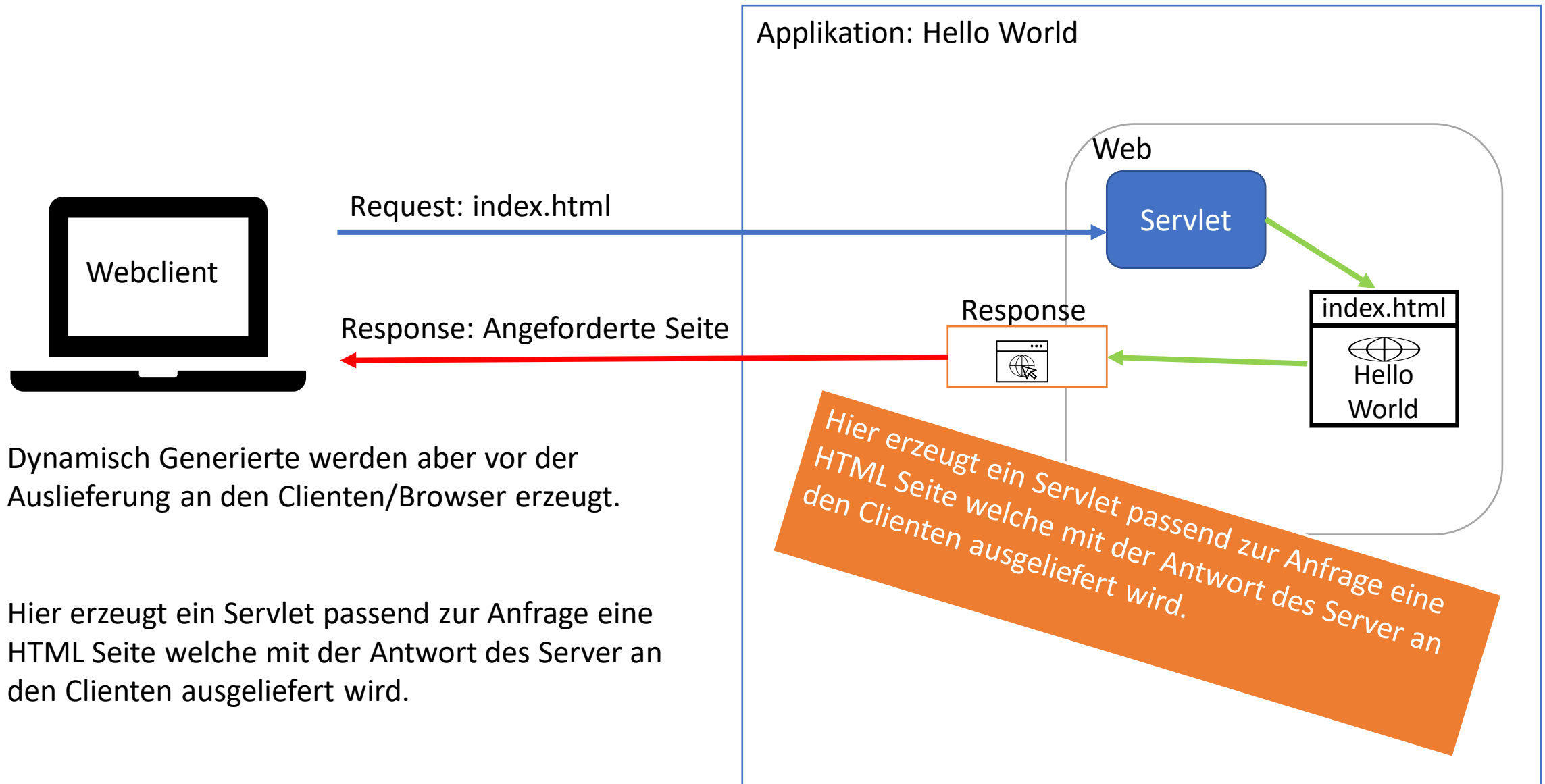
Klassisches / Statisches HTML



Webtechnologie

- Java Servlets
 - Kann zur Erzeugung einer View verwendet werden, jedoch ist der Einsatz eher im Controlling Bereich.
- Java Server Pages
 - Nicht mehr Aktuell, aus Kompatibilitätsgründen weiterhin unterstützt.
 - Logik wird entweder per Java-Code in speziellen Tags oder JavaBeans integriert.
- Java Server Faces
 - Facelets auf Basis von XHTML. JSF beinhaltet ein UI-Komponentenmodell mit Ereignisverarbeitung.
 - Logik wird durch ManagedBeans oder CDI und der Ereignisverarbeitung sowie über spezielle Tags integriert.

Dynamisch Generiertes HTML



Anmerkung

- Die Antwort eines Server muss keine HTML Datei sein.
- Textdateien
 - Werden meist vom Browser direkt angezeigt.
 - JSON – HTML – XML – CSV - TXT
- Binärdateien
 - Diese Dateien werden als Download angeboten und können auf dem eigenen System gespeichert werden.
 - MP3 – AVI – Archive – Programme
- Dynamisch Generiertes HTML ≠ Dynamisches HTML
 - Dynamische Generiert bedeutet von einem Interpreter erstellt.
 - Dynamisches HTML
 - https://de.wikipedia.org/wiki/Dynamisches_HTML

GET

- Mit beidem können Informationen an die Webseite gesendet werden.
- GET
 - Speicherung im Cache des Browsers möglich
 - Erhaltung in der Browser-Historie
 - Speicherung als Lesezeichen möglich
 - Keine Eignung für die Verwendung mit sensiblen Daten
 - Einschränkung der Länge (URL Länge)
 - In erster Linie ist ihr Zweck, Daten einzuholen
- POST VS GET
 - GET
 - Name-Wert-Paare werden in einem mit „&“ getrennten Datensatz zusammengefasst.
 - Der Datensatz wird an die URL angehängen
 - Post
 - Die Daten werden beim Absenden des Formulars mit einer POST-Anfrage an den Server geschickt. Der Datensatz wird anders als beim GET nicht an die URL angehängen, sondern an den Header der Anfrage angehängen.

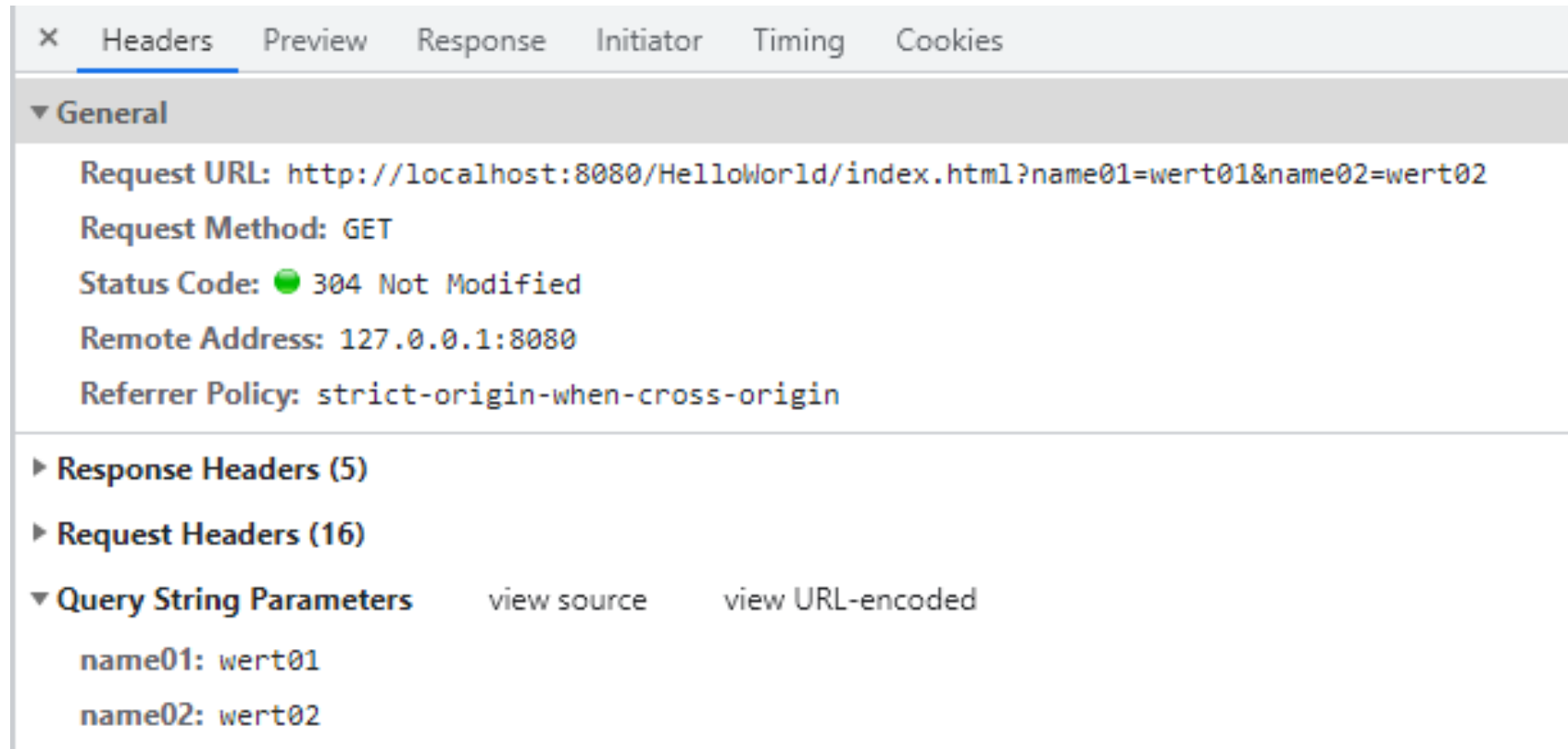
GET

- GET

- Beim GET sind die Daten Bestandteil der URL.
 - Vorteil: Wenn der Link gespeichert wird, sind auch die Daten gespeichert.
 - Nachteil: Daten sind in der URL im Klartext sichtbar und können in der Browserhistorie eingesehen werden.
- URL mit Daten in GET
 - `http://localhost:8080/HelloWorld/index.html?name01=wert01&name02=wert02`
 - Die Parameter erfolgen als „Name Werte“ Paar. Mehrere Parameter werden mit dem & zusammengesetzt.

GET

- GET
 - Header einer GET Anfrage



POST

- POST

- Daten in einer POST Anfrage

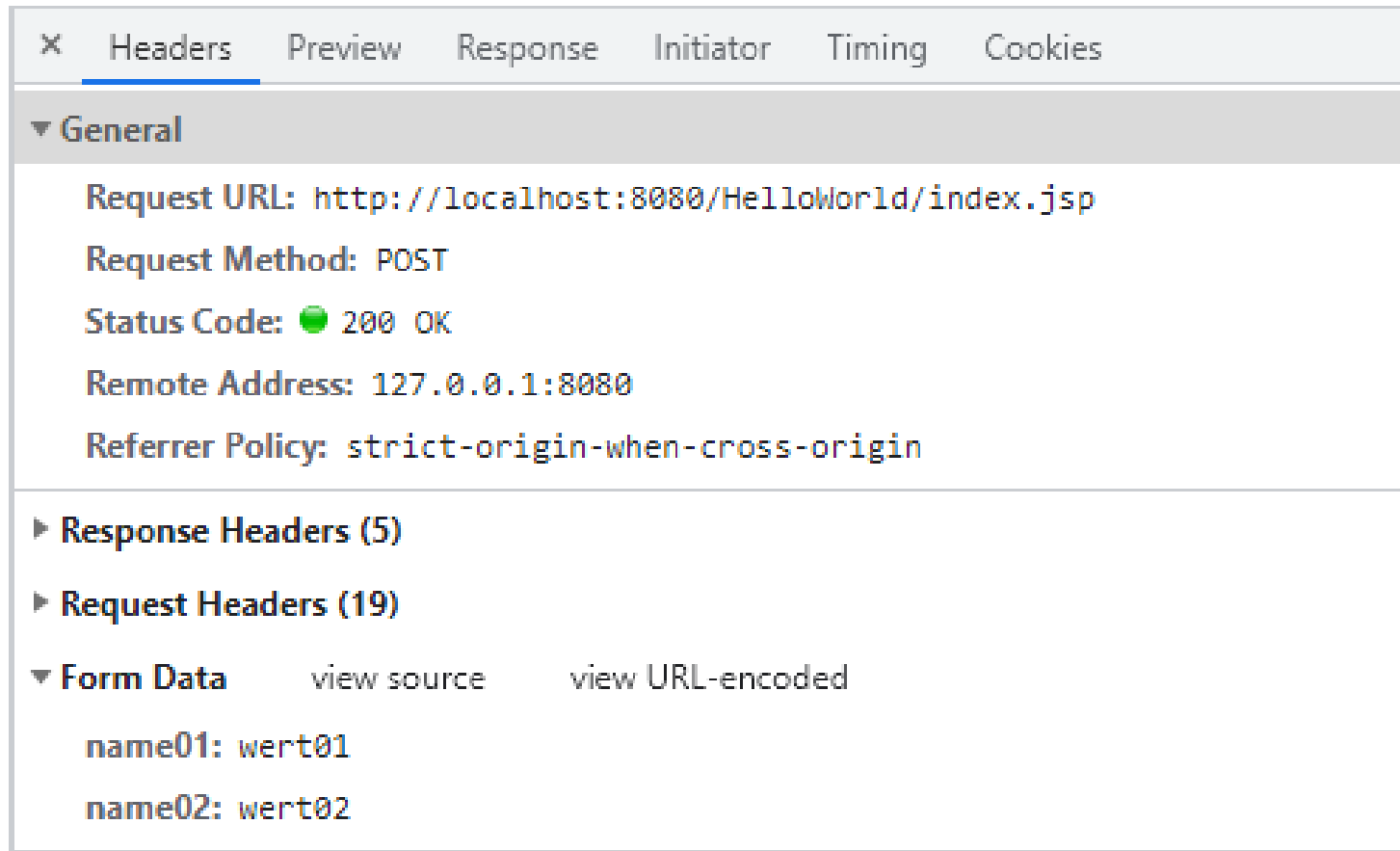
- Name-Wert-Paare werden in einem mit „&“ getrennten Datensatz zusammengefasst. Die Daten werden beim Absenden des Formulars mit einer POST-Anfrage an den Server geschickt. Der Datensatz wird anders als beim GET nicht an die URL angehängen, sondern an den Header der Anfrage angehängen.

- URL mit Daten in POST

- `http://localhost:8080/HelloWorld/index.jsp`
 - Die Daten sind in der URL nicht mehr anhalten.
 - Daten werden nicht in der Browserhistorie angezeigt.
 - Links und Lesezeichen enthalten keine möglichen Sensiblen Daten
 - Auch zum senden von Dateien geeignet

POST

- POST
 - Header einer POST Anfrage



The screenshot displays the 'Headers' tab in a web browser's developer tools. The 'General' section is expanded, showing the following details:

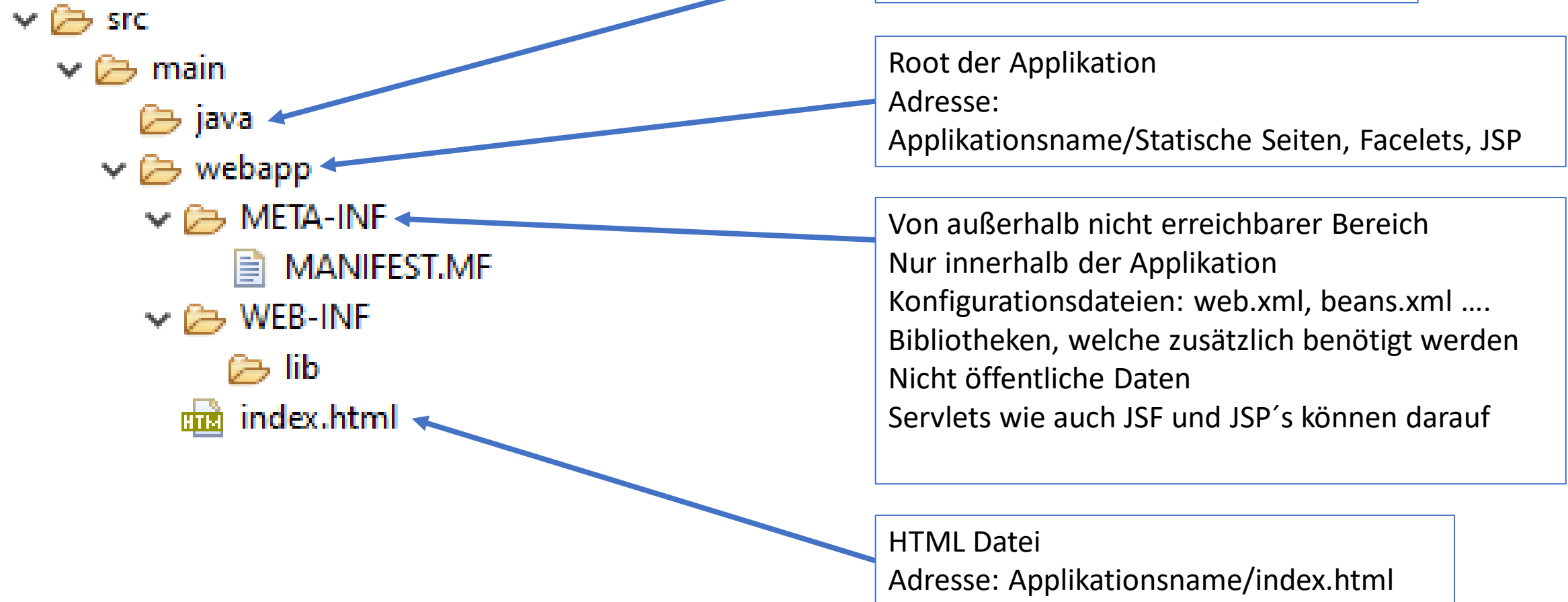
- Request URL:** `http://localhost:8080/HelloWorld/index.jsp`
- Request Method:** `POST`
- Status Code:** ● `200 OK`
- Remote Address:** `127.0.0.1:8080`
- Referrer Policy:** `strict-origin-when-cross-origin`

Below the 'General' section, there are expandable sections for 'Response Headers (5)', 'Request Headers (19)', and 'Form Data'. The 'Form Data' section is currently expanded, showing two entries:

- name01:** `wert01`
- name02:** `wert02`

JBoss und WildFly unterstützen kein POST für HTML Seiten. Daher wurde hier schon JSP eingesetzt.

Struktur eines Webarchivs (WAR) in J2EE 7

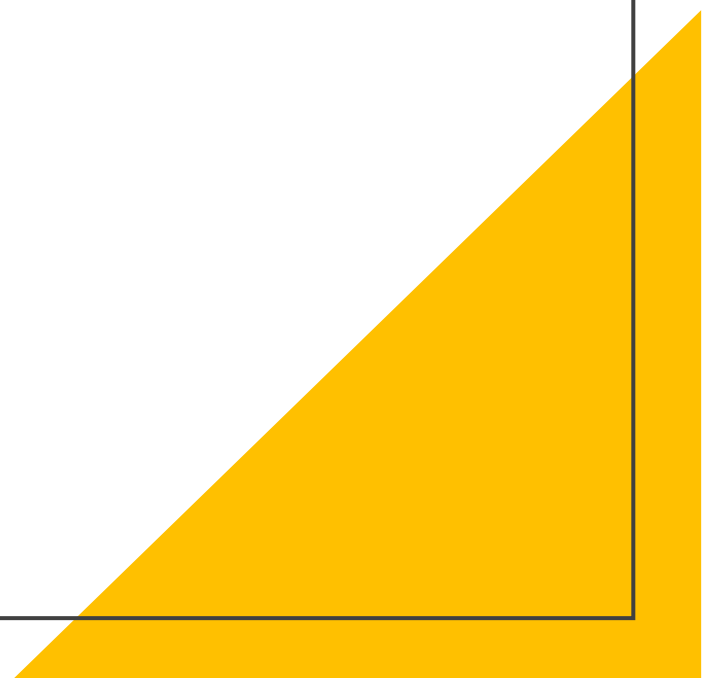




Deployment Descriptor

Deployment Descriptor

Die „web.xml“ zur Konfiguration der Applikation
Deklarative Konfiguration



J2EE 7 Grundgerüst der web.xml

```
<?xml version="1.0" encoding="UTF-8"?>  
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
    xmlns=http://xmlns.jcp.org/xml/ns/javaee  
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee  
    http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd",  
    version="3.1">
```

Innerhalb der web-app Tags kann die Konfiguration der Applikation erzeugt werden. Vieles lässt sich inzwischen per Annotation an den Klassen direkt konfigurieren, so dass die Erzeugung der „web.xml“ optional ist.

Häufig wird in diesem Zusammenhang von einer Programmatischen Konfiguration gesprochen.

Soll jedoch an einer Programmatischen Konfiguration eine Änderung stattfinden, so muss die gesamte Applikation anschließen erneut kompiliert und auf dem Server übertragen werden.

Änderungen an der „web.xml“ erfordern lediglich einen Neustart der Applikation.

```
</web-app>
```

J2EE 7 web.xml – Konfigurationen.

- `<servlet>(0-n)`
 - `<servlet-name>` (1) – Eindeutiger Name für das Servlet
 - `<servlet-class>` (1) – Vollständiger Klassenpfad des Servlets
 - `<jsp-file>` (1) – Angabe der JSP Datei, die als Servlet Konfiguriert werden soll.

<servlet-class> und <jsp-file> schließen sich einander aus. Eines von beidem muss jedoch angegeben werden

- `<load-on-startup>` (0-1) – Gibt an ob das Servlet zum Start der Applikation geladen werden soll.
 - Werte von 0-128. Je niedriger der Wert, desto eher wird das Servlet geladen
 - Standard ist -1, kein vorzeitiges laden.
- `<init-param>` (0-n)
 - `param-name` (1) – Name des Parameters
 - `param-value` (1) – Wert des Parameters

Das Servlet Tag bietet weitere Einstellungsmöglichkeiten, welche teilweise sehr Speziell bzw. an dieser Stelle nicht relevant sind.

J2EE 7 web.xml – Konfigurationen.

- `<servlet-mapping>(0-n)`
 - `<servlet-name> (1)` – Name des Servlet welches mit einer URL verbunden werden soll
 - `<url-pattern> (1-n)` – mit welcher URL das Servlet erreichbar sein soll
 - Mehrere URL können in einem Mapping zusammengefasst werden.
- `<welcome-file-list>(0-n)`
 - `<welcome-file> (1-n)` – In der hier aufgeführten Reihenfolge schaut der Server nach der Willkommensseite
- `<error-page> (0-n)`
 - `<error-code> (1)` – Http Error Code dem eine Fehlerseite zugeordnet werden soll
 - 1xx informational response
 - 2xx success
 - 3xx redirections
 - 4xx Client-Error's
 - 5xx Server-Error's
 - `<location> (1)` – Fehlerseite die zugeordnet werden soll
- `<security-constraint> & <security-role> & <login-config>`
 - Diese Tags werden im Abschnitt Security ausführlich besprochen.

Mehr unter

<https://de.wikipedia.org/wiki/HTTP-Statuscode>

https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

J2EE 7 web.xml – Konfigurationen.

- `<filter>`(0-n) – Konfigurieren von Filtern
 - `<filter-name>`(1) – Eindeutiger Name des Filters
 - `<filter-class>`(1) – Vollständiger Klassenname des Filters
- `<filter-mapping>` - Um einen Filter mit URLs zu verknüpfen
 - `<filter-name>`(1) – Name des Filters der verknüpft werden soll
 - `<servlet-name>`(0-n) – Servlets die mit dem Filter verknüpft werden sollen
 - `<url-pattern>`(0-n) – Request-URLS die mit dem Filter verknüpft werden sollen
 - `<dispatcher>`(0-n) – Auf welcher Art Dispatcher der Filter angewandt werden soll.
 - FORWARD – ERROR – INCLUDE – REQUEST – ASYNQ
 - Ohne Angabe regiert der Filter auf jeden Dispatcher Typ.
- `<listener>`(0-n) – Zum aktivieren von Listenern
 - `<listener-class>`(1) – Vollständiger Klassennamen des Listener der Aktiviert werden soll
 - Listener werden in der Reihenfolge wie sie deklariert wurden Aktiviert und geschaltet

J2EE 7 web.xml – Konfigurationen.

- <context-param> (0-n)
Anlegen von Applikationsweiten Parametern
 - <param-name> (1)
Eindeutiger Name des Parameter
 - <param-value> (1)
Wert des Parameter

„name“ und „value“ sind in der Applikation als String angelegt.

Parameter mit gleichen Namen überschreiben sich.

Wie zu erwarten gibt es nur einen SITH Lord

Auf Seiten der Jedi wären damit auch nur ein Meister zulässig.

```
<context-param>
    <param-name>JEDI Master</param-name>
    <param-value>Yoda</param-value>
</context-param>
<context-param>
    <param-name>JEDI Master</param-name>
    <param-value>Obi-Wan Kenobi</param-value>
</context-param>
<context-param>
    <param-name>SITH Lord</param-name>
    <param-value>Darth Maul</param-value>
</context-param>
<context-param>
    <param-name>SITH Lord</param-name>
    <param-value>Lord Vader</param-value>
</context-param>
```

J2EE 7 web.xml – Konfigurationen.

Der Deployment Descriptor bietet noch viele weitere Möglichkeiten die Applikation zu Konfigurieren.

- <post-construct> oder <pre-destroy> Möglichkeiten in den Life-Cycle der Applikation einzugreifen.
- <data-source> um eine Datenquelle zu initialisieren.
- <session-config> bietet eine Möglichkeit den Session-Timeout festzulegen.

Und weitere....



Servlets 3.1

Servlets 3.1

- Ein Servlet durchläuft 3 Phasen
 1. Initialisierungsphase
 - Methode `init()`
 - Wird beim ersten Laden/Aufrufen des Servlets ausgeführt
 2. Servicephase
 - Methode `service()`
 - Wird mit jedem Aufruf des Servlets ausgeführt
 3. Beendigungsphase
 - Methode `destroy()`
 - Wird ausgeführt, wenn die Applikation oder der Server sauber beendet wird.

Quelle https://de.wikipedia.org/wiki/Hypertext_Transfer_Protocol#HTTP-Anfragemethoden

- **GET**
ist die gebräuchlichste Methode. Mit ihr wird eine Ressource (zum Beispiel eine Datei) unter Angabe eines URI vom Server angefordert. Als Argumente in dem URI können auch Inhalte zum Server übertragen werden, allerdings soll laut Standard eine GET-Anfrage nur Daten abrufen und sonst keine Auswirkungen haben (wie Datenänderungen auf dem Server oder ausloggen).
- **POST**
schickt unbegrenzte, je nach physischer Ausstattung des eingesetzten Servers, Mengen an Daten zur weiteren Verarbeitung zum Server, diese werden als Inhalt der Nachricht übertragen und können beispielsweise aus Name-Wert-Paaren bestehen, die aus einem HTML-Formular stammen. Es können so neue Ressourcen auf dem Server entstehen oder bestehende modifiziert werden. POST-Daten werden im Allgemeinen nicht von Caches zwischengespeichert. Zusätzlich können bei dieser Art der Übermittlung auch Daten wie in der GET-Methode an den URI gehängt werden.
- **HEAD**
weist den Server an, die gleichen HTTP-Header wie bei GET, nicht jedoch den Nachrichtenrumpf mit dem eigentlichen Dokumentinhalt zu senden. So kann zum Beispiel schnell die Gültigkeit einer Datei im Browser-Cache geprüft werden.
- **PUT**
dient dazu, eine Ressource (zum Beispiel eine Datei) unter Angabe des Ziel-URIs auf einen Webserver hochzuladen. Besteht unter der angegebenen Ziel-URI bereits eine Ressource, wird diese ersetzt, ansonsten neu erstellt.
- **PATCH**
Ändert ein bestehendes Dokument ohne dieses wie bei PUT vollständig zu ersetzen. Wurde durch RFC 5789 spezifiziert.
- **DELETE**
löscht die angegebene Ressource auf dem Server.
- **TRACE**
liefert die Anfrage so zurück, wie der Server sie empfangen hat. So kann überprüft werden, ob und wie die Anfrage auf dem Weg zum Server verändert worden ist – sinnvoll für das Debugging von Verbindungen.
- **OPTIONS**
liefert eine Liste der vom Server unterstützten Methoden und Merkmale.
- **CONNECT**
wird von Proxyservern implementiert, die in der Lage sind, SSL-Tunnel zur Verfügung zu stellen.

Servlets 3.1

- Die Klasse `HttpServlet` stellt Methoden zu den HTTP Request Methoden zur Verfügung.
 - Die Standard „`service()`“ Methode von `HttpServlet` analysiert den Request nach HTTP Request Methoden
 - Die „`service()`“ Methode erhält vom Server 2 Objekte.
 - `HttpServletRequest` – Die anfrage des Clients
 - `HttpServletResponse` – Die zu erzeugende Antwort des Servlets

Servlets 3.1

- HttpServletRequest enthält alles aus der Anfrage des Clients.
 - getHeaderNames – Um an die Header Namen für getHeader. „Enumeration<String>“
 - getHeader – um einen Speziellen Header abzufragen
 - Header beinhalten Informationen zu den Clients
 - getAttributeNames – zum Abfragen der Attributnamen. „Enumeration<String>“
 - getAttribute – um ein Spezielles Attribut abzufragen
 - Attribute werden später besprochen
 - getParameterNames – zum Abfragen der Parameternamen. „Enumeration<String>“
 - getParameter – um ein Speziellen Parameter abzufragen
 - getQueryString – Wenn Daten in der URL geliefert oder von einem Formular mit GET gesendet werden.

Aufgabe 01

- Erzeugen Sie eine Webseite mit einem Formular, wo der Benutzer seinen Vor- und Zunamen eingibt.
- Auch soll die Möglichkeit bestehen eine kurze Textpassage einzugeben.
- Die Daten nehmen Sie in einem Servlet entgegen.
- Die Textpassage Speichern Sie in einen geeigneten Datentyp im Servlet.
- Das Servlet soll den Vor und Zunamen des Benutzer ausgeben gefolgt von einer Unsortierten Liste alle eingegeben Textpassagen.