



FH Bielefeld
University of
Applied Sciences

Campus Minden

Webbasierte Anwendungen

SS 2018

Client-Server-Kommunikation

Dozent: B. Sc. Florian Fehring
mailto: florian.fehring@fh-bielefeld.de

Studiengang Informatik

Client-Server-Kommunikation

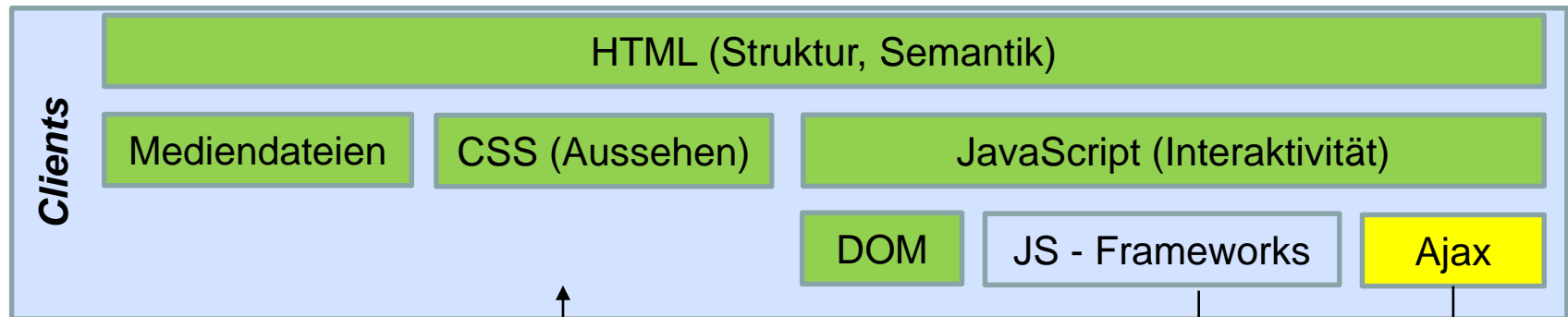
1. Kontext und Motivation

- 2. Dynamisches Laden
- 3. WebServer
- 4. Datenübertragung
- 5. Authentifizierung
- 6. Session-Management
- 7. Darüber hinaus
- 8. Projekt

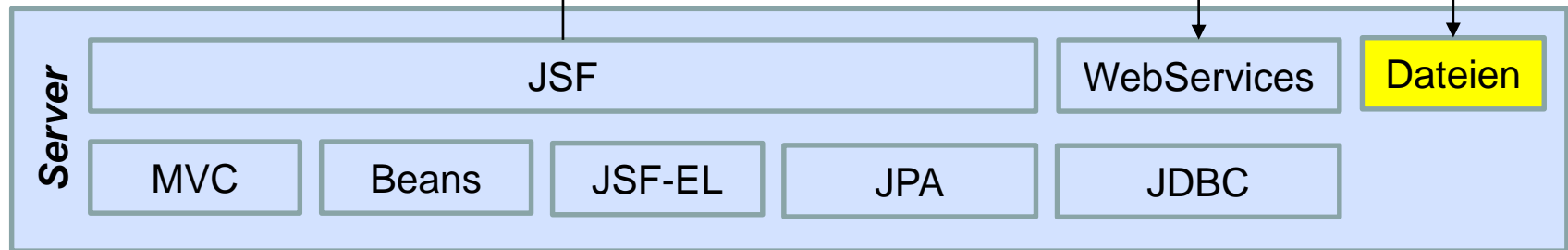
Problemfelder

Web-Anwendung

Mensch-Maschine-Kommunikation



Maschine-Maschine-Kommunikation



Anforderungen

Welche Anforderungen werden als nächstes bearbeitet?

TODO

- Externe Inhalte einbinden
- Artikel vom Server einbinden
- Kommentare vom Server
- Artikel zum Server übertragen
- Kommentare zum Server
- Medien zum Server
- Kommentare speichern
- Kommunikation untereinander

DONE

- Technologische Grundlagen erarbeiten
- Was ist eine Web-Anwendung?
- News darstellen
- Projekte vorstellen
- Aufgaben darstellen
- Formular für Kommentare
- Schickes Design für die Seite
- Mediendateien einbinden
- Animationen
- Mehrsprachen-Fähigkeit
- (lokales) Speichern von Artikeln
- Client-Position anzeigen
- Offline-Verwendung ermöglichen
- Inhaltsverzeichnisse
- Formlareingaben in Seite einfügen
- Navigation über Tastaturkürzel

Client-Server-Kommunikation

1. Kontext und Motivation
- 2. Dynamisches Laden**
3. WebServer
4. Datenübertragung
5. Authentifizierung
6. Session-Management
7. Darüber hinaus
8. Projekt

Dynamisches Laden I

Definition: Dynamisches Laden von Ressourcen bezeichnet das Laden von Ressourcen, wenn sie benötigt werden.

Grundsätzlich werden alle Ressourcen einer Webapplikation über Tags eingebunden und dadurch mit der Seite geladen.

Vorteile dynamischen Ladens:

- Einsparung von Datenvolumen, da nicht benötigte Ressourcen auch nicht geladen werden müssen
- Hinzuladen von Ressourcen, die aufgrund von Benutzerinteraktivität erzeugt werden
- Mit DOM-Manipulation ist es möglich Daten aus neu abgerufenen Ressourcen in die Seite einzupflegen, ohne die Seite komplett neu zu laden

Nachteile dynamischen Ladens:

- Verwendung der „zurück“-Schaltfläche eingeschränkt bzw. verändert nutzbar

Dynamisches Laden II – Synchron und Asynchron

Dynamisches Laden von Ressourcen erfolgt mittels JavaScript und kann synchron oder asynchron erfolgen.

Synchrones Laden:

- Blockiert den Programmfluss
- Benutzer müssen auf den Abschluss des Ladevorgangs warten das Skript „hängt“
- Der Programmfluss ist einfach nachvollziehbar

Asynchrones Ladens:

- Datenübertragung zwischen Client und Server asynchron im Hintergrund
- der Anwender bemerkt nichts von der Datenübertragung
- die Anwendung kann weiterhin bedient werden (z.B Reaktion auf click)
- Der Programmablauf ist nicht einfach Nachvollziehbar
- Nach dem Laden wird ein Ereignis „onLoadReady“ ausgelöst

Synchrones Laden kann beispielsweise über das dynamische Einfügen eines `<script>`-Tags im Header erfolgen.

Ajax - Datenübertragung: synchron oder asynchron?

- Antwort erfolgt erst nach vollständiger Übertragung aller angeforderten Daten

- Anfrage bleibt aktiv, bis Antwort vom Server erzeugt wurde
-> Blockade des weiteren Programmablaufs

- Ergebnis nach der Übertragung sofort und komplett verfügbar

- keine Statusinformationen während Datenübertragung verfügbar
-> bei längeren Datenübertragungen wird vom Anwender aus Unwissenheit oft abgebrochen

- Antwort des Servers erfolgt sofort
-> Requests sind nicht an den Rhythmus von „Formular absenden“ oder „Seite laden“ usw. gebunden

- Nächster Request kann gestellt werden, noch bevor vorheriger beantwortet wurde

- beliebige Elemente einer Seite können isoliert aktualisiert werden.

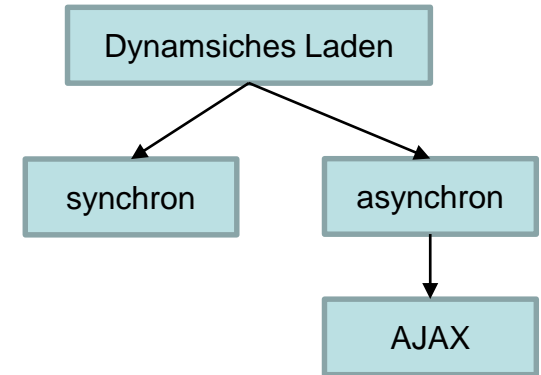
- aktueller Status wird während der Anfrage ständig geliefert
-> optimale Kontrolle über den Ablauf der Anfrage an den Server

Dynamisches Laden III - Ajax

Definition: Asynchron Javascript And XML (AJAX) ist keine neue Technologie, sondern eine sinnvolle Kombination bestehender Technologien

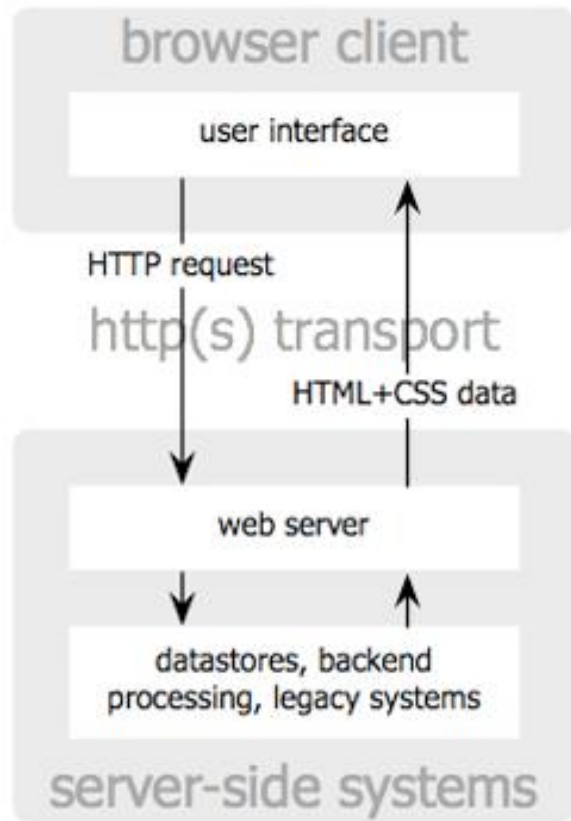
Komponenten von Ajax:

- XMLHttpRequest-Objekt
 - Besitzt asynchrone JavaScript-Methoden
- JavaScript als Schnittstelle aller Komponenten
- DOM
- XML (in letzter Zeit häufig abgelöst von JSON)



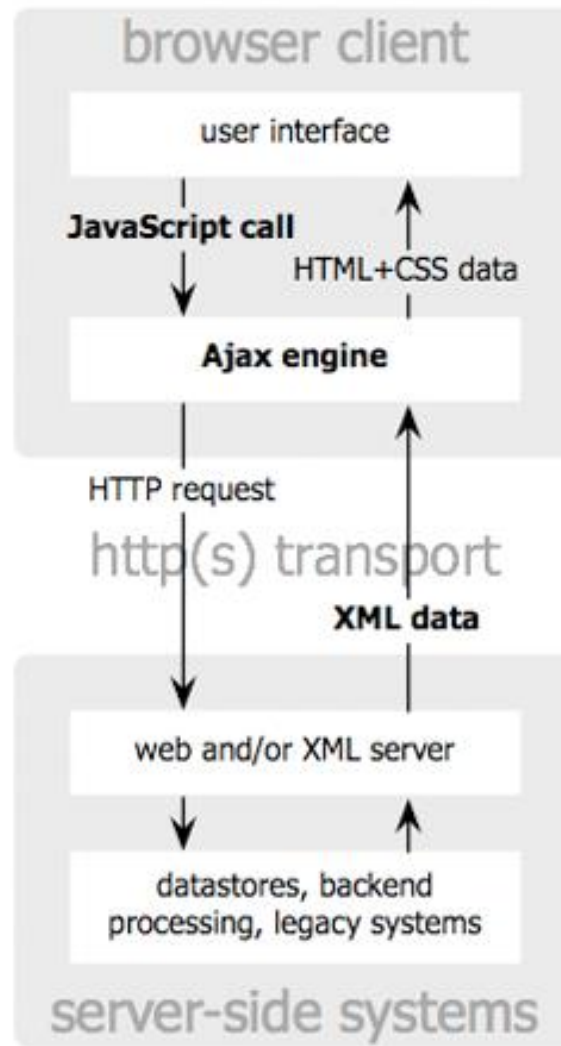
Hinweise:

- „Ajax“ funktionierte längere Zeit als Marketing-Begriff (ca.2005-2010)
- Begriff wurde im Februar 2005 von Jesse James Garrett geprägt
- Begriff sollte beitragen, eine neue Generation der Webentwicklung zu beschreiben und den Begriff „WEB2.0“ zu untermauern



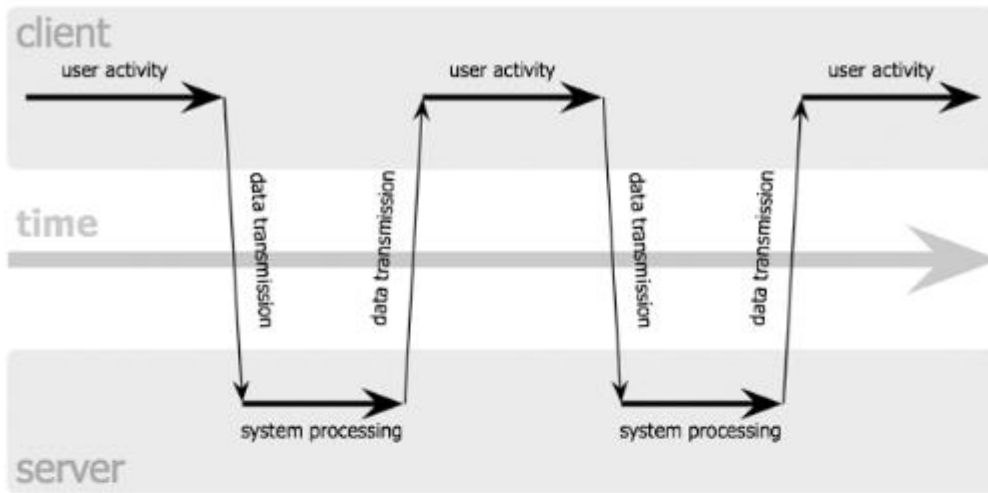
classic
web application model

Jesse James Garrett / adaptivepath.com

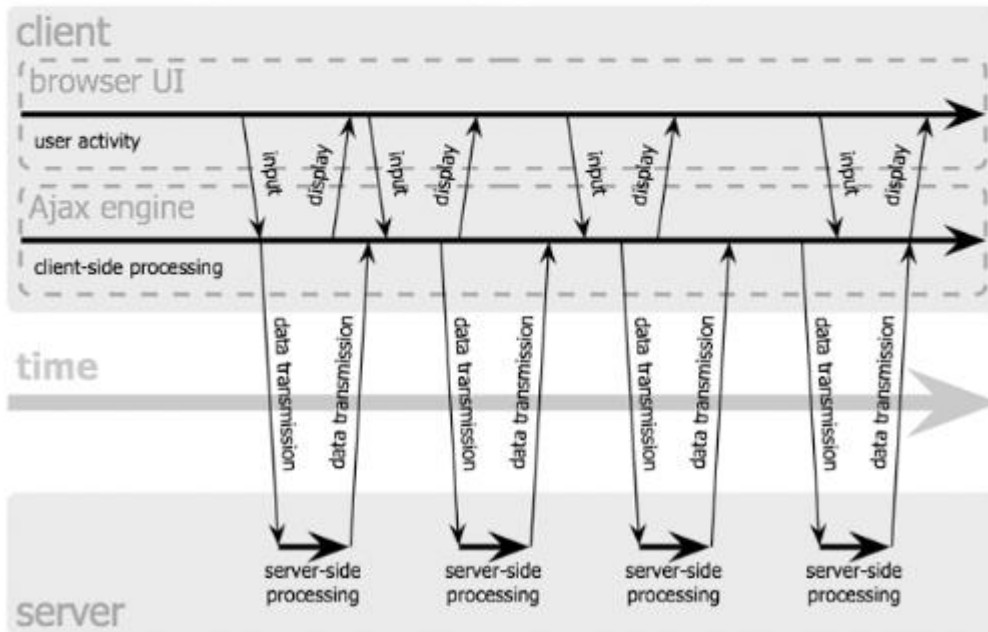


Ajax
web application model

classic web application model (synchronous)



Ajax web application model (asynchronous)



<http://www.adaptivepath.com/ideas/essays/archives/000385.php>

Dynamisches Laden III - XMLHttpRequest

Definition: Das XMLHttpRequest-Objekt ermöglicht Client/Server-Kommunikation mittels JavaScript

Eigenschaften:

- Übertragung von Textinhalten (und alles was als Text darstellbar ist)
- Übertragung von Binärdaten (unter Nutzung der base64 Codierung)
- Übertragung verläuft asynchron
- nach Erhalt der Daten wird eine callback-Methode aufgerufen
- unterstützt auch andere Protokolle (z.B. ftp, file,...)

Geschichte:

- wurde von Microsoft mit dem MSIE 5.0, als eine ActiveX-Komponente, welche Daten von einem Server anfordern kann, eingeführt
- Mozilla 1.0 und Netscape 7 (beide auch Gecko-Browser genannt) zogen schnell nach mit dem **XMLHttpRequest-Objekt**

Hinweis: Einige Frameworks (wie z.B. jQuery) bieten einfacher zu benutzende Schnittstellen

Dynamisches Laden III - XMLHttpRequest

```
new XMLHttpRequest();
```

Eigenschaften:

<code>responseText</code>	Serverantwort als String im Body
<code>readyState</code>	Aktueller Status des Requests
<code>status</code>	Numerischer Wert des Serverstatus
<code>statusText</code>	Beschreibung des Serverstatus
<code>onreadystatechange</code>	Event-Handler, der eine zugeordnete Funktion bei jeder Änderung im Status des Requests ausführt.
<code>timeout</code>	Zeit nach der die Anfrage abgebrochen wird (msek)

Methoden:

<code>open()</code>	starten eines Request an Zieladresse
<code>send()</code>	sendet Request an den Server
<code>setRequestHeader()</code>	Setzen eines Headers für Request
<code>getAllResponseHeaders()</code>	gibt Liste aller vorhandenen Header
<code>getResponseHeader()</code>	gibt Wert für angegebenen Headername
<code>abort()</code>	Abbruch der Anfrage

Dynamisches Laden III – open()/send()

```
open(method, url[,syncFlag, username, password])
```

Öffnet eine Verbindung zum Server. Die Anfrage wird dadurch noch nicht abgesendet.

Parameter:

method	zu verwendende HTTP-Methode (GET, POST,...)
url	URL für gefordertes Dokument (z.B. http://mysite.de/doc.json)
syncFlag	asynchron (true; default) oder synchron (false) abfragen
username	Benutzername falls erforderlich
password	Passwort falls erforderlich

```
send(body|null)
```

Sendet die Anfrage an den Server ab. Sendet dabei die angegebenen Daten mit.

Parameter:

body Daten die mitgesendet werden sollen

Dynamisches Laden III – open()/send()

```
// Funktion zum Reagieren auf einen Status-Wechsel
var readyStateCallbackFunction = function() {
    if (this.readyState == 4 && this.status == 200) {
        console.log("Folgende Antwort erhalten: ");
        console.log(this.responseText);
    } else {
        console.log("readyState: " + this.readyState + " Status: " +
this.status);
    }
};

window.onload = function() {
    let requestor = new XMLHttpRequest();
    requestor.open("GET", "xmlhttprequest.json");
    requestor.onreadystatechange = readyStateCallbackFunction;
    requestor.send();
}
```

```
readyState: 2 Status: 200
readyState: 3 Status: 200
{
  title : "Simple json",
  letters : ["a","b","c"]
}
```

Dynamisches Laden III – readyState

Werte und deren Bedeutung:

Wert	Bedeutung	Beschreibung
0	uninitialized	Request wurde noch nicht durch open() ausgelöst
1	loading	Request wird gestartet, wurde aber bisher noch nicht abgeschickt
2	loaded	Request wurde durch send() ausgeführt, die Serverantwort steht noch aus
3	interactive	Übertragung der Serverantwort läuft, Teile davon sind schon im Buffer und mittels responseText oder responseXML verfügbar
4	complete	Request wurde vollständig ausgeführt und beendet

Dynamisches Laden II – Weitere Methoden

`setRequestHeader (key, value)`

Setzt Header-Informationen (mit jedem Aufruf eine), die beim Absenden mit an den Server geschickt werden. Muss vor der send-Methode stehen.

```
req.setRequestHeader("Accept", "image.gif");           //offizieller Http_header  
req.setRequestHeader("PersonalExample", "value");       //eigener Header
```

`getAllResponseHeaders () ; getResponseHeader ()`

Diese Methoden liefern Headerinformationen des Servers.

`abort ()`

Beendet eine Anfrage vorzeitig.

Dynamisches Laden IV – FetchAPI

Definition: Die FetchAPI ist eine API für den Zugriff auf Ressourcen. Sie bietet flexiblere und umfangreichere Möglichkeiten als das XMLHttpRequest

```
fetch(URL[, init])
```

Eigenschaften:

- Zugriff auf lokale wie Netzwerkressourcen
- Umfangreiches Toolset
- Verwendet das Promise-Konzept
- Verwendbar in Web- und ServiceWorkern
- Konfiguration der Anfrage mit init-Objekt

GlobalFetch-Objekt:

- Objekt das die fetch() Methode enthält
- Liefert eine Promise und bei Erfolg ein Response-Objekt

```
fetch(URL).then(  
  function(response) {  
    ...  
  }  
) .catch(...)
```

Weitere Informationen: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

Dynamisches Laden IV – Response

`new Response()`

Eigenschaften:

<code>body</code>	Serverantwort als ReadableStream im Body
<code>status</code>	Numerischer Wert des Serverstatus
<code>statusText</code>	Beschreibung des Serverstatus
<code>ok</code>	ist true, wenn die Anfrage erfolgreich war (HTTP200-299)
<code>headers</code>	HTTP-Header der empfangenen Antwort

Methoden:

<code>text()</code>	liefert ein Promise, das zu einem String des Inhalts wird
<code>json()</code>	liefert ein Promise, das zu einem JSON-Objekt des Inhalts wird
<code>blob()</code>	liefert ein Promise, das zu einem Blob-Objekt wird

Dynamisches Laden IV – Response

```
fetch('fetchapi.json').then(  
  function(response) {  
    console.log("Get response as json-Promise");  
    return response.json();  
  }  
) .then(  
  function(jsonData) {  
    console.log("recived data: " + jsonData);  
  }  
) .catch(function(err) {  
  console.log("Opps, Something went wrong!", err);  
})
```

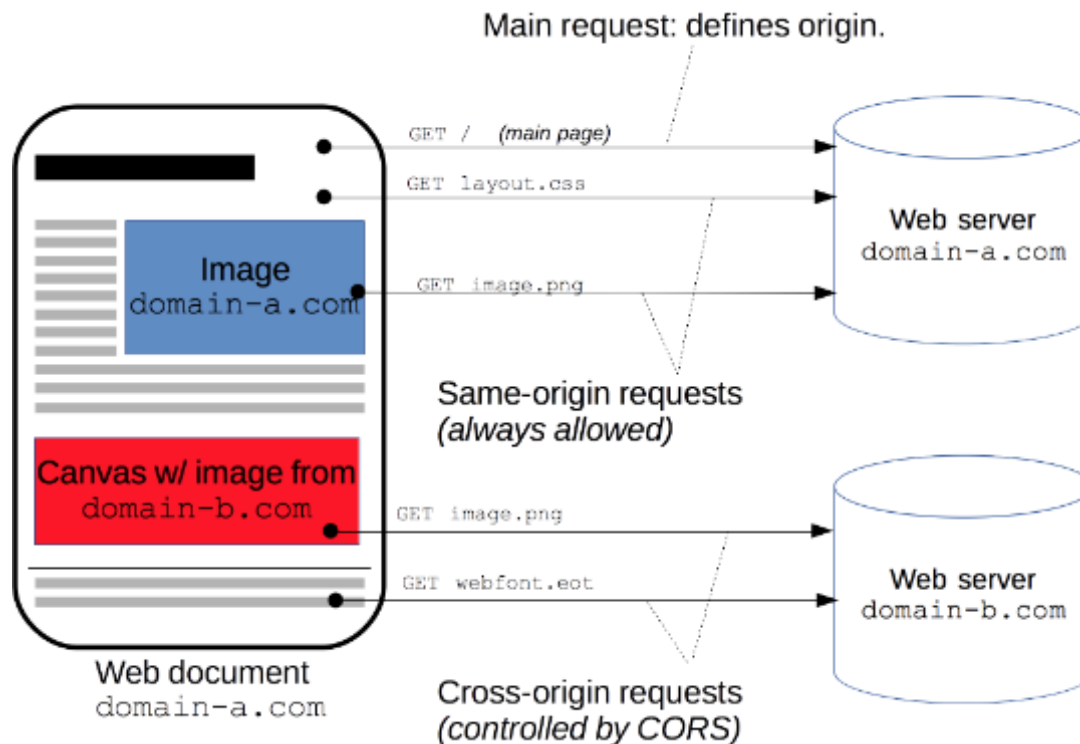
Dynamisches Laden IV – Response

```
function postData(url, data) {
  // Default options are marked with *
  return fetch(url, {
    body: JSON.stringify(data), // must match 'Content-Type' header
    cache: 'no-cache', // *default, no-cache, reload, force-cache, only-if-
cached
    credentials: 'same-origin', // include, *omit
    headers: {
      'user-agent': 'Mozilla/4.0 MDN Example',
      'content-type': 'application/json'
    },
    method: 'POST', // *GET, PUT, DELETE, etc.
    mode: 'cors', // no-cors, *same-origin
    redirect: 'follow', // *manual, error
    referrer: 'no-referrer', // *client
  })
  .then(response => response.json()) // parses response to JSON
}

window.onload = function() {
  postData('fetchapi-post-target.json', {answer: 42}).then(
    function(data) {
      console.log(data);
    }
  ) // JSON from `response.json()` call
  .catch(error => console.error(error))
}
```

Dyn. Laden V - Cross-Domain-Access

Definition: Cross-Domain-Anfragen sind Anfragen die vom Client, an einen Server geschickt werden, der nicht die Quelle der angezeigten HTML-Seite ist.



Quelle: https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS

Dyn. Laden V - Cross-Domain-Access

Definition: Die Same-origin Policy besagt, dass Browser Anfragen von einer geladenen Seite nur an die Quelle der geladenen Seite erlauben.

Grund ist, das ansonsten leicht manipulierte Inhalte in eine Seite eingeschleust werden könnten.

Cross-Domain-Access mit same-origin Policy:

- Anfragen via XMLHttpRequest / Fetch
- Web Fonts
- WebGL Texturen
- Videos / Bilder, bei Verwendung von drawImage()

Nicht betroffen sind:

- Laden von Bildern, Videos, CSS über HTML-Tags oder deren Attribute (src-Attribute)
- Laden von Scripten über das <script>-Tag

Dyn. Laden V - Cross-Domain-Access

Lösungen: JsonP oder CORS

JsonP:

- Nutzt das <script>-Tag für Anfragen.
- Funktioniert nur mit GET-Requests
- Muss die Antwort in einem „Umschlag“ erhalten um sie der aufrufenden Stelle zuordnen zu können

CORS:

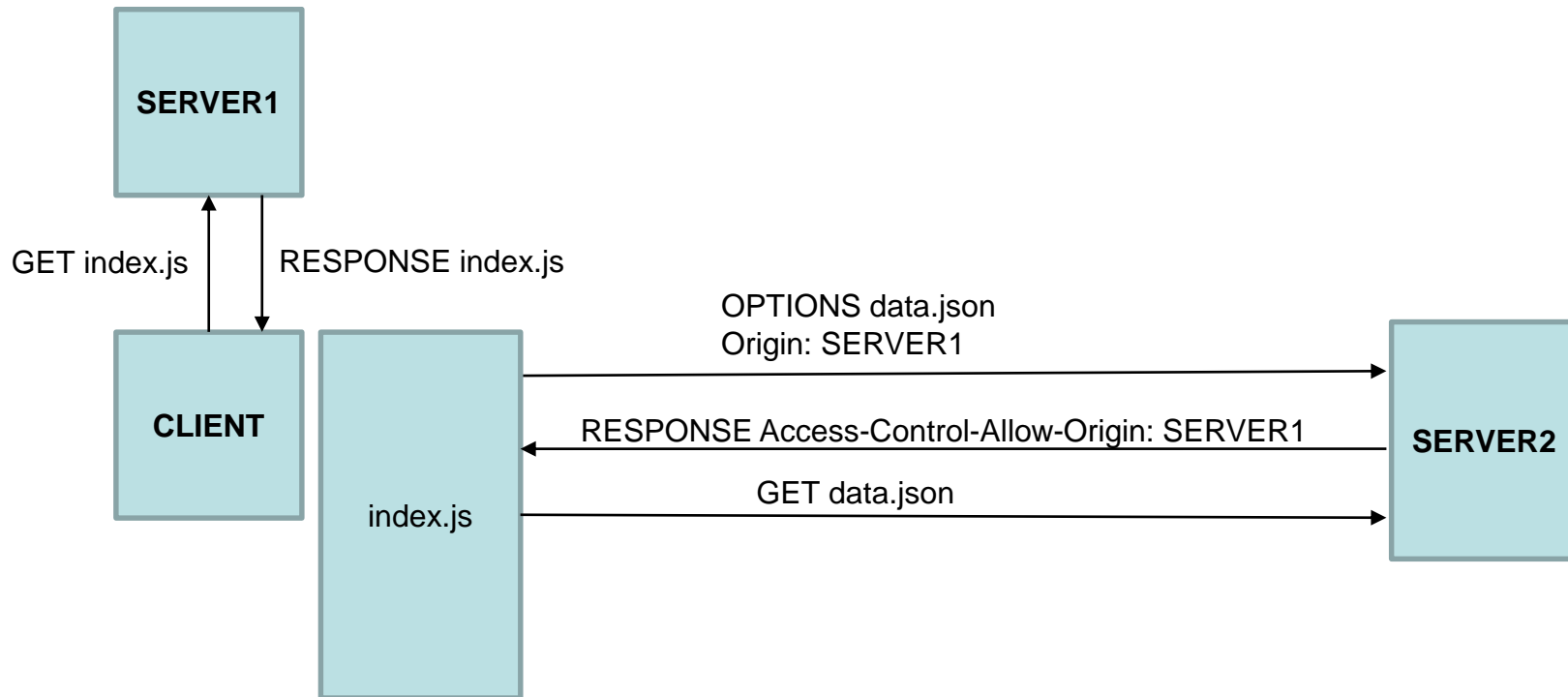
- Erlaubnis von CDA durch den, die Seite liefernden Server
- Nutzung von speziellen HTTP-Headern
- Funktioniert mit allen Request Arten (GET, POST, PUT,...)

Dyn. Laden V - Cross-Domain-Access

```
function postData(url, data) {
  // Default options are marked with *
  return fetch(url, {
    body: JSON.stringify(data), // must match 'Content-Type' header
    cache: 'no-cache', // *default, no-cache, reload, force-cache, only-if-cached
    credentials: 'same-origin', // include, *omit
    headers: {
      'user-agent': 'Mozilla/4.0 MDN Example',
      'content-type': 'application/json'
    },
    method: 'POST', // *GET, PUT, DELETE, etc.
    mode: 'cors', // no-cors, *same-origin
    redirect: 'follow', // *manual, error
    referrer: 'no-referrer', // *client
  })
  .then(response => response.json()) // parses response to JSON
}

window.onload = function() {
  postData('fetchapi-post-target.json', {answer: 42}).then(
    function(data) {
      console.log(data);
    }
  ) // JSON from `response.json()` call
  .catch(error => console.error(error))
}
```

Dyn. Laden V - Cross-Domain-Access



Client-Server-Kommunikation

1. Kontext und Motivation
2. Dynamisches Laden
- 3. WebServer**
4. Datenübertragung
5. Authentifizierung
6. Session-Management
7. Darüber hinaus
8. Projekt

WebServer

Definition: Ein WebServer stellt Dateien und Dienste über ein Netzwerk zur Verfügung.

File-Server

- Verwaltet Mengen von Dateien
- Ermöglicht den Zugriff (geschützt / ungeschützt)
- Anfragen können über verschiedene Protokolle erfolgen (z.B. FTP)

Web-Server

- Anfragen werden per HTTP-Protokoll gestellt und beantwortet
- Erlaubt die Ausführung von Skripten und somit dynamische Inhalte

Web-Application-Server

- Erlaubt die Ausführung von Programmen höherer Sprachen
- Kapselt Datenquellen
- Standardisierte Schnittstellen zu anderen Diensten (z.B. Persistenz)
- Unterstützt einen Software-Lifecycle

WebServer II – Verbreitete Server

Web-Server

- Apache HTTP-Server
 - Verwendet auf 47% der Server
 - Entwickelt seit 1995
 - Häufige Verwendung mit PHP
- NGINX
 - Verwendet auf 37% der Server
 - Entwickelt seit 2010
 - Häufig mit PHP Unterstützung
 - Auch häufige Verwendung als Gateway-Server



Statistische Werte von: https://w3techs.com/technologies/overview/web_server/all

WebServer II – Verbreitete Server

Web-Application-Server

- Tomcat
 - Verwendet auf 0,5% der Server
 - Entwickelt seit 1999
 - Java EE Server ohne EJB-Server
 - Erweiterung im Apache TomEE
 - Weitere Java-EE Unterstützung (WebProfile)
 - Unterstützt JavaEE6
- Glassfish / Payara
 - Verwendet auf 0,1% der Server
 - Entwickelt seit 2005
 - Voller Java EE Server mit EJB-Container
 - Unterstützt JavaEE 8
 - Eingebaute Unterstützung für DB-Verbindungen / Transaktionen
 - Unterstützung von REST WebServices



Statistische Werte von: https://w3techs.com/technologies/overview/web_server/all

WebServer III - Interfaces

Definition: Ein WebServer Interface ist die Verbindungsstelle zwischen dem WebServer und einer Programmausführung

Eigenschaften:

- Die Server nehmen Anfragen von Clients entgegen und reichen die Parameter an das Programm weiter
- Die Server starten die Programme und geben ihnen Informationen über die Umgebung mit.
- Die Server nehmen die Antworten der Programme entgegen und verpacken sie in HTTP-Antworten.

Arten:

- Common Gateway Interface (CGI)
- (Java) Servlets

WebServer IV - CGI

Definition: Das Common Gateway Interface (CGI) ist ein Standard für den Datenaustausch zwischen Server und nativer Anwendung

Eigenschaften:

- Ausführung nativer Programme auf dem Server
 - Programme sind plattformabhängig
 - Direkte Ausführung der Programme
- Programme werden als eigener Betriebssystemprozess ausgeführt
 - Benötigte Ressourcen müssen separat geöffnet werden
- Kein Datenaustausch unter Anwendungen
 - Datenaustausch nur über Dateien oder Datenbank
- Keine direkte Manipulation von HTTP-Headern
 - Kein direktes Auslesen oder Setzen von Cookies

Durch die Verwendung von Skriptsprachen besteht eine gewisse Plattformunabhängigkeit. Dafür müssen die Skripte zur Laufzeit geparkt werden.

WebServer V – Verbreitete Sprachen

Skript-Sprachen

- PHP
 - Verwendet auf 83,2% der Server
 - Entwickelt seit 1995
 - Objektorientiert
 - Interpretiert
- Ruby, Perl, Python
 - Kaum Verwendung für Webseiten

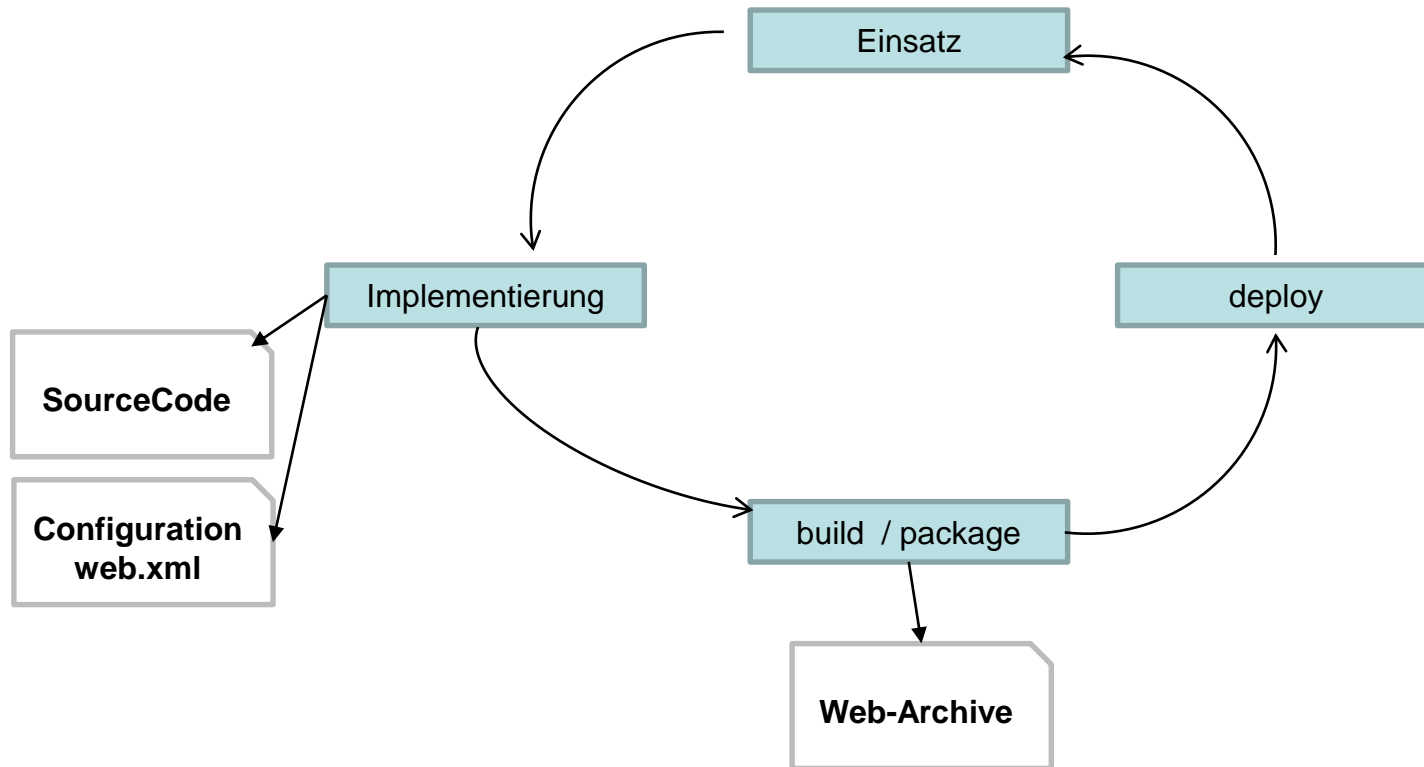
Höhere Sprachen

- ASP.net
 - Verwendet auf 13,9% der Server
 - Entwickelt seit 2002
 - Dahinter wird C# oder VB.NET verwendet
- Java
 - Verwendet auf 2,4% der Server
 - Verwendung mit umfangreichen Erweiterungen aus JavaEE

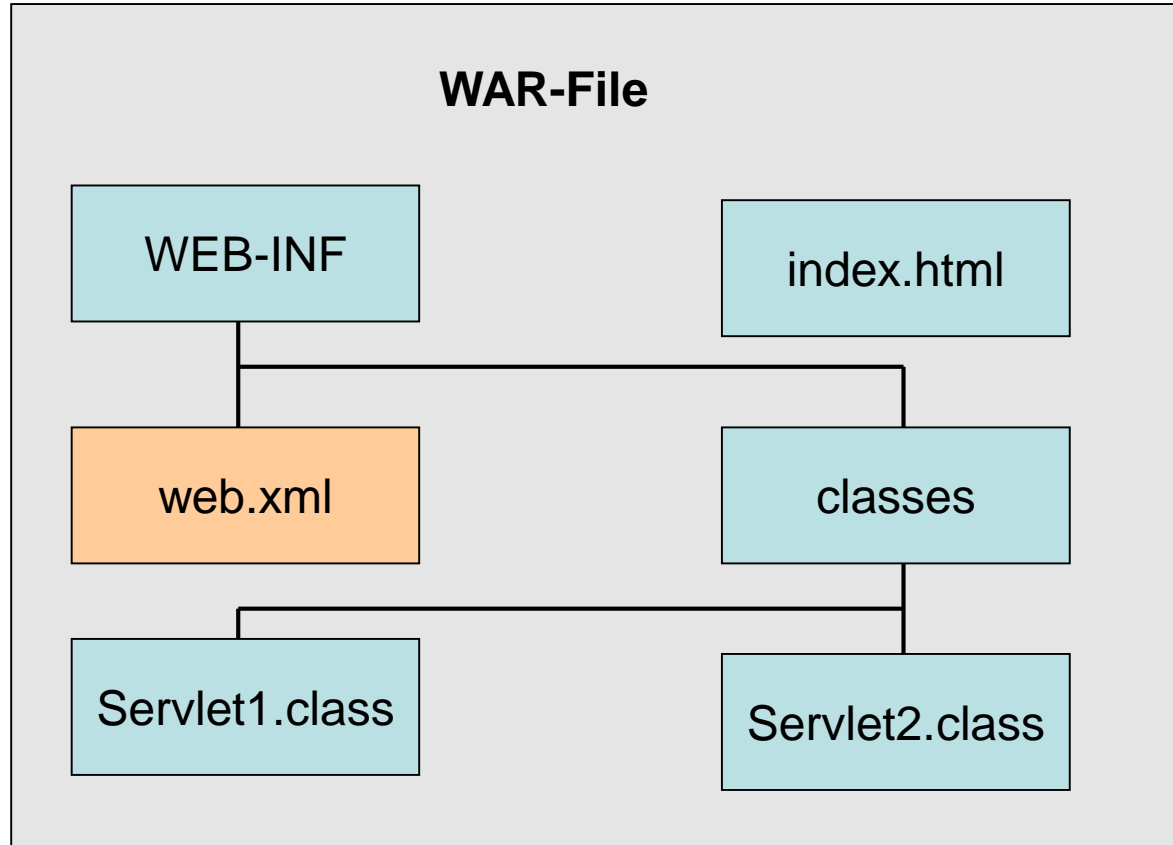
Werte von: https://w3techs.com/technologies/overview/programming_language/all (3/2018)

WebServer VI – Application-Server

Lebenslauf einer Anwendung auf einem Application-Server:



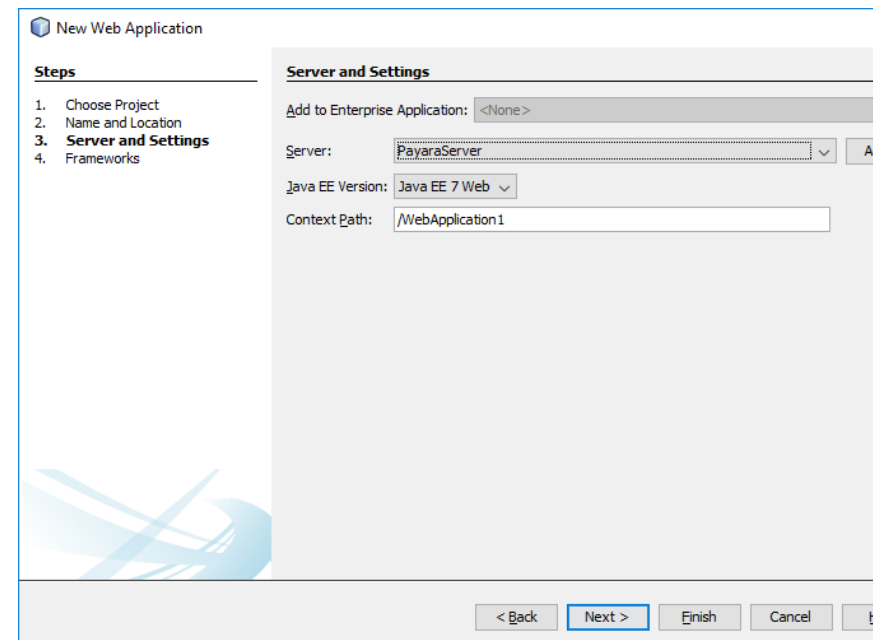
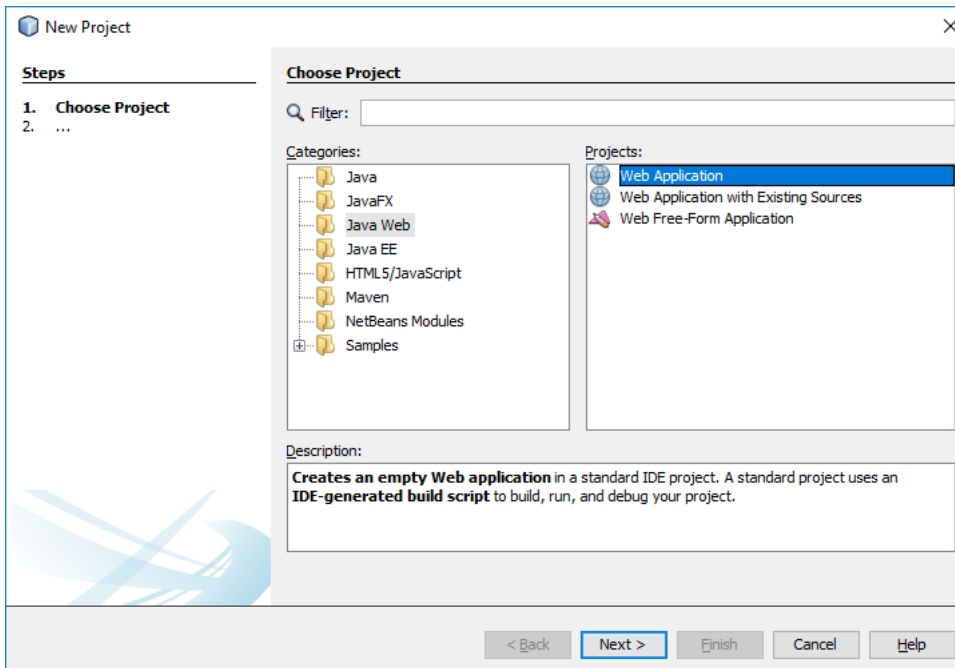
WebServer VII – Web-Archive



WebServer VIII – Java-WebApplication

Erstellen einer Java-WebApplication (am Beispiel Netbeans):

1. File / New Project...
2. Categories: Java Web
3. **Projects: Web Application**
4. Projektnamen vergeben
5. **Server: Integrierten Server auswählen**
6. Frameworks: Auswahl von mitgelieferten Frameworks



WebServer IX – Java-WebApplication

Die web.xml enthält eine Beschreibung und die Konfiguration der Java WebApplication.

```
<?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://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
          version="3.1">

    <!-- Hier stehen weitere Konfigurationseinträge -->

</web-app>
```

Client-Server-Kommunikation

1. Kontext und Motivation
2. Dynamisches Laden
3. WebServer
4. Datenübertragung
5. Authentifizierung
6. Session-Management
7. Darüber hinaus
8. Projekt

Client-Server-Kommunikation

1. Kontext und Motivation
2. Dynamisches Laden
3. WebServer
- 4. Datenübertragung**
5. Authentifizierung
6. Session-Management
7. Darüber hinaus
8. Projekt

Datenübertragung I – Ablauf

Definition: Browser laden nach der Adressauflösung das Dokument und nach und nach die verknüpften Ressourcen für ein Dokument.

Grundlegender Ablauf:

1. der erste Aufruf lädt das angefragte Dokument
2. der Browser schaut nach, ob im Dokument weitere Ressourcen angegeben sind und lädt diese nach
3. der Browser parst und rendert alle Dokumente

Probleme:

- Es müssen erst alle Daten übertragen werden
- Der Benutzer muss lange warten, bis er etwas sieht

Dieses Vorgehen war in den ersten Browser-Generationen implementiert


Datenübertragung I – Ablauf

Verbesserung: Browser verarbeiten ein Dokument, sobald sie eine Teil davon empfangen haben.

Verbesserter Ablauf:

1. der erste Aufruf lädt das angefragte Dokument
2. **der Browser parst einen Teil des Dokuments, sobald er verfügbar ist**
3. der Browser schaut nach, ob im Dokument weitere Ressourcen angegeben sind und lädt diese nach
4. **Ressourcen können parallel geladen werden**
 - Dadurch: Bessere Ausnutzung der Bandbreite, „schnelleres“ laden
5. **die hinzugeladenen Dokumente werden sobald verfügbar eingefügt**

Problem:

- Verschwendung von [ lumen, wenn immer wieder alle Dokumente neu geladen werden

Datenübertragung II – Caching

Verbesserung: Durch einen Client erfragte Ressourcen können in Caches zwischengespeichert werden.

Verbesserter Ablauf:

1. der erste Aufruf lädt das angefragte Dokument (**wenn vorhanden aus Cache**)
2. der Browser parst einen Teil des Dokuments, sobald er verfügbar ist
3. der Browser schaut nach, ob im Dokument weitere Ressourcen angegeben sind und lädt diese nach (**wenn vorhanden aus Cache**)
4. Ressourcen können parallel geladen werden
 - Dadurch: Bessere Ausnutzung der Bandbreite, „schnelleres“ laden
5. die hinzugeladenen Dokumente werden sobald verfügbar eingefügt

Hinweis:

- Tatsächlich haben nicht nur Clients einen Cache, sondern auch Server. Zum Beispiel solche, die auf dem Weg vom ApplicationServer zum Client durchlaufen werden.

Datenübertragung II – Caching

Vorteile:

- schnellere Bereitstellung von Ressourcen
- Reduzierung der notwendigen Bandbreite zur Datenübertragung
- Verbesserung der Nutzbarkeit (Usability) der Webanwendung

Problem:

- Was soll im Cache gespeichert werden?
- Woran wird erkannt, dass es sich um dieselbe Ressource handelt?
- Wie lange soll eine Ressource im Cache bleiben?

Lösungen:

- Statische Ressourcen identifizieren
 - Häufig Bilder, Videos,...
- Ressourcen eindeutig identifizierbar machen
 - die selben Ressourcen immer über die selbe URL referenzieren
- Informationen über die Cache-Nutzung implementieren
 - Verfallsdatum für eine Ressource angeben

Datenübertragung II – Caching

Expires: [date]

Cache-Control: [directive]

Expires:

date Datumsangabe, bis wann das Dokument gültig sein soll

Cache-Control:

no-cache Cache muss vor Lieferung einer gecachten Ressource einen Validierungsrequest an den Server stellen.

no-store Cache darf die Ressource nicht zwischenspeichern

max-age Legt Verfallszeitpunkt fest, mit Angabe der Zeit in Sekunden

Validierungsrequest:

Der Request-Header wird mit der Angabe **if-modified-since** ergänzt. Der Server liefert die Ressource nur dann, wenn sie aufgrund Ihres Zeitstempels nicht mehr gültig ist.

```
Cache-Control: max-age=3600
```

```
Expires: Thu, 01 Dec 1994 16:00:00 GMT
```

Datenübertragung II – Caching

Einstellen der HTTP-Header:

- HTTP-Header können in einer Java-WebApplication über Filter modifiziert werden
- Filter werden in der web.xml registriert
- Filter implementieren ein Filter-Interface

```
<filter>
    <description>Set cache expiry for static content</description>
    <filter-name>ExpiresFilter</filter-name>
    <filter-class>de.fhbielefeld.scl.server.filter.ExpiresFilter</filter-
class>
    <init-param>
        <description>Add an Expires Header</description>
        <param-name>days</param-name>
        <param-value>30</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>ExpiresFilter</filter-name>
    <url-pattern>*.html</url-pattern>
    <dispatcher>REQUEST</dispatcher>
</filter-mapping>
```

Datenübertragung II – Caching

Methoden:

`doFilter()` Methode, welche die Anfrage bekommt und verarbeitet
`init()` Initialisierung des Filters (einmalig beim Deployen)

Vereinfachtes Beispiel:

```
public class ExpiresFilter implements Filter {

    @Override
    public void doFilter( ServletRequest request, ServletResponse response)
        throws IOException, ServletException {

        // Schreibe in die Response
        response.setHeader( "Expires", "Thu, 01 Dec 1994 16:00:00 GMT");
    }

    @Override
    public void init( FilterConfig filterConfig ) {
        // Lese einen Parameter des Filters aus der web.xml
        String expiresAfter = filterConfig.getInitParameter("days");
    }
}
```

Komplettes Beispiel im Source-Ordner zur Vorlesung

Datenübertragung III – AppCache

Verschiedene Technologien können das Caching steuern

Service Workers:

- Welche Ressourcen im Cache bleiben sollen, wird durch ein Skript gesteuert
- Ermöglicht der Anwendung dynamisches Caching
- Aktuelle Technologie

Cache-Manifest:

- Datei mit Anweisungen zum Cache-Verhalten festgelegter Ressourcen
- Statische Festlegung, entweder fehlen Ressourcen oder es sind zuviele

html-equiv-header:

- Steuerung des Cachings einer einzelnen (HTML) Seite
- Verwendet Tags
- Keine Beeinflussung des Cachings von referenzierten Dokumenten

Client-Server-Kommunikation

1. Kontext und Motivation
2. Dynamisches Laden
3. WebServer
4. Datenübertragung
- 5. Authentifizierung**
6. Session-Management
7. Darüber hinaus
8. Projekt

Authentifizierung I – HTTP Auth

Definition: Das HTTP Protokoll definiert ein paar einfache Methoden um den Zugriff auf Dateien zu beschränken.

HTTP – Basic

- seit HTTP 1.0
- Benutzername und Passwort werden unverschlüsselt übertragen
 - wenig sichere Datenübertragung

HTTP – Digest Authentifizierung

- seit HTTP 1.1
- symmetrische Verschlüsselungstechnik
- Passwort wird nicht an den Server übertragen
- Vorgang:
 1. Server generiert Zufallswert
 2. clientseitige Verknüpfung des Zufallswertes mit dem Passwort
 3. von der Verknüpfung wird ein Hash-Code berechnet
 4. Übertragung des Hashcodes an den Server

Authentifizierung I – HTTP Auth

Ablauf von HTTP Auth:

1. GET-Request

```
GET /secure_document.html HTTP/1.0
Accept: image/gif, image/jpeg, */*
Accept-charset: iso-8859-1, *, utf-8
Accept-encoding: gzip
Accept-language: en
User-Agent: Mozilla/4.51 [en] (WINNT; I)
```

2. Server-Antwort mit Status 401 Unauthorized

```
/HTTP/1.1 401 Unauthorized
Date: Mon, 13 Jan 2003 08:35:41 GMT
Server: Apache/1.3.24 (Win32) PHP/4.3.0
WWW-Authenticate: basic realm="geschuetzterBereich"
```

3. Browser fragt nach Benutzernamen und Passwort

4. GET-Request aus 1 wird noch einmal gesendet mit zusätzlichem Authorization-Header

```
GET /secure_document.html HTTP/1.0
Accept: image/gif, image/jpeg, */*
Accept-charset: iso-8859-1, *, utf-8
Accept-encoding: gzip
Accept-language: en
User-Agent: Mozilla/4.51 [en] (WINNT; I)
Authorization: Basic aGVpa286d29laHI
```

Bemerkung: Username und Passwort werden im Authorization Header Base64 – kodiert (aber unverschlüsselt) versendet

Authentifizierung II – Application Auth

Definition: Bei der Authentifizierung über eine Anwendung werden die Daten über ein Formular in der Webanwendung an den Server weitergereicht.

Eigenschaften:

- Übertragung durch Parameterstrings (HTTP - GET) oder Datenpakete (HTTP - POST)
- Bei Nutzung des HTTPS erfolgt gesamter Datentransfer verschlüsselt
- Bei Nutzung von HTTP-GET werden Parameterstrings in Logfiles mitgeschrieben
 - Authentifizierungsformulare immer über POST und mit HTTPS!

Authentifizierung III – HTTPS Auth

Definition: Bei der Authentifizierung über HTTPS (SSL) wird der Anwender über einen Schlüsselaustausch identifiziert.

Eigenschaften:

- Nutzer benötigt ein digitales Zertifikat von einer *Certifying Authority (CA)*
- CA 's sind kommerzielle Organisationen
- Digitale Zertifikate enthalten Stammdaten des Zertifikateigners sowie einen öffentlichen Schlüssel
- Zertifikat wird an den Server übertragen und identifiziert den Client eindeutig

Authentifizierung III – HTTPS Auth

Praktische SSL-Implementierung (Secure Socket Layer):

Stellt Kombination des symmetrischen und asymmetrischen Verschlüsselungsverfahrens dar mit folgender zugrundeliegender Idee:

1. Server überträgt sein **digitales Zertifikat** und **öffentlichen Schlüssel** an den Browser
 2. Browser generiert einen **zufälligen Session-Key**
 3. Die Nachricht wird **mit dem Session – Key symmetrisch kodiert**
 4. Session – Key wird mit öffentlichem Schlüssel des Servers verschlüsselt (auch als **digitaler Briefumschlag** bezeichnet)
 5. Ergebnis aus 3 und 4 (**sym. verschlüsselte Nachricht und verschl. Session Key**) werden an den Server übertragen.
 6. **Server extrahiert Session – Key** unter Verwendung seines privaten Schlüssels
 7. Server **entschlüsselt die Nachricht symmetrisch** unter Verwendung des Session – Keys.
- Einmalig im digitalen Briefumschlag übertragener **Session – Key** wird später weiter benutzt für symmetrische Verschlüsselung aller zwischen Browser und Server ausgetauschten Nachrichten

Client-Server-Kommunikation

1. Kontext und Motivation
2. Dynamisches Laden
3. WebServer
4. Datenübertragung
5. Authentifizierung
- 6. Session-Management**
7. Darüber hinaus
8. Projekt

Session- Management I

Definition: Eine Session beschreibt einen Dialog, der sich über mehrere Requests und Responses erstreckt. Session Management befasst sich mit dem Sammeln und Speichern von Informationen innerhalb einer Session.

Allgemeine Eigenschaften von Sessions:

- Identifizierung des Benutzers einer Session
- Speichern von Zuständen

Techniken:

- Clientseitig
 - Hidden Fields
 - URL Rewriting
 - Cookies
- serverseitig

Session- Management II - Hidden Fields

Definition: Hidden Fields sind HTML-Formularfelder, die name = “wert” – Paare an den Server übertragen und die für den Benutzer nicht sichtbar sind.

Vorteile

- Generelle Unterstützung
- Client bleibt im Dialog anonym

Nachteile

- Erzwungene Kontinuität
- Bei Unterbrechung des |



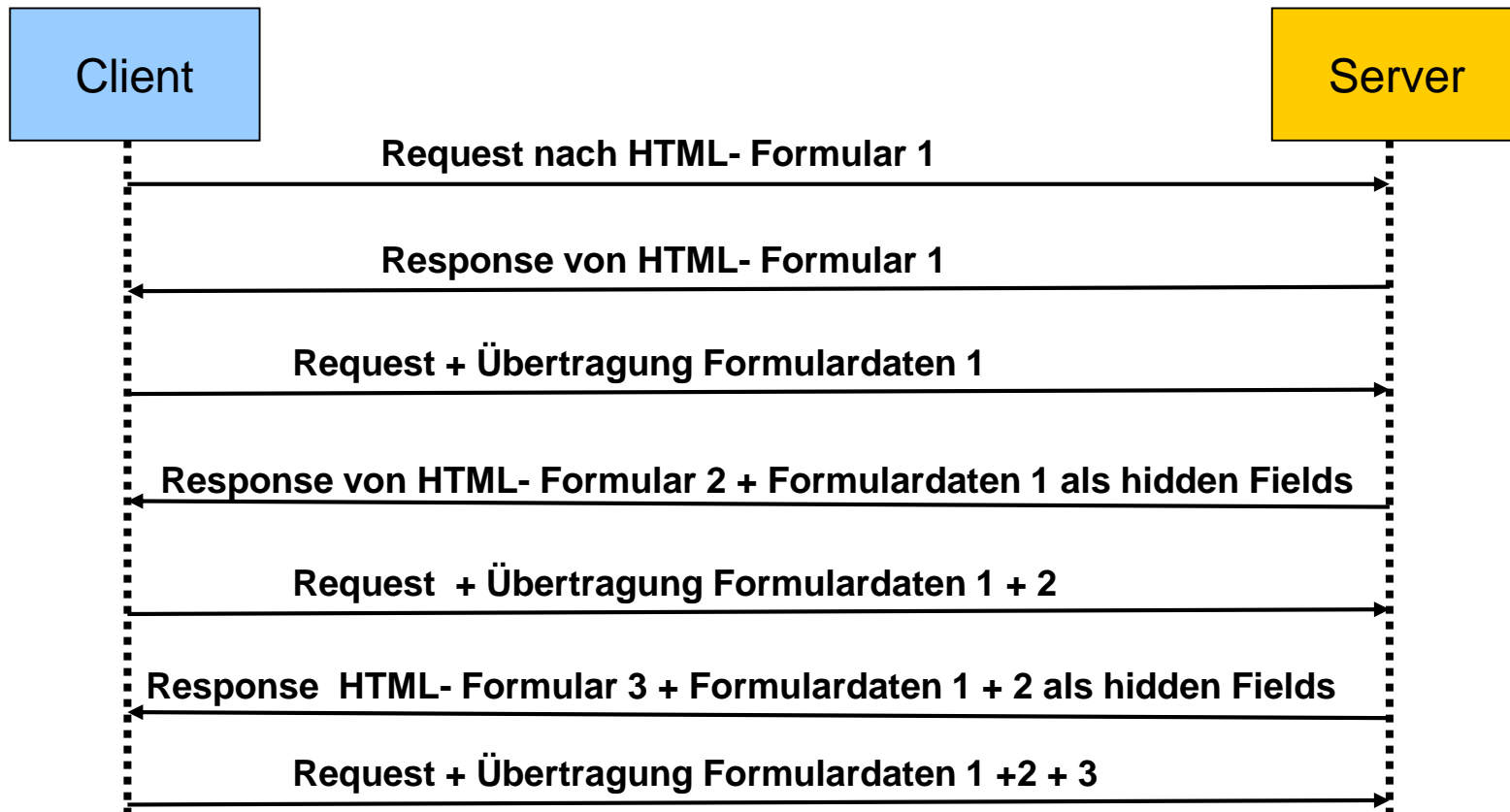
nder folgender Formulare

Verlust der bereits übertragenen Daten

Session- Management II - Hidden Fields

Definition: Hidden Fields sind HTML-Formularfelder, die name = "wert" – Paare an den Server übertragen und die für den Benutzer nicht sichtbar sind.

Beispieldialog für „Füllen eines Warenkorbes“ mit *hidden fields* schematisch:



Session- Management II - Hidden Fields

```
<form method="get" action="forms-buttons.html">
  <fieldset>
    <legend>Personalia:</legend>
    <label for="name">Name:</label><br>
    <input id="name" type="text"><br>
    <label for="email">Email:</label><br>
    <input id="email" type="text"><br>
    <label for="birth">Date of birth:</label><br>
    <input id="birth" type="text"><br>
    <input type="hidden" name="art1" value="Artikel_1">
  </fieldset>
  <input type="submit" value="Submit">
</form>
```

```
<form method="get" action="forms-buttons.html">
  <fieldset>
    <legend>Personalia:</legend>
    <label for="name">Name:</label><br>
    <input id="name" type="text"><br>
    <label for="email">Email:</label><br>
    <input id="email" type="text"><br>
    <label for="birth">Date of birth:</label><br>
    <input id="birth" type="text"><br>
    <input type="hidden" name="art1" value="Artikel_1">
    <input type="hidden" name="art2" value="Artikel_2">
  </fieldset>
  <input type="submit" value="Submit">
</form>
```

Session- Management II – URL Rewriteing

Definition: Beim URL Rewriting werden Session- Informationen als Teil der URL an den Server übertragen.

- Realisierung in Form von GET-Parametern.
- Alle Links auf einer Seite müssen um aktuelle Session- Informationen ergänzt werden.

Nachteile

- Informationen werden offen übertragen -> geringe Datensicherheit
- Abspeichern inaktueller Favoriteneinträge im Browser

```
<a href="nextstep.html?sessionid=82djw83d2">Nächster Schritt</a>
```

Session- Management III – Cookies

Definition: Beim Cookies sind kleine, im Browser gespeicherte Datenblöcke, die bei jedem HTTP-Request zum Server übertragen werden.

Eigenschaften

- sind eine Erweiterung des HTTP-Protokolls
- sind durch einen Namen und einen Wert definiert
- Größe eines Cookies ist auf 4 kB beschränkt.
- Anzahl ist beschränkt: meist 20 Cookies / Domain

Vorteile

- nicht an HTML gebunden
- Unabhängig von Dialoghistorie
- Bleiben je nach Konfiguration sehr lange erhalten
-> Session kann nach längerer Zeit wieder aufgenommen werden

Nachteile

- Können vom User
- Beschränkung in C



nden werden

nd Anzahl pro Server

Session- Management III – Cookies

Ablauf bei der Verwendung von Cookies:

1. Client fragt Server nach einer Seite unter Angabe von Session bezogenen Informationen

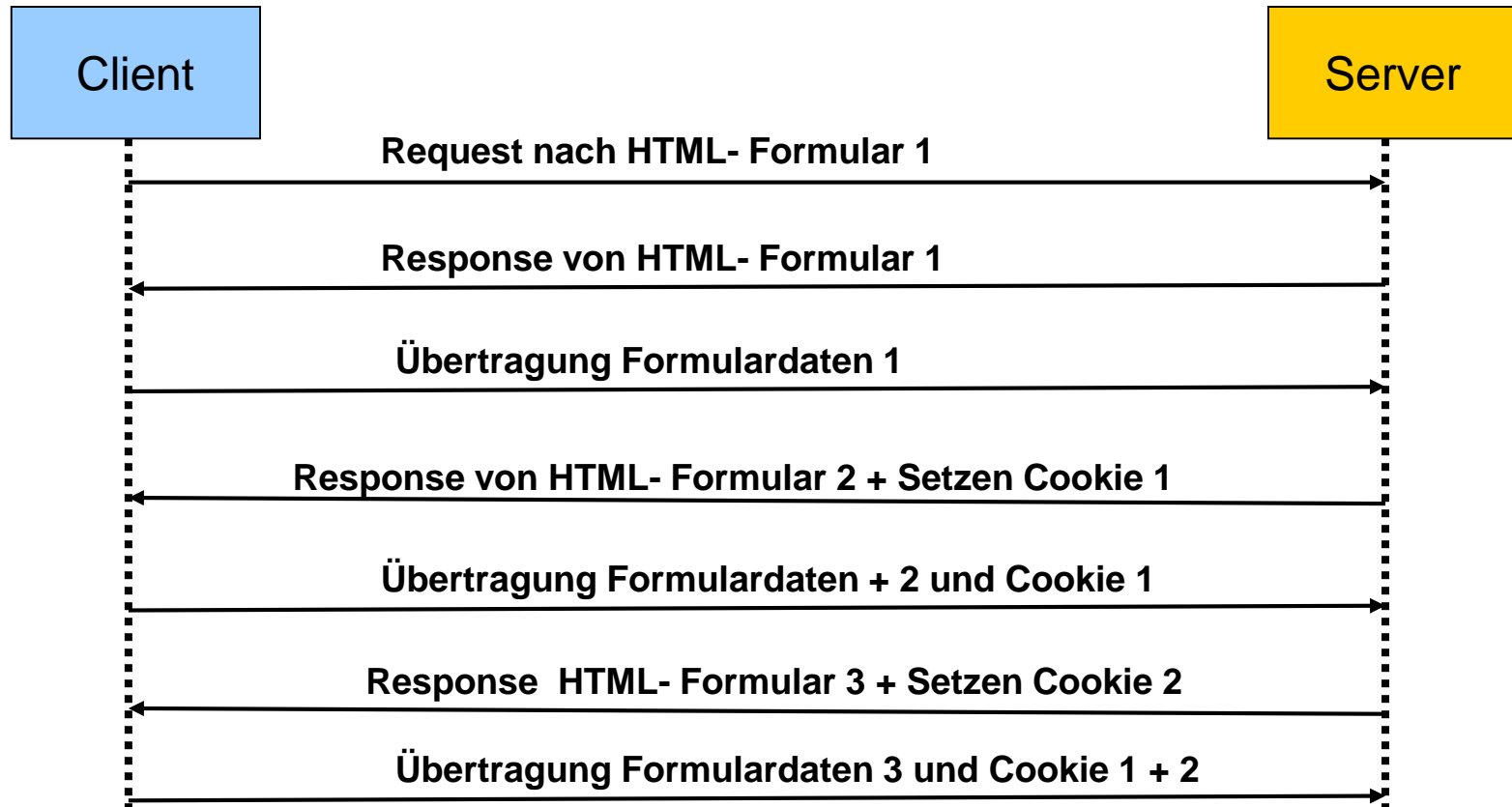
```
HTTP/1.1 200 OK
Date: Mon, 13 Jan 2003 08:35:41 GMT
Server: Apache/1.3.24 (Win32)
      PHP/4.3.0
Set-Cookie: name=wert
```

2. Server antwortet unter Angabe der Session – Informationen an den Client durch Nutzen des „Set-Cookie“ HTTP-Response – Headers

```
GET /secure_document.html HTTP/1.0
Accept: image/gif, image/jpeg, */*
Accept-charset: iso-8859-1, *, utf-8
Accept-encoding: gzip
Accept-language: en
User-Agent: Mozilla/4.51 [en] (WINNT;
I)
Cookie: name=wert; name2=wert2
```

3. Bei jedem weiteren HTTP-Request wird das Cookie wieder an den Server übertragen im HTTP – Request-Header

Session- Management III – Cookies



Session- Management III – Serverseitig

Nachteile clientseitigen Session-Managements:

- Wiederholter Transport der Zustandsinformationen im Netz
 - Performance – Verluste
 - beeinträchtigte Sicherheit der Daten
- Nutzer kann Daten auf dem Client einsehen und evtl. modifizieren
 - beeinträchtigte Sicherheit der Serverapplikation

Bei großen Webapplikationen werden Session – Informationen auf dem Server gespeichert.

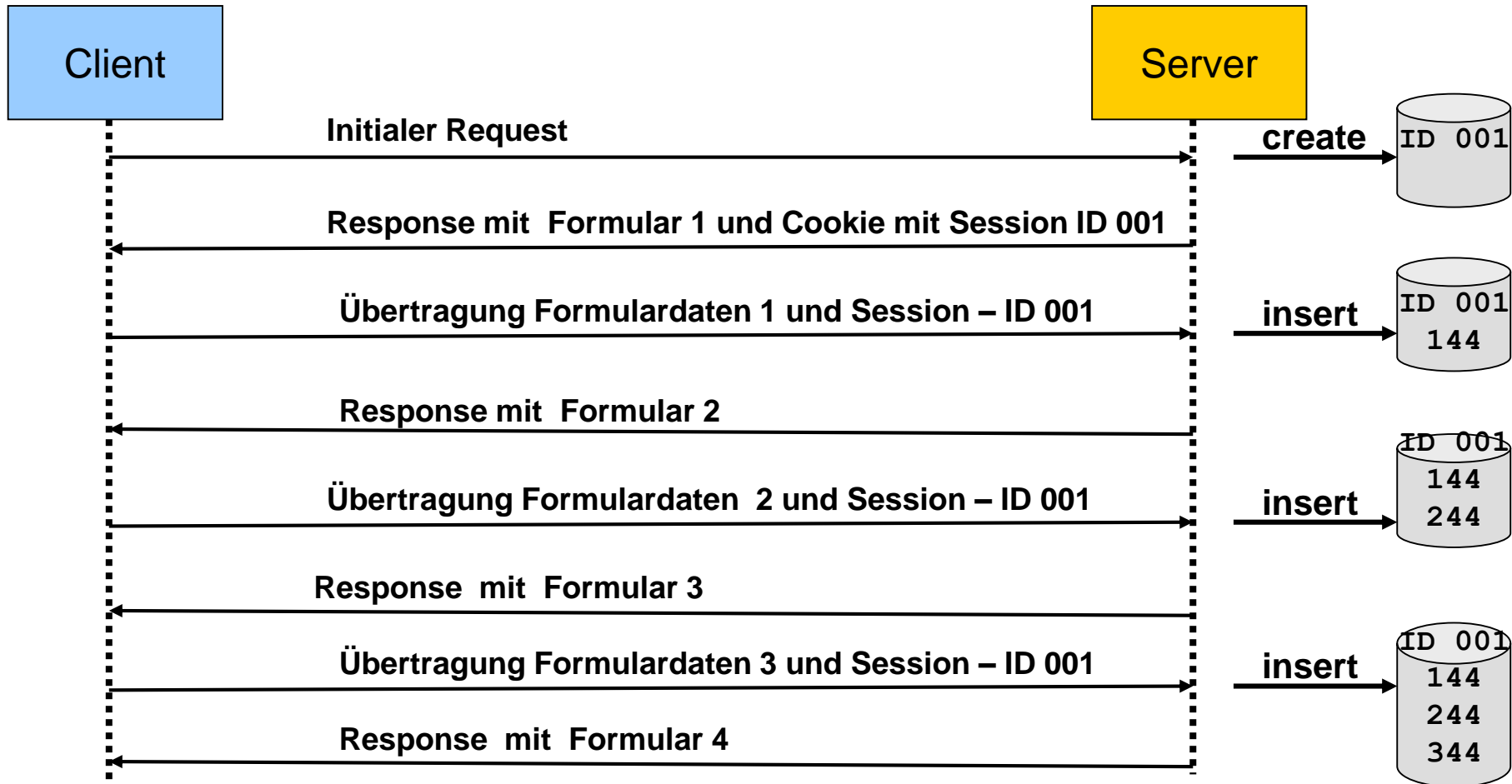
Die serverseitigen Speichermöglichkeiten sind vielfältig und müssen den Bedürfnissen der Applikation angepasst werden.

Session- Management III – Serverseitig

Technologiebeispiel:

- Server erzeugt für jede Session eine Session-ID
- Speicherung von Zustandsinformationen unter eindeutiger Session-ID
- Session-ID wird einmalig an den Client übertragen
- Die Session-ID wird bei jeder Anfrage des Clients mit an den Server gesendet (z.B. mit Cookie, URL-Rewriting oder Hidden Fields)
- Dauer einer Session wird begrenzt durch:
 - Programmlogik (z.B. Ende eines Bestellvorganges)
 - serverseitigen Timeout
 - Benutzer (z.B. wenn Cookie gelöscht wurde)

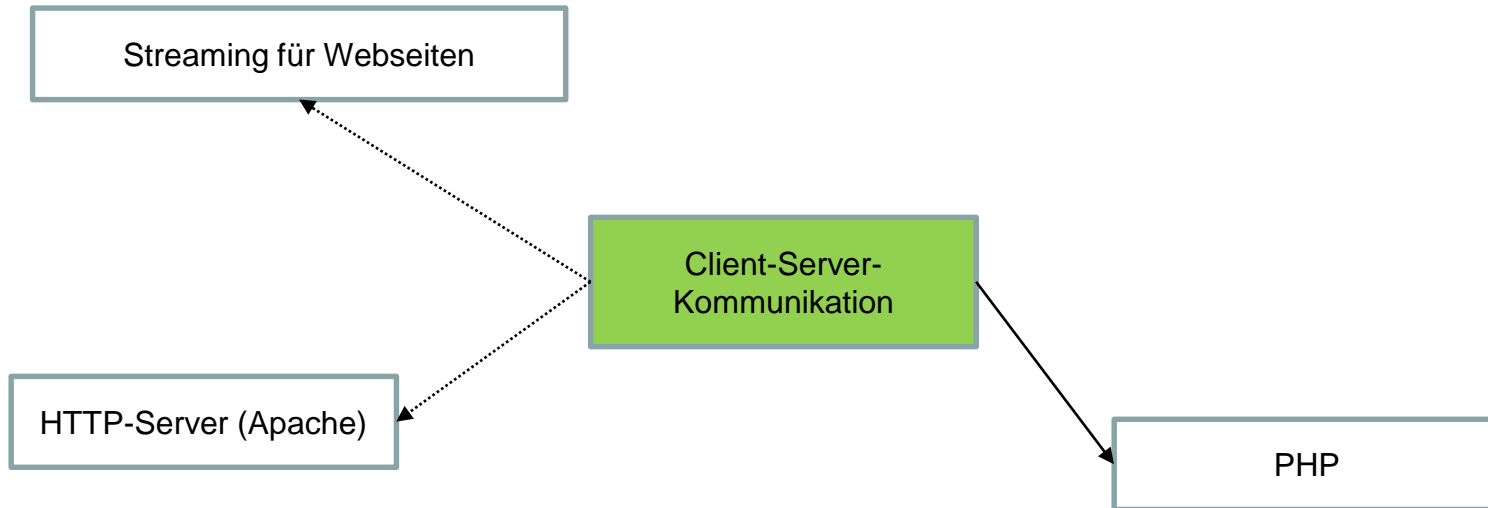
Session- Management III – Serverseitig



Client-Server-Kommunikation

1. Kontext und Motivation
2. Dynamisches Laden
3. WebServer
4. Datenübertragung
5. Authentifizierung
6. Session-Management
- 7. Darüber hinaus**
8. Projekt

Darüber hinaus



Links:

Apache HTTP-Server
PHP
Streaming

- <http://httpd.apache.org/>
- <http://php.net/manual/de/intro-what-is.php>
- [https://developer.mozilla.org/en-US/Apps/Fundamentals/Audio_and_video_delivery/Live_streaming w](https://developer.mozilla.org/en-US/Apps/Fundamentals/Audio_and_video_delivery/Live_streaming_w)

Client-Server-Kommunikation

1. Kontext und Motivation
2. Dynamisches Laden
3. WebServer
4. Datenübertragung
5. Authentifizierung
6. Session-Management
7. Darüber hinaus
- 8. Projekt**

Anforderungen

Welche Anforderungen werden als nächstes bearbeitet?

TODO

- Externe Inhalte einbinden
- Artikel vom Server einbinden
- Kommentare vom Server
- Artikel zum Server übertragen
- Kommentare zum Server
- Medien zum Server
- Kommentare speichern
- Kommunikation untereinander

DONE

- Technologische Grundlagen erarbeiten
- Was ist eine Web-Anwendung?
- News darstellen
- Projekte vorstellen
- Aufgaben darstellen
- Formular für Kommentare
- Schickes Design für die Seite
- Mediendateien einbinden
- Animationen
- Mehrsprachen-Fähigkeit
- (lokales) Speichern von Artikeln
- Client-Position anzeigen
- Offline-Verwendung ermöglichen
- Inhaltsverzeichnisse
- Formlareingaben in Seite einfügen
- Navigation über Tastaturkürzel