INFORMATIONSINTEGRATION UND WEBPORTALE

Vorlesung # 01, 27.10.2014 : Web-Technologien im Überblick

Dr. Andreas Walter
mail@awalter.info

Klick-And-Bau

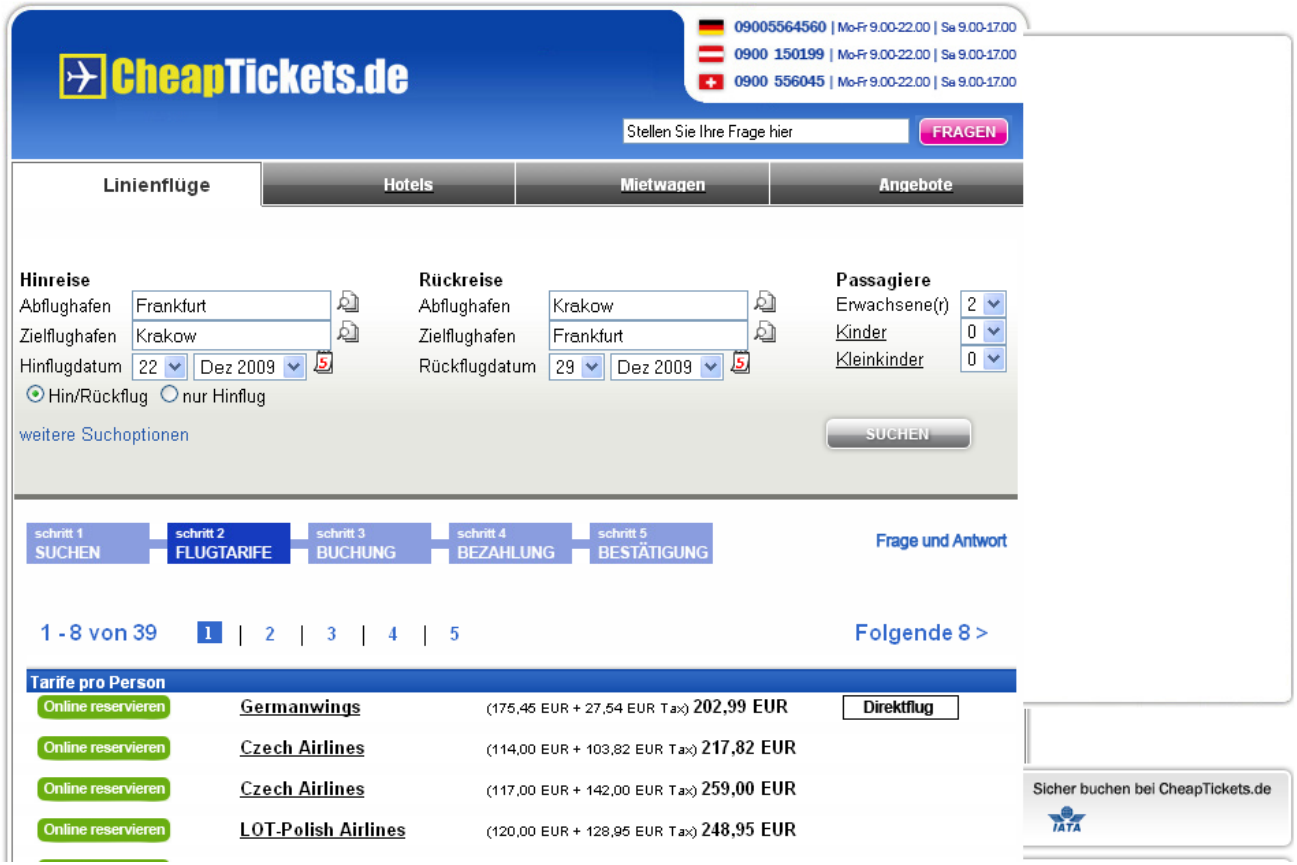
Informationsintegration und Webportale

Webanwendungen: Statische Webseiten

- **Statische Webseiten:** z.B. für eine Ferienwohnung

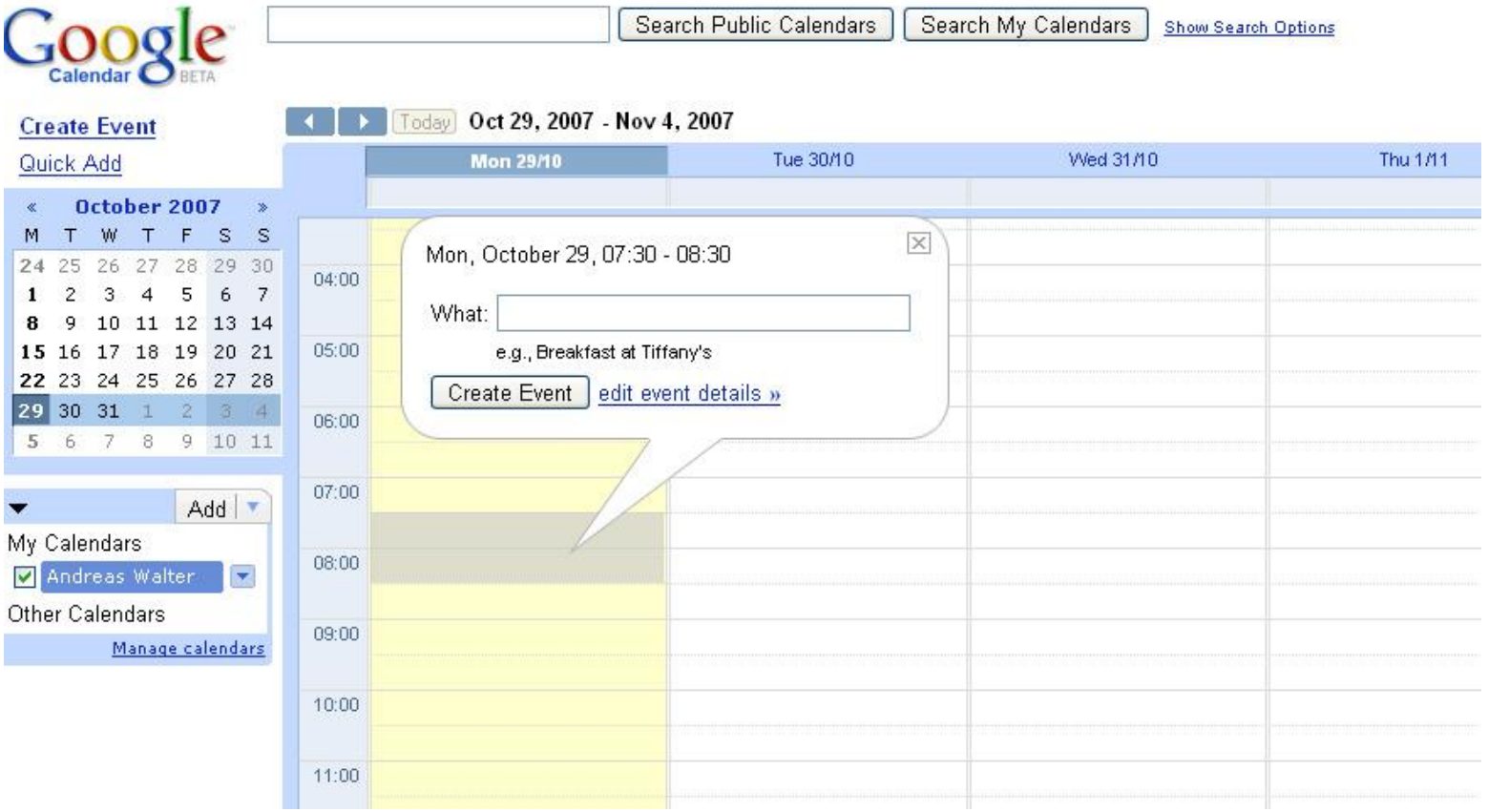
Webanwendungen: Dynamische Webseiten

- **Dynamische Webseiten: z.B. Portale zur Flugbuchung**



Webanwendungen: Anwendungen

- **Webseiten im Stil von Desktop-Anwendungen**



Inhalt

- **Einordnung**
  Webtechnologien - Informatiosinstegration und Webportale
- Beispiel einer umfangreichen Webanwendung
  ii4sm – *Patient Recruitment System*

- Grundlagen Webtechnologien
- Desktop vs. Webanwendungen
- Webtechnologien für kleinere Webanwendungen
- Anforderungen bei der Erstellung umfangreicher Webanwendungen
- Konzepte von Webrahmenwerken

Informationsintegration und **Webportale**

- Webportale
  - Früher mehr eine Linksammlung zu interessanten Seiten eines Themas, z.b. Yahoo! Webkatalog
  - Heutzutage: umfangreiche Webanwendungen, die unter einheitlichem Look & Feel z.b. Waren zu einem Bereich verschiedener Lieferanten anbieten.
- Informationsintegration
  - Integration / Speicherung nötiger Daten für eine Webanwendung
  - Aufbereitung / Verständnis für Daten
  - Verarbeitung von Events

Vorlesung heute: **Webtechnologien**

- Konzepte zum Aufbau umfangreicher Webanwendungen
- Ziel ist Auslieferung von Inhalten auf unterschiedliche Devices

Ii4sm – International Institue for the Safety of Medicinces, Basel

Dr. Andreas Walter, Head of Development

http://www.ii4sm.com

Ii4sm - Patient Recruitment System

BEISPIEL EINER UMFANGREICHEN WEBANWENDUNG

li4sm Patient Recruitment System

Domäne Pharma, Medikamentensicherheit

- Pharma entwickelt neue Medikamente, erhält Patent
  Muss dann beweisen, dass Wirkung vorhanden, u.a. keine tödlichen Nebenwirkungenen => klinische Studien, Studienprotokolle
- Klinische Studien werden in u.a. Krankenhäusern durchgeführt, erfordert Auswahl geeigneter Patienten, z.b. mit Pneumonie : manueller Prozess

li4sm Patient Recruitment System

- Automatisierte Suche geeigneter Patienten (Patent pending)

Fokus nun (Ausschnitt): Online Recruitment

Online Recruitment Anwendungen

Prozessschritte Online Recruitment

1. **PRS Serverinstallation** in Krankenhaus (Datenintegration, Big data)
2. **PRS Registry**: Verwaltung von klinischen Studien / Benutzern, Krankenhäusern
3. **PRS Coding**: Patientendaten sind kodiert mit Terminologien (Ontologien) => semantische Integration.
   Coding: Erstellung eines semantischen Studienprotokoll
4. **PRS Search**: Aktivierung von Studienprotokoll in Registry
   PRS startet regelmässige Suche von Patienten in Patientendaten
   (Anmerkung: hier auch sehr viel Datenanonymisieurng integriert!)
5. **PRS Validation**: Passende Kandidaten werden gesammelt und in PRS Validation GUI angezeigt.
   Treffer werden validiert, wenn passend, Patienten angefragt, ob sie an Studie teilnehmen wollen

PRS Registry

Verwaltung klinischer Studien / Benutzern, Krankenhäusern

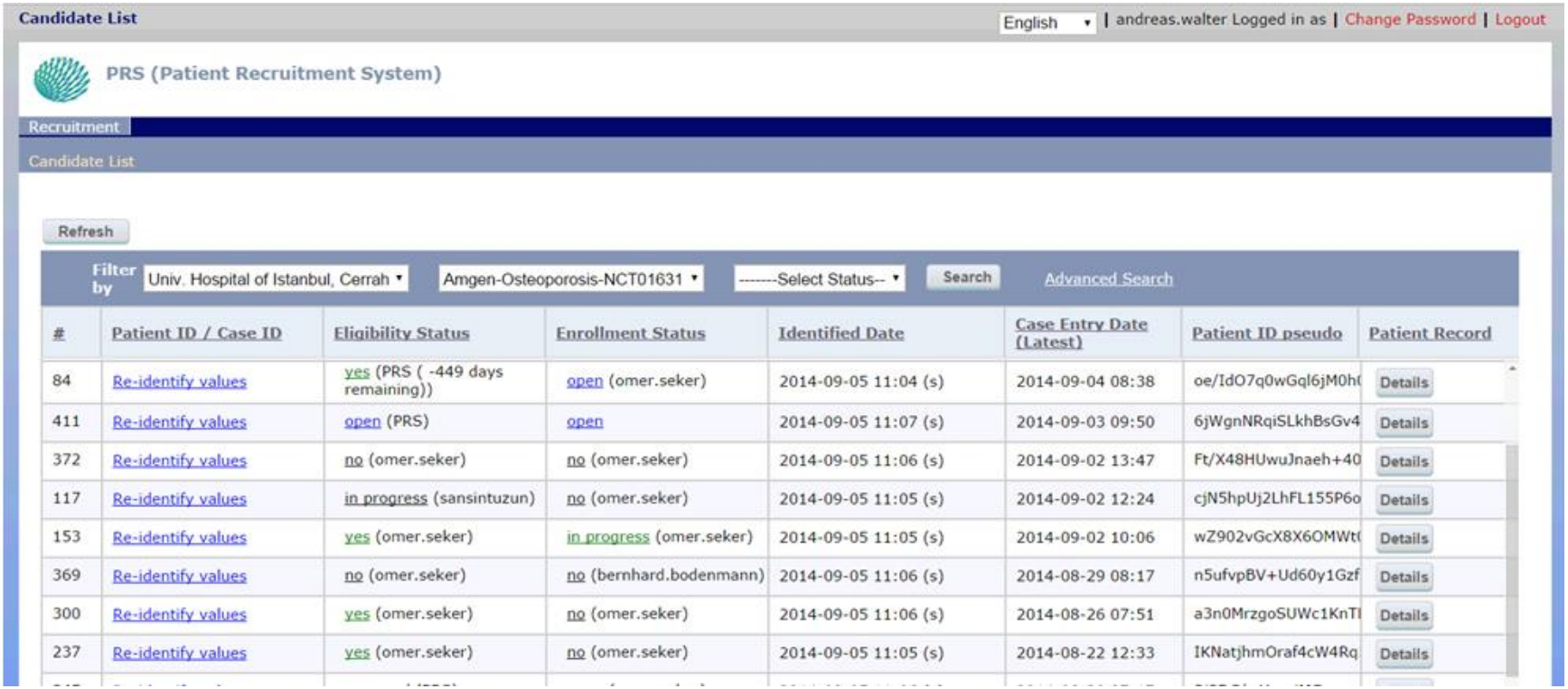
PRS Coding (Excel)

- Technisch: es werden hierarchische Protokolle erstellt, Verknüpfung mit boolschen Operatoren. Umsetzung zur Suche auf Index (erfordert natürlich zuerst Aufbau von Indexen. Verteilt, semantisch, skalierbar, ...)

	AND	MIX	system	system	[A] 1. Male or female, age >=		
		AND	DEMOGRAFIC	system	system	[A] Male or female with age >= 18 years	
			OR	DEMOGRAFIC	system	system	[O] male
			Demografic	<original_protocol_text>	MALE	18	<age to>
			OR	DEMOGRAFIC	system	system	[O] fema
			Demografic	<original_protocol_text>	FEMALE	18	<age to>
	AND	MIX	system	system	[A] 2. Diagnosis of CDI		
		OR	PROCEDURES	system	system	[O] c) positive test for toxigenic C. difficile from a	
		Procedure	[O] 907190 Clostridium difficile	SUT	907190	Clostridium difficile toxin-A ve B	<freetext
EXCLUSION							
AND	EXCLUSION	system	system	Protocol Exclusions			
	OR	MIX	system	system	des		
		OR	PROCEDURES	system	system	[O] a) CDI (Clostridium difficile-infection)	
		Procedure	[O] 901420 Gebelik testi (idrarda)	SUT	901420	Gebelik testi (idrarda)	<freetext
		OR	DIAGNOSES	system	system	[O] a) CDI (Clostridium difficile-infection)	
		Diagnose	<original_protocol_text>	ICD10	Z32	Gebelik muayenesi ve testi	
		Diagnose	<original_protocol_text>	ICD10	Z33	Gebelik durumu, tesadüfi	
		Diagnose	<original_protocol_text>	ICD10	Z34	Normal gebeliğin gözlemi	

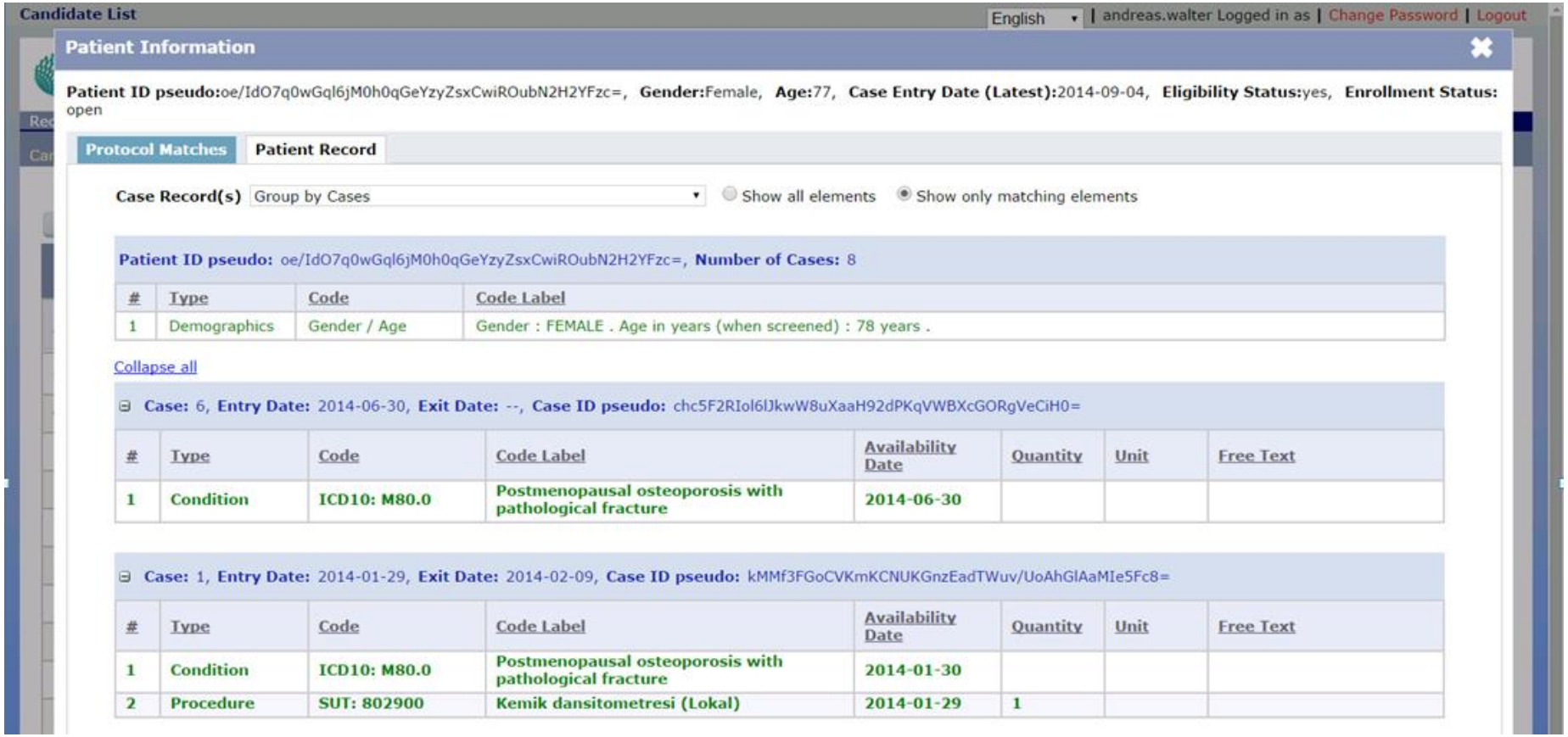
PRS Validation (1)

Passende Kandidaten werden gesammelt und GUI angezeigt



PRS Validation (2)

Detailansicht Patientenakte



Blick auf PRS Webanwendung

- Verschiedene Web-Frontends nötig, einheitlicher Login (Single Sign On), einheitliche Darstellung der Seiten (Look And Feel)
- **Mehrsprachigkeit**: aktuell u.a. Installationen in Türkei, Schweiz, Deutschland – Erweiterung in andere Sprachen
- **Permanente Notwendigkeit neuer Funktionen**
  Waterfal Model hier unmöglich.
  In Praxis: Scrum, Sprint basiert auf Featuer Driven Development
- Umfangreiche Erstellung von Tests nötig – **Quality Management**
- **Nicht alles muss ein Web-GUI sein**
  Fachpersonal zur Erstellung von Protokollen ist Excel gewohnt
  => Excel als Frontend, Transformatoren zur Erstellung von Suchanfragen

- Verwendetes Webrahmenwerk: Google Web Toolkit
  Gründe u.a. : schnelle GUIs, einfache Erlernbarkeit für Entwickler

Worauf basieren Webanwendungen?

GRUNDLAGEN

BASISTECHNOLOGIEN IM WEB

http: Transportprotokoll im WWW

- **http**: HyperText Transfer Protocol (aktuell Version 1.1)
  - Von W3C standardisiertes Protokoll zum Abruf und Senden von Daten von/an einen Web-Server (RFC 2616)
    - Ursprünglich: Webseiten, Grafiken
    - Mittlerweile: auch für WebServices, Skype (Telefonie) etc.
      - Grund: Firewalls müssen Port 80 freigeben, um Abruf von Webseiten zu ermöglichen
    - Protokoll auf TCP/IP basierend mit 3 wichtigen Eigenschaften:
      - Verbindungslos: nach Anfrage trennt Client die Verbindung u. wartet auf Server-Antwort; Server muss Verbindung wieder aufbauen (http 1.1 ermöglicht „keep alive“ – nur für z.B. Chats sinnvoll)
      - Medienunabhängig: jede Art von Datentypen kann versendet werden, solange Client u. Server damit was anzufangen wissen
      - Zustandslos: Jeder Abruf von Daten erfordert neue Verbindung; Client kann keine Informationen zw. Anfragen/über Webseiten hinweg behalten
    - **Bereich MESSAGE HEADER**: Für Metainformationen (z.B. Anfragemethode, Statuscodes, Dokumenttyp, Länge des Dokuments), Authentifizierung
    - **Bereich MESSAGE BODY**: folgt dem HEADER, enthält eigentlichen Inhalt

Adressierung mit http-Protokoll

- **URI: Uniform Ressource Identifier**
  Identifizierung und Adressierung von Ressourcen

  <Schema>://[*<Benutzer>*[:*<Passwort>*]@]
  <Server>[:*<Port>*][/*<Pfad>*][?*<Anfrage>*][#*<Fragment>*]

- Davon abgeleitet: Uniform Ressource Locator (URL) für http
  z.B. http://www.fzi.de (bedeutet http://www.fzi.de:80/)

- httpRequest GET: senden von Parametern, z.B. über Adressfeld des Browsers:
  http://www.fzi.de/search.php?autor=Braun&ordering=newest
  - Einsatz: für Seiten, die über ein Skript generiert werden und gebookmarkt werden sollen
  - Daten als Teil der URL bleiben erhalten

- httpRequest POST: senden von umfangreichen Datenmengen, Authentifizierung aus Formular heraus.
  - Datenübertragung im BODY Bereich von http
  - Keine Speicherung der übertragenen Daten möglich

Beispiel Messages

- HTTP GET Request:

```
GET /search.php?autor=Braun&ordering=newest HTTP/1.1
Host: www.fzi.de
```

- HTTP POST Request:

```
POST /search.php HTTP/1.1
Host: www.fzi.de
Content-type: application/x-www-form-urlencoded
Content-length: 27

autor=Braun&ordering=newest
```

HEADER

BODY

Anmerkung: https (secured protocoll) – zusätzlich Austausch von SSL Zertifikaten

Zum Anschauen von http-Anfrage/Antworten:
http://web-sniffer.net/

Beispiel Messages

- HTTP Response

```
HTTP/1.1 200 OK
Date: Sun, 23 Oct 2011 10:23:03 GMT
Server: Apache/2.2.3 (CentOS)
Connection: close
Content-Type: text/html; charset=utf-8

<html xml:lang="de" … </html>
```

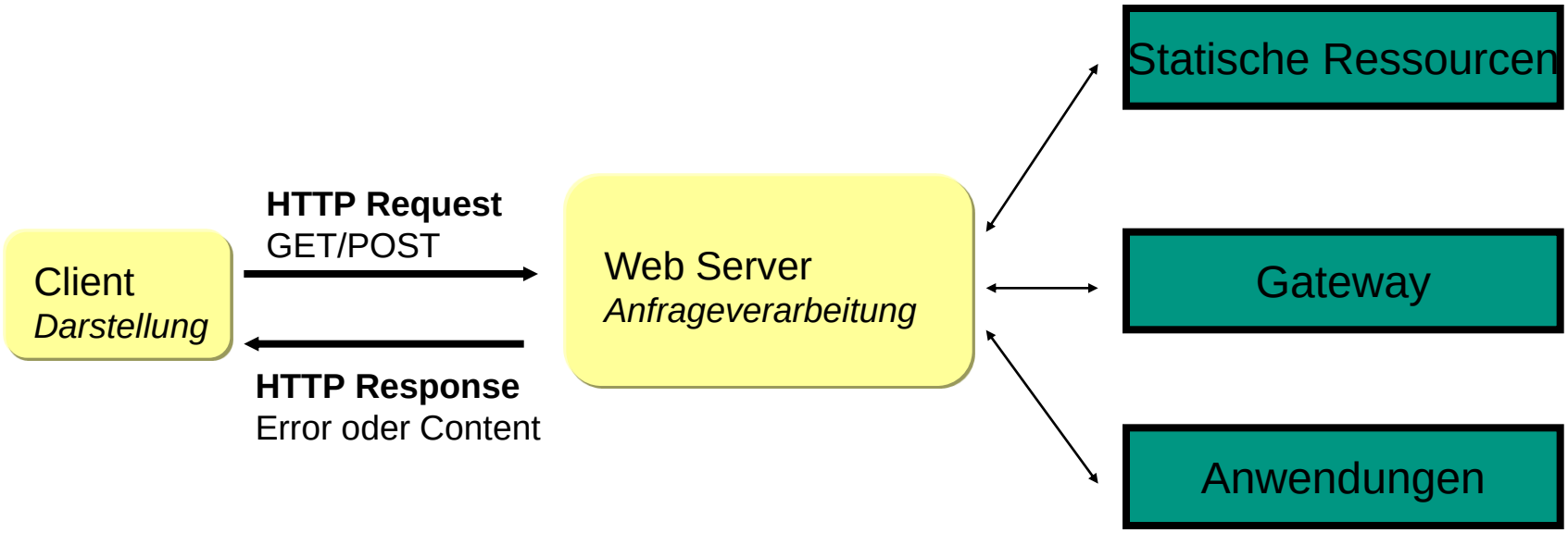
```
HTTP/1.1 404 Not Found
Date: Sun, 23 Oct 2011 10:23:03 GMT
Server: Apache/2.2.3 (CentOS)
Connection: close
Content-Type: text/html; charset=iso-8859-1
```

- Statuscodes: 5 Klassen
  - 1xx – informatorisch
    - 118 Connection timed out
  - 2xx – erfolgreich
    - 200 OK
  - 3xx – Umleitung
    - 301 Moved Permanently
  - 4xx – Client-Fehler
    - 403 Forbidden
  - 5xx – Server-Fehler
    - 503 Service Unavailable

Web Server

- Setzt http-Protokoll um, erreichbar über Port (z.B. 80 default)
  - Bearbeitung von Client Anfragen, bearbeitet diese und gibt wenn möglich einen Inhalt zurück, sonst http-Fehlercode

- Schematischer Aufbau



Browser

- Programm zum Abruf und zur Darstellung von Inhalten aus dem Internet
  - z.B. Abruf von Webseiten über das Protokoll http, zumeist auch andere Protokolle wie z.B. ftp (file transfer) integriert.
  - W3C Konsortium definiert Standards, welche alle Webbrowser können soll(t)en
    - HTML, CSS, JavaScript, XSLT
  - **Cache**: dient zur Speicherung von häufig verwendeten Inhalten, reduziert Datentransfer mit Server
  - **Cookies**: Über http können z.B. sessionrelevante Inhalte im Browser abgelegt werden, z.B. sessionID, Username, bevorzugte Sprache
  - **Plugins**: Browser sind über Plugins erweiterbar, z.B. Flash

Cache & Cookies

- HTTP Response

```
HTTP/1.1 200 OK
Date: Sun, 23 Oct 2011 10:23:03 GMT
Server: Apache/2.2.3 (CentOS)
Set-Cookie: d7ae737f0e17da5a64121bfb8fbe92c0=j4hcsnpjhg2fbfpucb6242a9t0; path=/
Set-Cookie: lang=deleted; expires=Sat, 23-Oct-2010 10:28:48 GMT; path=/
Expires: Mon, 1 Jan 2001 00:00:00 GMT
Last-Modified: Sun, 23 Oct 2011 10:28:50 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Connection: close
Content-Type: text/html; charset=utf-8

<html xml:lang="de" … </html>
```

Formatierungsanweisung: (X)HTML (1)

- **Hyper Text Markup Language**
  Textbasierte Formatierungssprache zur Bestimmung der Darstellung von Inhalten wie Texten, Bildern und Hyperlinks zu anderen Dokumenten in einem Browser

```
<HTML>
    <HEAD>
        <TITLE>Forschungszentrum Informatik</TITLE>
    <HEAD>
    <BODY>
        <IMG SRC=„logo.jpg“>
        …
        <A HREF=„kontakt.html“>Kontakt</A><BR/>
        …
    </BODY>
</HTML>
```

Kopf: Metadaten

Body: Seiteninhalt

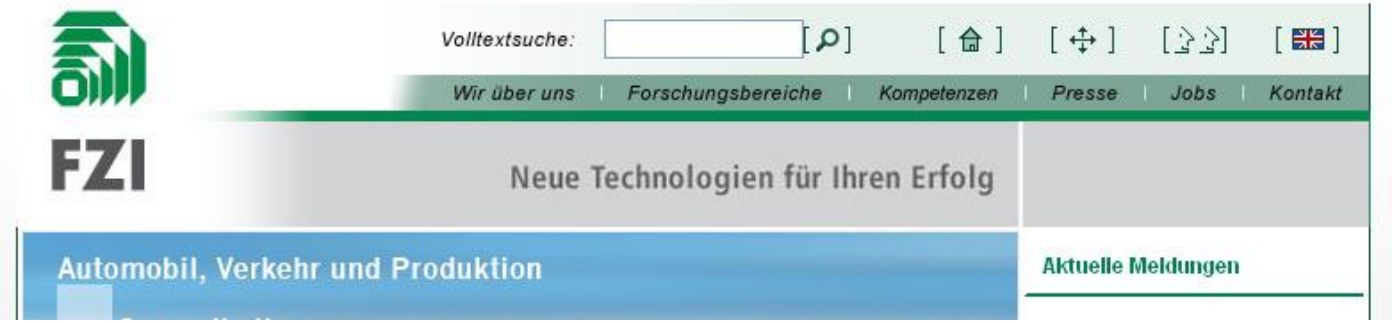
Bild

Link

Vgl. XML: XHTML wohlgeformt (HTML nicht)

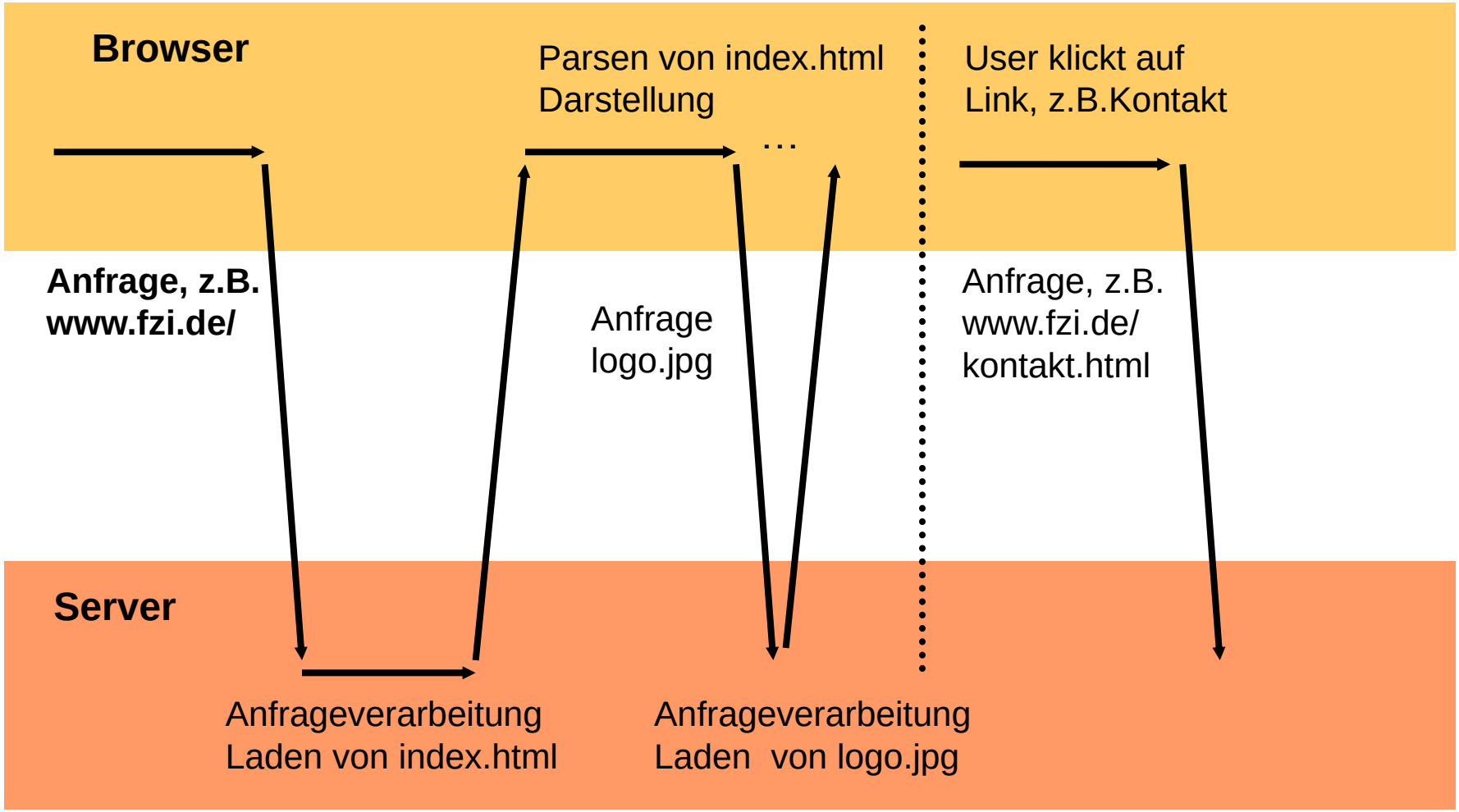
XHTML(2)

- Browser parst (X)HTML und zeigt dieses im Browser an



  - Weitere wichtige Elemente: Tabellen, Ebenen, Formulare

Kommunikation zwischen Browser und Webserver



Widerverwendung von Darstellung: CSS

- **Cascading Style Sheets**
  Legt fest, wie ein besonders ausgezeichneter Inhalt oder Bereich einer Webseite dargestellt werden soll.
  - Trennung von Inhalt (HTML, XML) und Darstellung
  - CSS als eigene Ressource ladbar (z.B. stylesmobile.css)
    - Ermöglicht unterschiedliche Ausgabe der gleichen Seite für verschiedene Medien (Browser, Tablet, Mobiltelefon, …)

- Beispiel:
  HTML: `<span class=„demostyle“>text mit css</span>`
  CSS: `.demostyle {color:black;background:red;font-weight: bold;}`

  Ausgabe: **text mit css**

Skripte in einer Webseite: Java Script

- **JavaScript** (!= JAVA!!!) ist objektbasierte Skriptsprache, direkt in HTML integrierbar oder als Ressource integrierbar
  - Ermöglicht die Client-seitige Ausführung von Skripten, durch Browser interpretiert und umgesetzt, z.B.
    - Eventhandling, z.B. Validierung von Formulareingaben
    - Interaktive Menüs, Drag & Drop Funktionen
    - Aktualisierung von Seitenteilen
  - In Verbindung mit XML und httpRequest → AJAX (später)

- **Problem**: JavaScript ist schwer wartbar
  - Ist objektbasiert, aber nicht typisiert oder wirklich objektorientiert
  - Schwer zu organisieren, debuggen und testen
  - Verhalten von JavaScript in Browsern nicht immer einheitlich
  - Wiederverwendung von Bibliotheken statt Selbsterstellung von Scripten wann immer möglich!

Java Script Beispiel

- **Javascript in HTML Text eingebettet**

```
<script type="text/javascript">
    function checkFormValues()
    {
      if (pw1 != pw2)
      {
                alert(„reenter password and pw are not equal“)>
                return false;
      }
}

<form name=„loginForm“ onSubmit=„checkFormValues()“>
    PW: <Input type=„password“ name=„pw1“/>
    Reenter: <input type=„password“ name=„pw2“/>
    <input type=„submit“ value=„login“/>
</form>
```

Skript bewirkt: Überprüfung von eingegebenen Passwörtern

Zusammenfassung Grundlagen

- **http**: Protokoll im Internet, Adressierung mit URLs
- **WebServer**: Serverseite, erweiterbar (z.B. Gateways)
- **Browser**: Clientseite, dient zur Darstellung von Inhalten
- **(X)HTML**: dient zur Formatierung von Webseiten
- **Kommunikation**: zustandslos, Request – Response Prinzip
- **CSS**: dient zur Trennung von Inhalt und Formatierung
- **JavaScript**: ermöglicht Ausführung von Skripten im Browser

... Die Unterschiede verschwinden

DESKTOP VS. WEBANWENDUNGEN

Desktop vs. Webanwendungen (I)

- **Desktop-Anwendungen** (ohne Netzwerkkommunikation)
  - „Fat-Client / Rich-Client“: Anwendungslogik, Darstellung (und ggf. Datenhaltung) in einer Software
  + Zustandshaltung, z.B. geöffnetes Worddokument
  + Umfangreiche Menüfunktionen
  + Schnelle Reaktion auf Eingaben durch Aktualisierung von einzelnen Teilen einer Seite.
  + Darstellung: Programmierer können Standardbibliotheken verwenden (Swing (JAVA) /…)
  - Meist sehr umfangreiche Installation erforderlich (in Firmen hierfür Kontakt mit SysAdmin nötig)
  - Updates nur manuell

Desktop vs. Webanwendungen (II)

- **Desktop-Anwendungen** (mit Netzwerkkommunikation)
  - Datenhaltung und (Teile der) Anwendungslogik ausgelagert, Software auf Client dient zur Darstellung
  + Weiterhin clientseitige Zustandshaltung (z.B. einmaliges Laden von Terminen → Darstellung als Kalender)
    Kommunikation mit Server nur zur Synchronisation
  + Client weniger umfangreich, Aktualisierung der Software bei jedem Start möglich (z.B. JAVA Web Start)
  - Weiterhin lokale Installationen notwendig
    - Entweder Interpreter (z.B. kompatible JAVA-Version)
    - oder Plugins (z.B. Flash Player)

Desktop vs. Webanwendungen (III)

- **Browserbasierte Webanwendungen**:
  - Webbrowser auf Clientseite wird zur Darstellung der Anwendung verwendet, Server für die Datenhaltung und Anwendungslogik

  \+ Keine Installation von Software auf Clientseite nötig (Sämtliche Betriebssysteme für PCs beinhalten Browser)

  „Web 1.0“ Stil:
  - Eine Webseite wird komplett aktualisiert für jede Anfrage (statisch)

  \- Ermöglicht keine Anwendungen mit „Desktop-feeling“

  „Web 2.0“ Stil
  - Aktualisierung von Seitenteilen möglich („Desktop-feeling“)

  \+ Ermöglicht die Erstellung von umfangreichen Webanwendungen im Stile einer Desktopanwendung

Entwicklung: Desktopanwendungen im Web

- Webanwendungen werden Desktopanwendungen immer ähnlicher, profitieren aber von den Vorteilen schlanker Clients
  - Verwendung des Browsers zur Darstellung, keine Installation von Plugins oder eigenständiger Software
  - Client immer auf neuestem Stand
  - Hauptgründe und Techniken hierfür:
    - JavaScript und CSS
    - AJAX (Details später)
    - **Breitbandzugänge (schnelles Laden von Seitenteilen möglich)**

… hier ist „skripten“ noch sinnvoll

WEB-TECHNOLOGIEN FÜR KLEINERE WEBSEITENPROJEKTE

Merkmale kleiner Webseitenprojekte

- Hoher Anteil an statischen (gleich bleibenden) Webseiten
- Merkmale der dynamischen Webseiten
  - Anwendungslogik ändert sich kaum bis selten,
    Wartbarkeit ist somit keine Anforderung
  - Gleichzeitige Teilnehmerzahl für Nutzung von dynamischen Seiten gering
    – Skalierbarkeit also keine Anforderung

- **Vorgehen generell**
  - Design der Webseite und statische Seiten erstellen (Grafiker)
  - Programmierer wählt eine geeignete Skriptsprache aus und erstellt Anwendung
  - Webspace bei Internetprovider anmelden, das Sprache unterstützt
  - Aufladen der Dateien (Daten) und Skripte: Fertig!

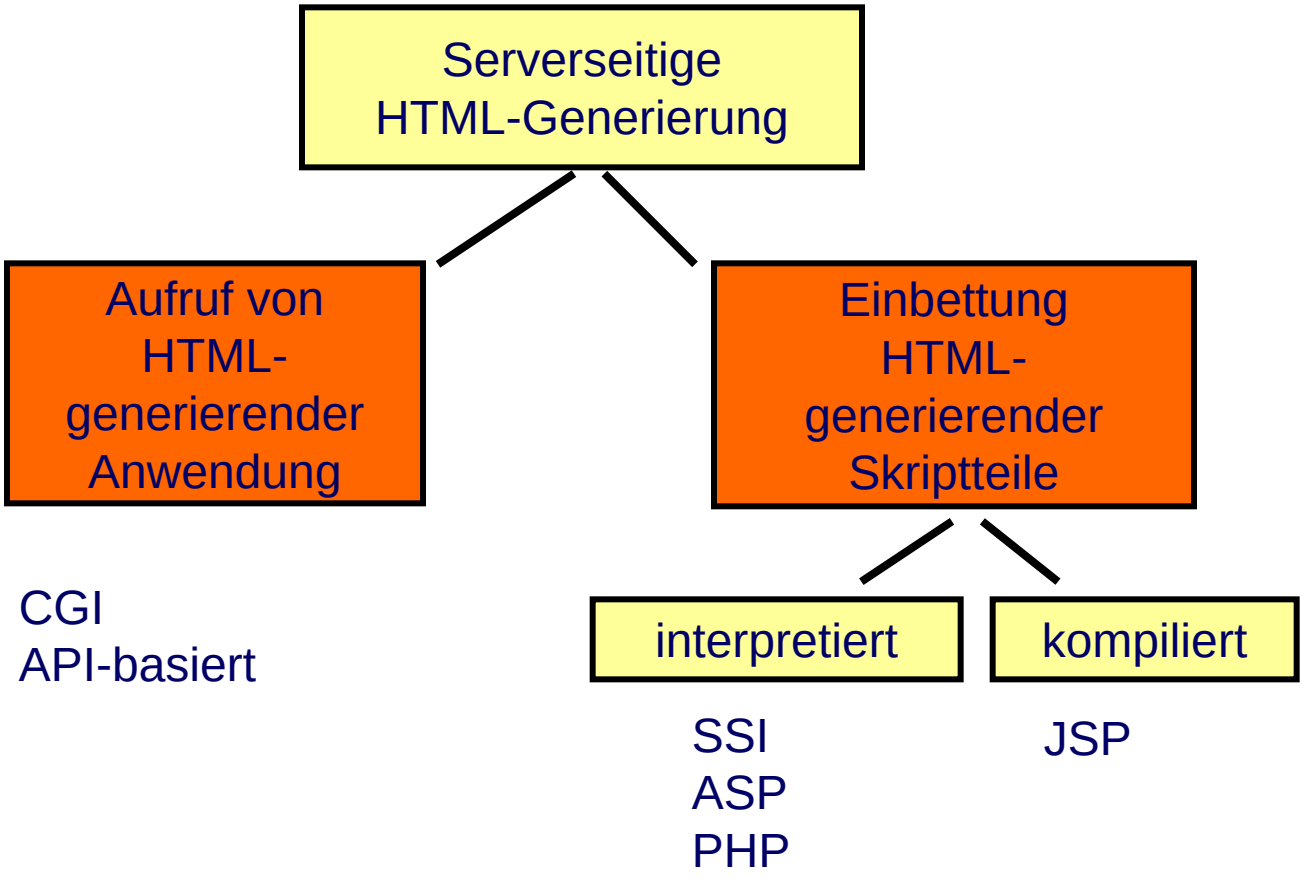
Vorgehen zur Erstellung dynamischer Seiten

Anwendung:

Für eine geplante Feier sollen sich Freunde und Verwandte online anmelden, um Catering planen zu können

1. HTML Formular erstellen, das Datenfelder beinhaltet
   - z.B. Name, Kontakt, Anzahl Personen, Auswahl Menüs, Vegetarisches Essen gewünscht, …
2. Skript erstellen, das diese Werte einliest
3. Anwendungslogik:
   - überprüfe, ob Teilnehmer bereits registriert ist
   - wenn nein, speichere Daten ab
4. Dynamisches Generieren einer Antwortseite
   - z.B. „Danke für Deine Registrierung“ oder „Du bist bereits registriert, möchtest du Deine Daten ändern?“

Serverseitige Anwendungslogik und Datenhaltung

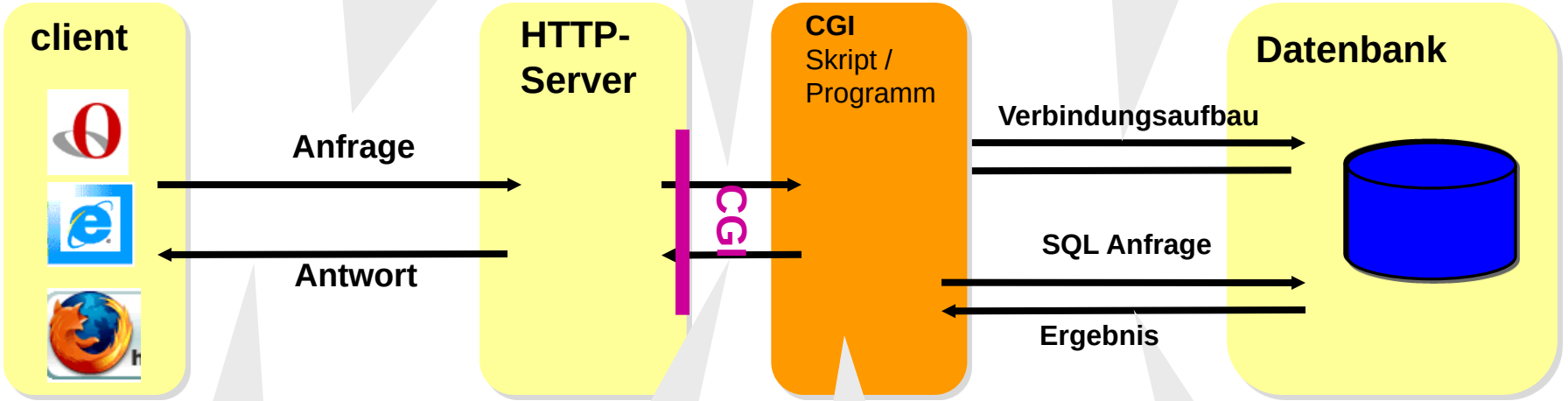


Die Old-School Variante: CGI

1) Betätigung Submit-Button <FORM METHOD="POST" ACTION="http://www.fzi.de/cgi-bin/my-form">

2) Aufruf des Skriptes mit Parametern **Initialisierung bei jedem Aufruf**

3) Aufbau der **DB-Verbindung bei jedem Aufruf**



7) HTTP Server Antwortet

6) Skript übermittelt HTML und **beendet sich**

5) Skript generiert HTML

4) Skript stellt Anfrage an DB

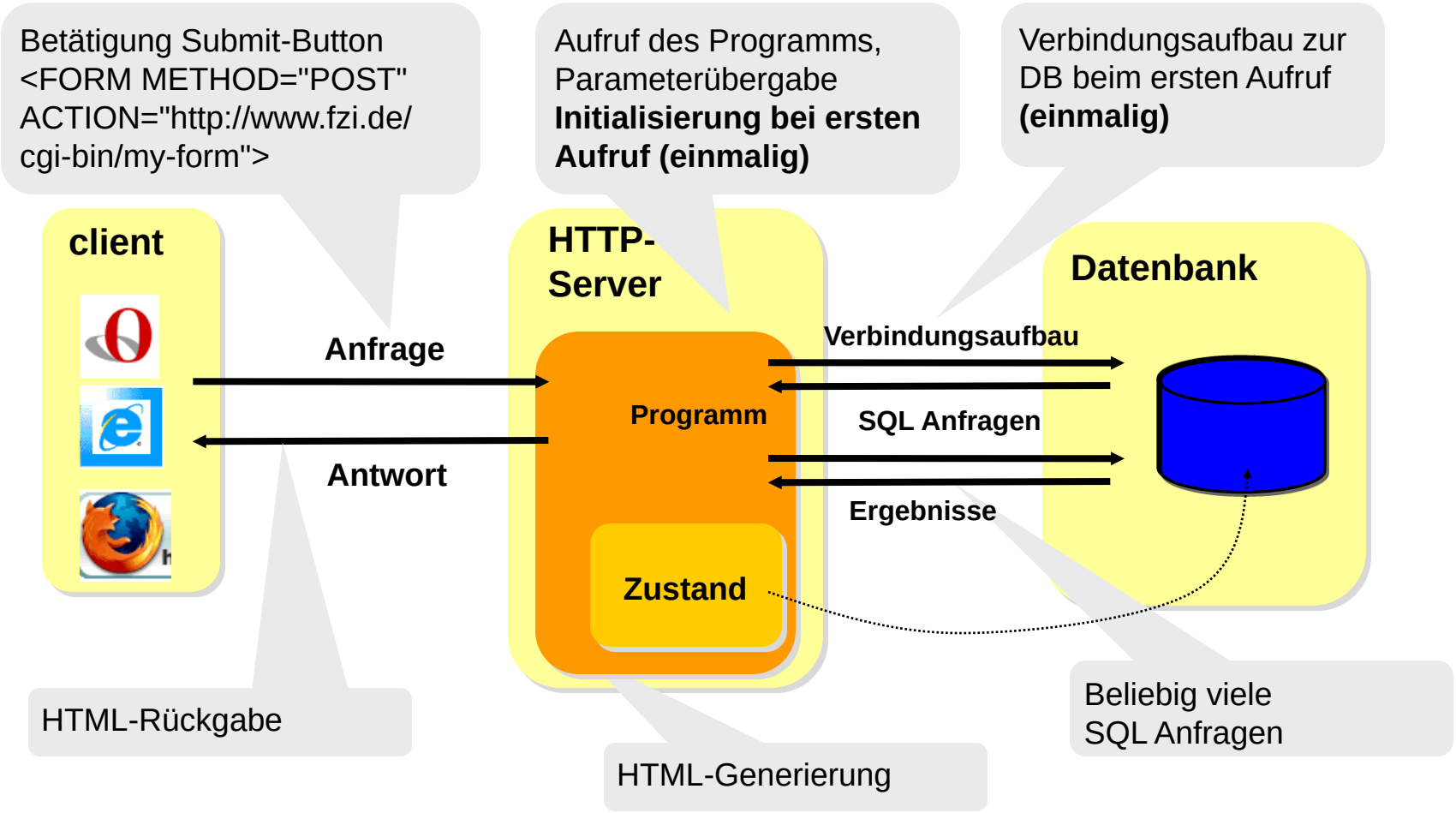
Die Old-School Variante: CGI

- Gateway an Web-Server als Erweiterung zur Integration von Skriptsprachen oder Programmen, z.B. Python, Perl

\+ Einzelne Aufrufe konkurrieren sich nicht, da jeder Aufruf in eigenem Prozess

- Ein Prozess pro Anfrage (Ressourcenintensiv)
- Keine Speicherung des Zustands (CPU intensiv)
- Für jede DB-Anfrage Verbindung aufbauen und trennen (zeitintensiv)
- Keine Trennung von Präsentation und Anwendungslogik (Wartbarkeit für größere Projekte kaum möglich)

Optimierung durch API-basierte Ansätze



Beispiel: Java Servlet

- **Programm wird in den Adressraum des Servers geladen**

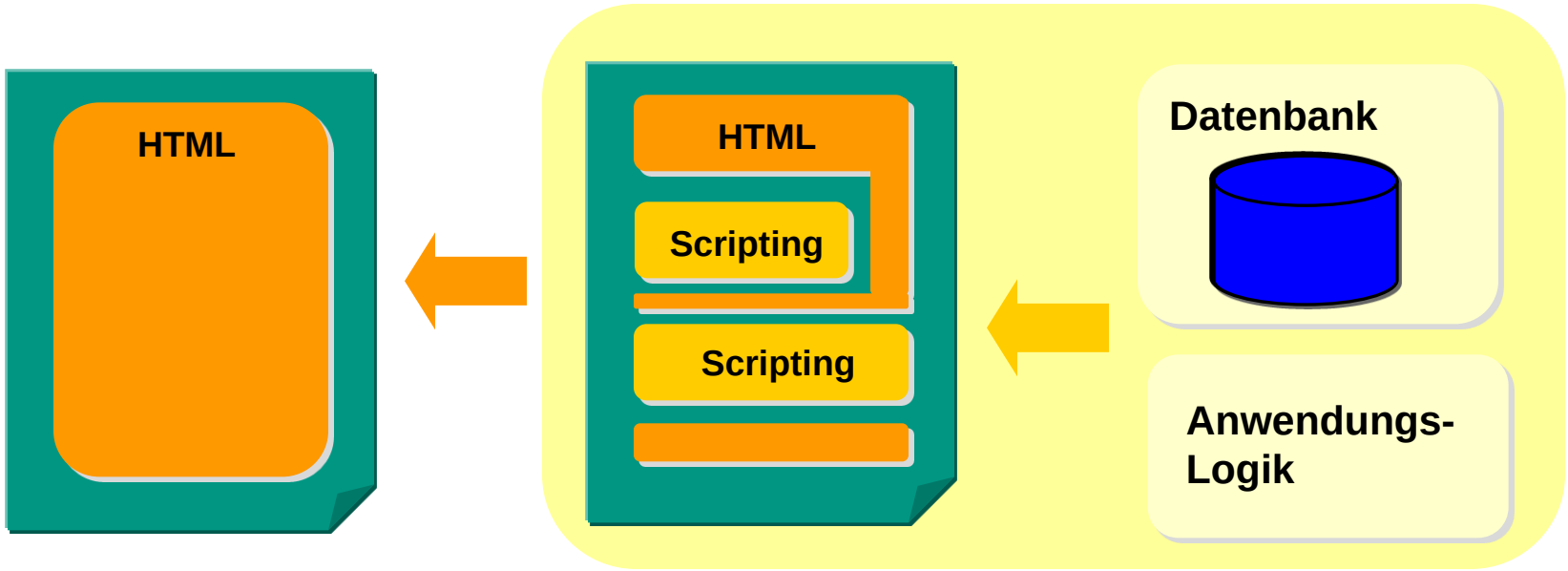
+ Programm startet einmalig, nur eine Datenbankverbindungen erforderlich
+ Zustandshaltung möglich, z.B. durch Session-IDs
+ Anfragen werden in Threads statt Prozessen ausgeführt (deutlich Ressourcenschonender)

- Installation von speziellem Web-Server zur Ausführung von Java-Servlets nötig, z.B. Apache Tomcat (für kleine Projekte ungeeignet)
- Keine Trennung von Präsentation und Anwendungslogik

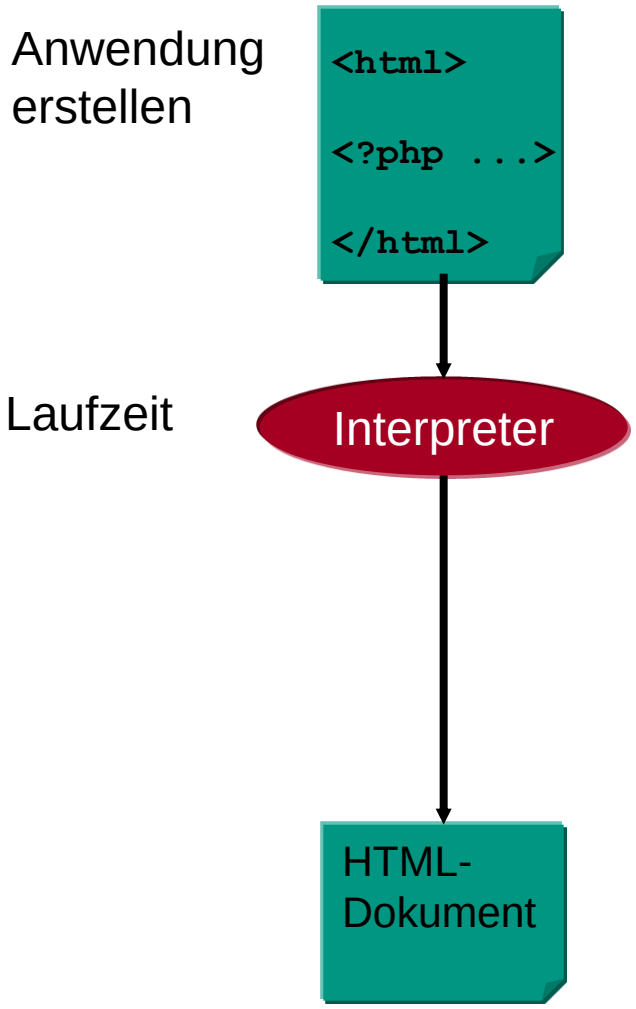
⇒ Für kleine Anwendungen zu aufwendig, für große ungeeignet

Server-Side Scripting

- In HTML-Seiten werden zusätzliche HTML-generierende Quellen integriert



Variante 1: Interpretation zur Laufzeit



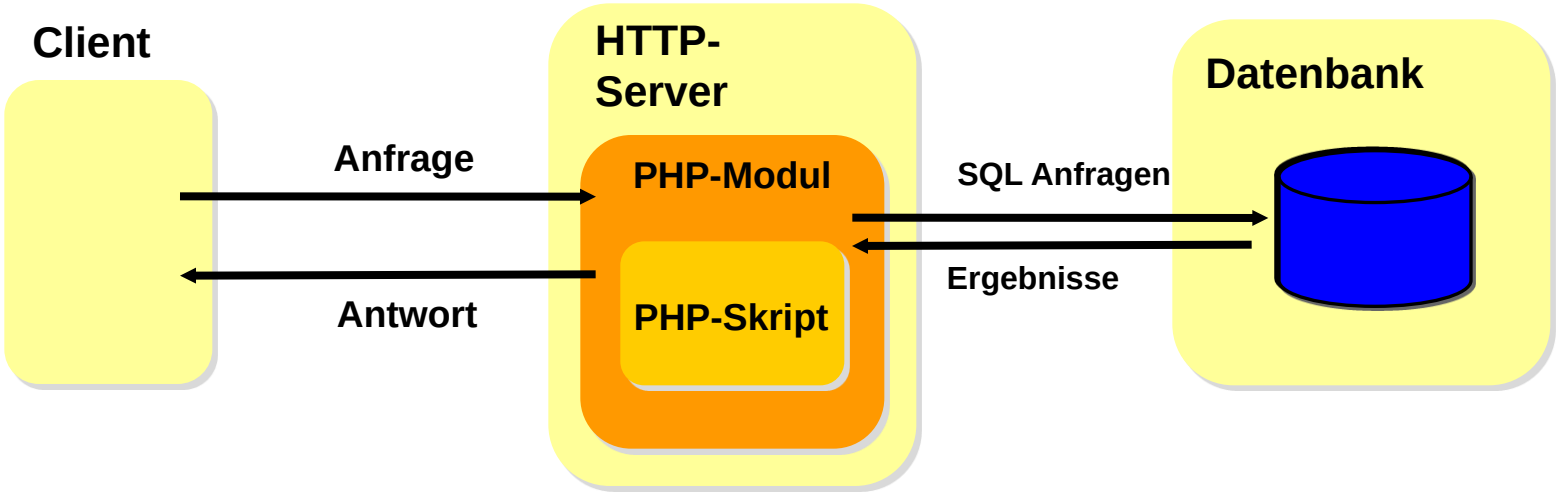
Interpretierte Skriptsprachen: z.B. PHP

- Skript wird direkt in eine HTML-Seite geschrieben
- Bei Anfrage einer Seite wird Skript durch Interpreter interpretiert und HTML-Text geschrieben
- Generiertes Dokument wird an Server zurückgegeben

\+ einfache Art, Skripte einzubinden
\+ Interpreter ist nur ein einzelner Prozess
\- CPU-Last bei jedem Seitenabruf durch Neuinterpretation
\- Keine Trennung von Präsentation und Logik

⇒ Für kleinere Webanwendungen sehr gut geeignet

PHP-Beispiel




```
<html>
<head><title>Login</title></head>
<body>
<h3>Your login data </h3>
<br/>
<table>
        <tr><td>Name:</td><td><?=$_GET[„username“]?></td></tr>
         <tr><td>Password:</td><td><?=$_GET[„pwd“]?></td></tr>
</table>
</body>
</html>
```

PHP für große Projekte: Wikipedia

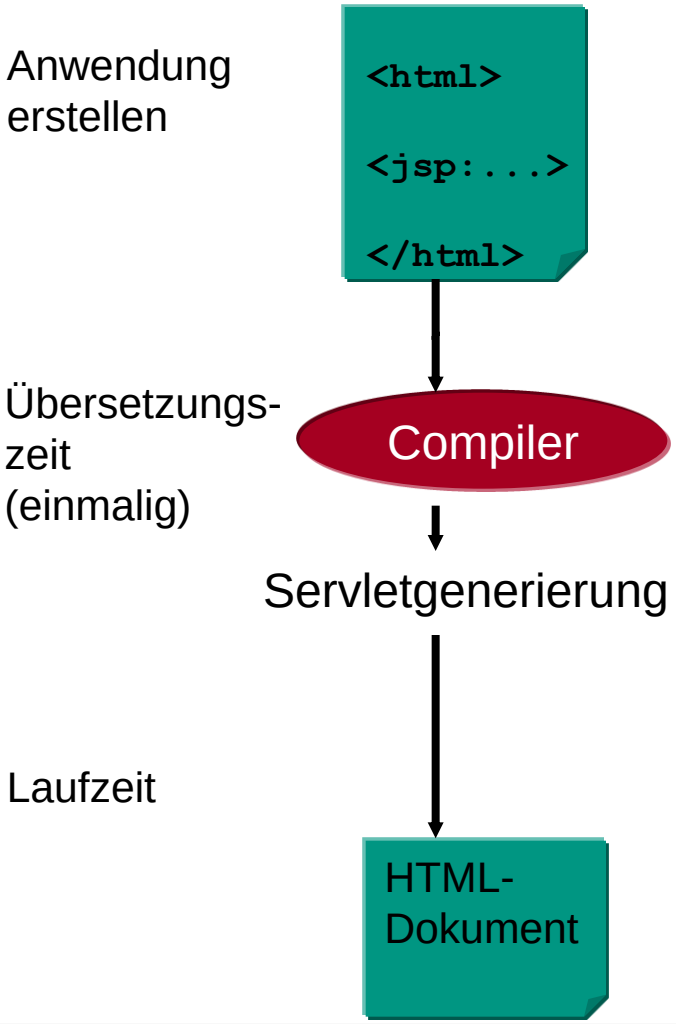
- **Praxisbeispiel**

  Wikipedia.org verwendet PHP für ein sehr umfangreiches Projekt.
- **Prinzip**: Statisches Schreiben von Webseiten
  - Verhältnis von Seitenabruf zu Seitenänderung sehr gering
  - Anwendungslogik vergleichsweise gering: Seitenänderungen annehmen und protokollieren

  ⇒ Statisches Schreiben von Webseiten reduziert CPU-/ Ressourcenlast: HTML-Seite muss nur geladen werden

! Wenn möglich, sollten aus dynamischen Seiten statische Seiten generiert werden, um die Skalierbarkeit zu erhöhen

Variante 2: Kompilierte Skriptsprachen



Kompilierte Skriptsprachen z.B. JSP

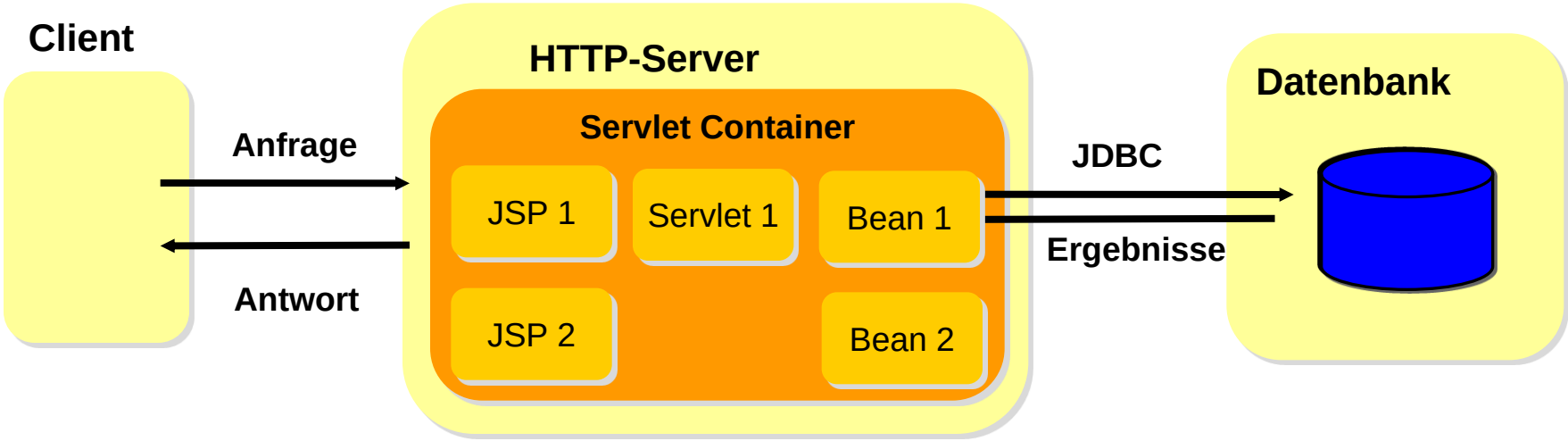
- Erfordert Installation von JAVA-fähigem Web-Server
- Kompilierung erzeugt Servlets zur Ausführung der JSP-Seiten

\+ JSP 1.1 ermöglicht durch „Tags“ klarere Trennung von Darstellung und Anwendung; noch besser mit Expression Language ab JSP 2.0

\- JSP-Programmierung entspricht nicht den meisten Programmierern bekannte GUI-Programmierung (z.B. event-Verarbeitung

Tutorial: http://www.jsptutorial.org/

Java Servlets & JSP-Beispiel



- Servlet wird kompiliert, sobald es in den Servlet-Container geladen wird
- Servlet wird beim ersten Bedarf instanziiert und initialisiert
- Servlet-Instanz wird einem Request zugeordnet.
- Nach Beendigung der Bearbeitung bleibt die Instanz für weitere Requests verfügbar.
- JSP wird zum Servlet kompiliert, sobald sie in den Servlet-Container geladen wird
- .. dann weiter wie beim Servlet

beispiel.jsp

```
<html>
  <head>
<%@ page import= “java.util.Date“%>
    <title><=request.getParameter("name");></title>
  </head>
  <body>
    <%=request.getParameter("text");%>
    <br/>
    Hello, Time is <%=new Date()%>
  </body>
</html>
```

JSP → Servlet

```
Import java.io.*; import javax.servlet.*; import javax.servlet.http.*;
public class BeispielServlet extends HttpServlet {
  public void doGet (HttpServletRequest req,  HttpServletResponse res)
      throws ServletException, IOException {
        RequestDispatcher dispatcher =
        req.getRequestDispatcher("/beispiel.jsp");
        dispatcher.forward(req, res);
}
}
```

Clientseitige Anwendungslogik

- **Motivation**
  - Die Funktionen im Browser reichen für geplante Anwendung nicht aus, z.B.
    - gleichzeitiges Aufladen von Bildern aus einem lokalen Ordner für ein Fotoalbum (Sicherheitsbeschränkungen von http erlauben immer nur eine Datei auszuwählen → Umständlich für Benutzer)
    - Abspielen von Multimedia (z.B. Videos)
  - Die Anwendung soll sehr dynamisch sein, z.B.
    - Produktwerbung (viele bewegte Bildsequenzen)
  - Die Anwendung soll wie eine Desktop-Anwendungen wirken
    - Vor ein paar Jahren gab es noch kein AJAX ;-)

Browser Plugins

Erweiterung des Browsers um neue Softwarekomponente

- am Beispiel von Adobe Flash
  - Einsatz: sehr dynamische Anwendungen, Multimedia, Desktop
  - Integrierte Skriptsprache: Action Script
- Teile oder komplette Anwendungslogik wird integriert
  - Kommunikation mit Server über http oder soap

Starten einer Plugin-Anwendung

1. Benutzer muss jeweils kompatible Version des Plugins haben
2. Eigener HTML-Tag (<object><embed>) zum Starten des Plugins in einer Webseite
3. Komplettes Programm wird zum Client übertragen
4. Plugin lädt Programm und stellt es dar

\+ Verhalten der Software exakt vorhersagbar (mit HTML problematisch)

\- Plugin, speziell aktuellste Version nicht auf allen Client PCs (z.B. beschränkte Installationsrechte auf Firmen-PSs)

\- Erfordert Spezialisten im Umgang mit der jeweiligen Programmiersprache

\- Meist Wartung schwierig

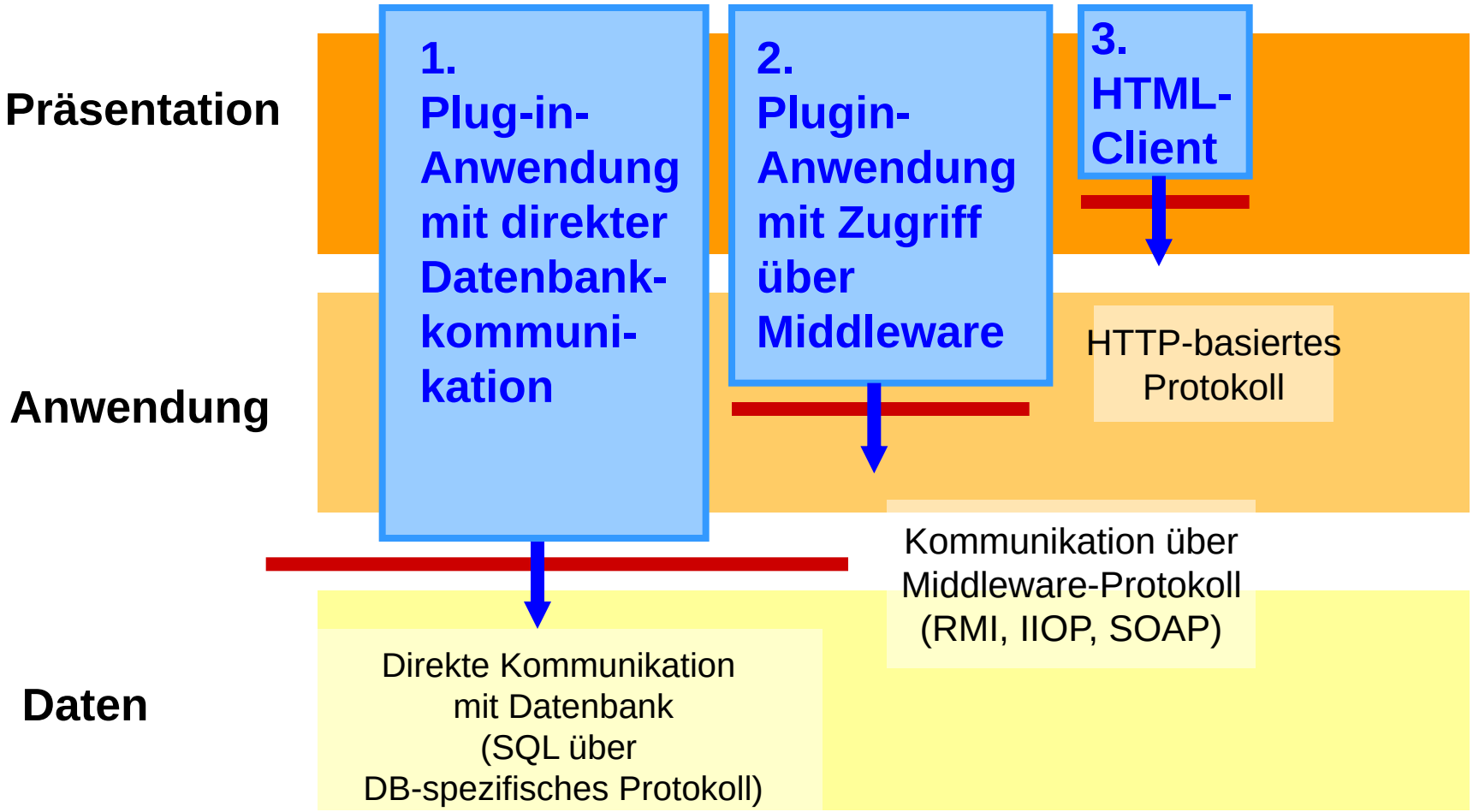
JAVA Applets

- **Motivation**
  - Über ein Browser soll eine Desktopanwendung geladen werden
  - Anwendungslogik komplett integrierbar
  - Theoretisch auch Datenhaltung direkt ansprechbar (praktisch nicht: aus Sicherheitsgründen – erfordert dies Öffnung eines Ports direkt zur Datenbank !)

+ Entwicklung der Anwendung komplett in JAVA möglich
+ Saubere Trennung von Darstellung und Anwendungslogik möglich
+ Wiederverwendbarkeit von Komponenten
- vgl. Plugins: Installationen notwendig

```
<object classid="java:example.class" codetype="application/java-vm" width=„400" height="400"></object>
```

Schichtenarchitektur mit Skripten und Plugins



Zusammenfassung: kleine Webanwendungen

- **Für kleine Anwendungen**
  - **Skalierbarkeit**, **Wartbarkeit** und **Wiederverwendbarkeit** stellen keine Anforderung dar
  - Plugins ermöglichen Spezialanwendungen
  - Alle gezeigten Skriptansätze ermöglichen die Generierung von dynamischen HTML-Seiten.
  - Erstellung von Anwendungen mit Desktop-Charakter hier nicht nötig
  - Generierung von statischen Seiten ermöglicht auch umfangreiche Anwendungen bei geringen Ressourcen

    Umsetzung der Anwendung also entscheidend von Vorkenntnissen der Programmierer und den installierten Interpretern / Gateways auf einem Web-Space

.. „Mal schnell ein Skript schreiben“ reicht da nicht mehr!

WEB-RAHMENWERKE FÜR UMFANGREICHE WEBSEITENPROJEKTE

Umfangreiche Webanwendungen

- **Beispiele:**
  - Ebay, Google (Mail), Facebook, …
- **Eigenschaften**
  - Hoher Anteil an dynamischen Seiten
  - Komplexer Seitenaufbau
  - Umfangreiche Anwendungslogik, häufige Änderungen und Erweiterungen
  - Verschiedene Ausgabesprachen, Ausgabemedien
  - Zuverlässiger Betrieb auf unterschiedlichsten Browsern
- **Erstellung mit Web-Rahmenwerk**
  - Rahmenwerk, das Erstellung einer umfangreichen Anwendung ermöglicht

Rahmenwerke über

Tapestry Echo JSF Cocoon Millstone Click

OXF Struts Cassandra WebWork SOFIA AppFuse

WebOnSwing Xoplon Groovy Flex

Genie Wicket Chiba Jaffa JBanana GWT

Smile MyFaces JWarp Facelets Melati

Eclipse RAP Japple Helma Dinamica ZK

Liste unvollständig.

Welches Rahmenwerk verwenden?

- **Hauptfrage bei einem neuen Projekt**
  - Welches Framework verwendet man?

- **Generelle Faktoren**
  - Programmiersprache: abhängig vom Team
  - Lizenz: abhängig von Budget, Infrastruktur
    (z.B. Alles in Microsoft .NET -> Lizenzen für Server, Software etc nötig)

Hauptparameter bei umfangreichen Projekten

Trennung von Anwendungslogik und Darstellung

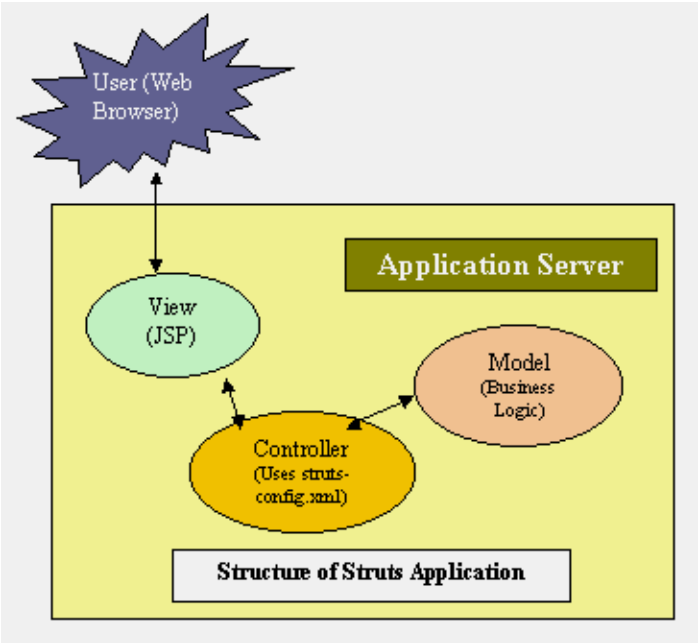
- **Gründe**
  - Code gemischt mit HTML schwer lesbar und wartbar (in Skriptsprachen und JSP der Fall)
  - Änderung der Darstellung können unabhängig von der Anwendungslogik gemacht werden.

- **Gängige Umsetzungen**
  - *Id-Marken* in HTML-Text, die später durch HTML-Text ersetzt werden – z.B.
  - Web-Rahmenwerk bietet Standardkomponenten an, die verwendet werden können (Tables, Pulldowns, etc.)
  - Formatierung durch CSS
  - Wünschenswert: eigene Komponenten erstellbar

Model View Controller

- **Dreischichtenarchitektur** trennt
  - Präsentation
  - Anwendung
  - Datenhaltung

- **Model View Controller** trennt ähnlich
  - View: Darstellung
  - Controller:
    Request / Response Verwaltung
  - Model: Daten



⇒ Saubere Trennung von Darstellung und Anwendung
⇒ Wiederverwendbarkeit der Anwendungslogik
⇒ Im Einsatz bei SWING und den meisten Web-Frameworks

Web-Rahmenwerke

- **Prinzip**
  - meist: Model - View - Controller
- **Es gibt Fülle von Rahmenwerken …**
  Welches nimmt man?
- **Ziele**
  - Zusammenstellung von Anforderungen an ein Rahmenwerk
  - Zusätzliche Anforderungen durch „AJAX-Trend“
  - GWT also Rahmenwerk für umfangreiche Anwendungen

Anforderungen an umfangreiche Webanwendungen und Web-Rahmenwerke (1)

- **Regelmäßige Umstrukturierung**, z.B. Umbau / Erweiterung der Seite:
  - *Erfordert strikte Trennung von Anwendungslogik und Darstellung*
- Unterstützung von **einheitlichem Look & Feel**
  - *Templates, auch für verschiedene Ausgabearten (PC, Tablet, Mobile, …)*
- Betrieb der Webseite in **mehreren Sprachen**
  - *Automatisches Füllen von Textstellen in Webseiten mit Ländersprache*

Mehrsprachigkeit

- Webseiten umfangreicher Projekte müssen meist in mehreren Sprachen erstellt werden
- Manuelle Erstellung von sprachspezifischen Seitenteilen ist zeit- und pflegeaufwendig, macht schwer wartbare Quelltexte

Anforderung

- Framework sollte sprachspezifische Textbausteine automatisch in Webseiten einfügen können

Gängige Umsetzung: i18n Standard

- Verwendung von Properties-Dateien für länderspezifische Sprachbausteine, z.B. (response_en.properties)
- Dynamisches Laden der jeweiligen Properties-Datei durch das Web-Framework für die Spracheinstellungen eines Nutzers

Anforderungen an umfangreiche Webanwendungen und Web-Rahmenwerke (2)

- **Hoher Anteil an dynamischen Seiten**, z.B. für Suchergebnisse, Shop-Systeme, Warenkörbe, …
  - *Erstellung von umfangreicher Anwendungslogik*
- **Umfangreiche Datenintegration** / umfangreiche Dateneingaben
  - *Erfordert Einbindung von Frameworks zur Datenintegration*
  - *Integrierte Prüfung der Konsistenz von Benutzereingaben mit Datenmodell und Anwendungslogik*

Datenintegration und Eingaben

- **Umfangreiche Datenintegration**
  - Erweiterung mit speziellen Frameworks mit Objektmapping
    - Hybernate, Spring (Details: Vorlesung am 09.12.2013)
- **Umfangreiche Dateneingaben**
  - wenn nötig Transaktionen, Speicherung, …
  - Durch Web-Rahmenwerk selbst oder in Verbindung mit Erweiterungen: Prüfung der Konsistenz von Benutzereingaben,
    - z.B. PLZ hat fünf Ziffern
    - Emailadresse muss „@“ beinhalten

Anforderungen an umfangreiche Webanwendungen und Web-Rahmenwerke (3)

- **Zugangsbeschränkungen** für geschützte Bereiche
  - *User-Sessions und Weiterleitung auf Login-Seite*
- **Schnelle Reaktionszeiten** auf Benutzeranfragen
  - *Zustandshaltung von aktuell benötigten Daten / Kontext eines Nutzers*
- Anwendungen mit **Desktopstil**
  - *Einfache Intergrierbarkeit von AJAX-Techniken*

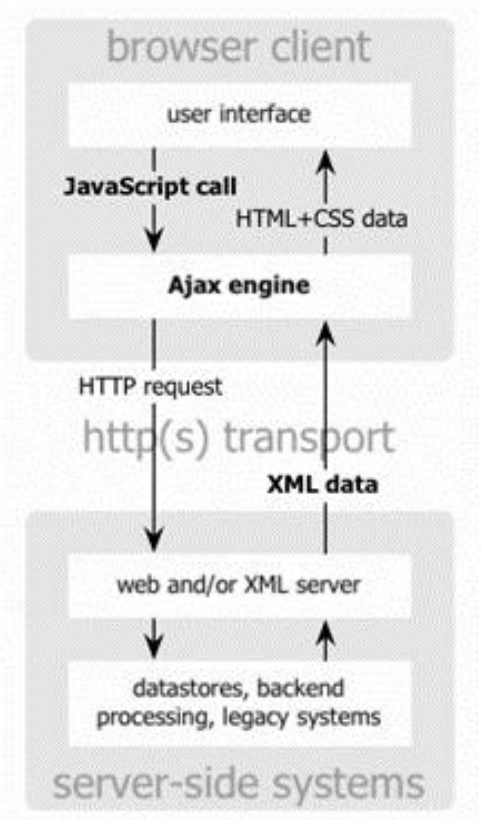
Zugangsbeschränkungen für geschützte Bereiche

- **Gängige Verfahren (da http zustandslos)**
  - Rahmenwerk generiert nach einem erfolgreichen Login ein „Session-ID String“.

    z.B. *jsessionid=0EEC1A1BE9980145F81DF6B99418B027*

  - Rahmenwerk hält ein Session-Objekt pro Benutzer zur Bindung von Objekten eines Benutzers
  - Nutzeridentifikation:
    - Session-id als cookie (falls von Browser akzeptiert), sonst als GET Parameter in allen URLs der Webseite
    - Bei jedem Seitenaufruf:
      - Rahmenwerk sucht nach vorhandenem Session-Objekt
      - Falls nicht vorhanden: Weiterleitung auf Login-Seite (falls die aufgerufene Seite eine geschützte Seite ist)

Anwendungen mit Desktopstil – AJAX



- **AJAX** (**a**synchronous **J**avaScript **a**nd **X**ML)
  - Ermöglicht die Aktualisierung von Seitenteilen
    - Somit muss nicht immer die ganze Seite neu geladen werden und die Anwendung bekommt desktop-feeling
  - Bestandteile:
    - Über JavaScript wird ein „XMLHttpRequest“ an Server geschickt
    - Server verarbeitet Request und gibt XML oder HTML zurück
    - JavaScript nimmt Request entgegen und verarbeitet diesen, meist durch Aktualisierung eines Seitenbereichs (z.B. Layers)

$\Rightarrow$ Keine wirklich neue Technologie, sondern eine Kombination von bestehenden Techniken

AJAX in der Praxis

- **Beispiele**
  - Auto-Complete für Suchanfragen
  - Pop-Up Fenster (in Seite integriert)
  - Drag & Drop
  - Direkte Fehlermeldungen auf Falscheingaben in Formularen

- **Best Practices**
  - AJAX-Funktionen manuell zu erstellen ist zeitaufwendig und fehleranfällig (Nur für JavaScript Experten)
    → **Verwendung eines geeigneten Rahmenwerks**
  - Bei Aktualisierung von Seitenteilen: Aktion dem Benutzer anzeigen (z.B. durch Ladebalken)
  - AJAX nicht wahllos einsetzen!

AJAX integriert in Web-Frameworks

- AJAX stellt umfangreiche Anforderungen an ein Web-Framework
  - Ohne AJAX: es wird eine ganze Seite generiert und Komponenten, Zustände der Seite komplett geändert
  - Mit AJAX: Webtechnologie sollte möglichst schnell und unkompliziert die Aktualisierung von Seitenteilen unterstützen
    - Komplexität der JavaScript Aufrufe verbergen
    - Methoden für AJAX-Calls anbieten
    - Aktualisierung von Seitenteilen / Komponenten ermöglichen
    - AJAX Unterstützung möglichst direkt in einem Framework (und nicht über viel „Hackerei“ eingebunden)

⇒ **Für zeitgemäße Webanwendungen:**
AJAX Support in Framework zwingend erforderlich

Anforderungen an umfangreiche Webanwendungen und Web-Rahmenwerke (4)

Weitere Anforderungen

- **Objektorientierte Programmierung** der Anwendung mit für Entwickler vertrauten Prinzipien (vgl. Swing: Datenbindung, Events, etc)

- **Schnelle Erlernbarkeit** der Webtechnologie, Abstraktion von zugrundeliegenden Techniken (z.B. Details von http)

- Durchführung von **automatischen Funktionstests** (vgl. JUnit Tests)
  Grund: manuelle Tests zeitaufwendig!

Auswahl des geeigneten Rahmenwerks

- **Zusammenfassung Hauptkriterien**
  - Erfüllung der genannten Anforderungen, Fokus auf
    - AJAX-Support
    - Erweiterbarkeit
    - **Trennung von Anwendung und Darstellung**
  - Lizenzierung: Proprietär vs. open-source
  - Programmiersprache (im Folgenden: Focus auf JAVA)
  - Zeitaufwand zur Erstellung von Web-Anwendungen
  - Support: Qualität der Mailinglisten-Antwortzeit

KONZEPTE VON WEB-RAHMENWERKEN

- Google Web Toolkit (GWT)
- Weitere Web-Rahmenwerke und Technologien

Google Web Toolkit (1)

- Programmiersprache: JAVA
- Widget Library: Verhalten von GUI Komponenten wie in Swing programmierbar (Events, Container-Zuordnung, …)
- Verhalten der Komponenten wie in Applets (mittels AJAX), d.h. möglichst alles im Client
- Proprietärer Compiler generiert daraus sämtlichen XHTML und JavaScript Code für AJAX, welcher zum Client gesendet wird
- Saubere Trennung von Darstellung und Anwendungslogik

Google Web Toolkit (2)

- Entwickler benötigen (fast) keine HTML-Kenntnisse
  (nur wenige Host-HTML Seiten werden benötigt)
- Design der Komponenten über CSS anpassbar
  (Aufruf von CSS über Host-HTML Seite)
- Spring, Hybernate für Datenhaltung integrierbar
- Für Anwendungen in der Cloud mit Google App Engine
- GWT auch nur zur GUI-Erstellung (Webseiten-Frontend) nutzbar

Google Web Toolkit Bewertung

+ **Trennung von Anwendung und Darstellung**
+ **Einfach erlernbar und schnell anwendbar**
+ Stetig wachsende Community – guter Support

- Getrennte Betrachtung von Client und Serverseite (Bereitstellung von Daten durch SOAP basierte RPC Calls)
- Proprietärer Compiler
- Generiertes JavaScript ist kryptisch (proprietär), Debugging nur in Java

- Tutorial: https://developers.google.com/web-toolkit/gettingstarted

Weitere Web-Rahmenwerke und Technologien

- **Apache Wicket**
  - Offizielles Apache Projekt, open source
  - Performante Event-getriebene Architektur
  - Strikte Trennung von Anwendungsentwicklung mit JAVA Objekten und Seitendesign mit HTML
    - Integration von Komponenten mit HTML Tags (<wicket:id=“..“ .. >)
  - Umfangreicher AJAX-Support
    - Allerdings AJAX-Calls seriell
  - Spring, Hybernate für Datenhaltung integrierbar
  - Keine Widget Library; GUI-Komponenten müssen selbst erstellt werden
  - Hohe Serverlast – jeder Klick geht zum Server

Weitere Web-Rahmenwerke und Technologien

(kleine Auswahl)

- **Eclipse Rich Application Client** (RAP)
  - Entwicklung von „richtigen“ Desktop-Anwendungen mit Eclipse
  - Interessant für schnelle Portierung von Desktop-Anwendungen ins Web
  - Tutorial: http://www.vogella.com/articles/EclipseRAP/article.html
- **Alles nur mit XML** / XSLT / … (eher nur noch im Backend)

Zusammenfassung

- Umfangreiche Webanwendungen erfordern Web-Rahmenwerke
  - Ermöglichen langfristige Wartung einer Anwendung
  - Trennung von Darstellung und Anwendung wichtig
  - Verdecken Komplexität der zugrunde liegenden Webtechnologien
    - Umfangreiche Anforderungen an ein Web-Rahmenwerk
  - Integration von AJAX in ein Rahmenwerk ermöglicht die Erstellung von Anwendungen mit Desktop-Charakter
- Auswahl des „richtigen“ Rahmenwerks
  - Bei neuen Projekten: Prüfen, ob aktuelle Rahmenwerke die Anforderungen besser/direkter/schneller erfüllen können

Literatur

(eine Auswahl hier, Vielzahl an Tutorien im Web verfügbar)

- **Buch: Web-Technologien**
  - von Heiko Wöhr von Dpunkt Verlag (Taschenbuch - Mai 2004)
- **Buch: Web-Technologien: Grundlagen, Web-Programmierung, Suchmaschinen, Semantic Web**
  - Christoph Meinel, Harald Sack von x.media.press (2013)
- **JSP**
  - http://www.jsptutorial.org/
- **AJAX**
  - http://www.w3schools.com/ajax/default.asp
- **Google Web Toolkit**
  - http://code.google.com/webtoolkit/
- **Eclipse RAP**
  - http://www.eclipse.org/rap/