

Angewandte Informatik 2 - Tutorium 5

Besprechung: Übungsblatt 5

Götz Bürkle
(goetz@buerkle.org)

Institut für Angewandte Informatik und
Formale Beschreibungsverfahren - AIFB
Universität Karlsruhe (TH)

KW 27/28

Organisatorischer Hinweis

Fragestunde am 16.07.2007

Fragen bitte an goetz@buerkle.org

Übersicht

Organisatorisches

Heimarbeitsblatt 5

Aufgabe 1 - HTTP

Aufgabe 2 - Webanwendungen

Aufgabe 3 - PHP

Aufgabe 4 - Webservices

Zusammenfassung

GET ⇔ POST

- ▶ Beide Methoden GET und POST erläutern
- ▶ Sind beide gleichermaßen brauchbar?
 - Begründen!

GET-Methode

- ▶ bedient sich des URL-Encodings
- ▶ Daten und Parameter werden im Klartext übertragen
 - ⇒ können im Browser oder sonstwo abgelesen werden
 - ⇒ können sehr einfach manipuliert werden
- ▶ Längenbeschränkung der zu übertragenden Strings, siehe RFC 2068 (HTTP)

GET-Methode

praktische Längenbeschränkung bei GET

- ▶ man muß die maximale URL-Länge des jeweils verarbeitenden Servers kennen, um in seinem CGI-Skript den Maximalwert anzupassen und zu limitieren
- ▶ sicherlich keine gute und empfehlenswerte Strategie
- ▶ Sinnvoll wenn:
 - ▶ sehr wenige Daten verarbeitet werden sollen
 - ▶ die Daten sich sehr leicht überprüfen lassen

GET-Methode

Längenbeschränkung nach RFC 2068 (HTTP)

The HTTP protocol does not place any a priori limit on the length of a URI.

Servers MUST be able to handle the URI of any resource they serve, and SHOULD be able to handle URIs of unbounded length if they provide GET-based forms that could generate such URIs. A server SHOULD return 414 (Request-URI Too Long) status if a URI is longer than the server can handle (see section 10.4.15).

Note: Servers should be cautious about depending on URI lengths above 255 bytes, because some older client or proxy implementations may not properly support these lengths.

POST-Methode

- ▶ zu übermittelnde Daten werden nach dem HTTP-Header übertragen und über STDIN (standard input stream) an das CGI-Skript übergeben
- ▶ Manipulation der Daten sehr viel schwerer und aufwendiger

GET ⇔ POST

- ▶ generell bei CGI-Programmierung POST-Methode vorzuziehen
 - Gegen GET sprechen:
 - ▶ Probleme der Längenbeschränkung
 - ▶ erhöhtes Sicherheitsrisiko
- ▶ Überprüfung der übergebenen Daten ist in beiden Fällen durchzuführen

siehe auch <http://de.selfhtml.org/servercgi/cgi/formularverarbeitung.htm>

Technologien - JavaScript

- ▶ clientseitige Scriptsprache, in HTML-Seiten eingebettet oder als externe Datei
- ▶ hat (trotz des Namens) mit Java nicht viel gemeinsam
- ▶ zur Gestaltung clientseitig dynamischer und interaktiver Webseiten
 - ⇒ Änderung des HTML-Codes zur Laufzeit im Browser (z.B. um Menüs einzublenden, heutzutage bedeutende Rolle im Zusammenhang mit „AJAX“
 - Nachladen von Daten, ohne komplette Seite neuzuladen)
- ▶ Sprachumfang begrenzt
- ▶ Einsatz mit gewissen Unsicherheiten verbunden (Interpreter kann abgeschaltet werden)

Ticketreservierungssystem

Stellen Sie sich vor, Sie erhalten den Auftrag, für die Karlsruher Schauburg ein **internetfähiges Ticketreservierungssystem** zu konzipieren und später auch zu implementieren. In Ihrer Projektskizze haben Sie dem Kunden dargelegt, dass Sie in der Praxis **bewährte und standardisierte Technologien wie Javascript und Servlets/JSP einsetzen** wollen.

Machen Sie Vorschläge, wie Sie beispielsweise die E-Mail-Adresse, die Kundennummer und Kontoverbindungen der potenziellen Kunden **auf Korrektheit überprüfen** können und nennen Sie **Vor- und Nachteile**, die jeweils für den Einsatz von Javascript oder JSP sprechen bzw. eine Kombination beider Techniken sinnvoll erscheinen lassen.

Technologien - Java-Servlets/JSP

- ▶ Servertechnologie
- ▶ dient der dynamischen Erzeugung von HTML-Seiten zur Laufzeit auf dem Server
- ▶ eine Java-Klasse erzeugt den kompletten HTML-Code
- ▶ die komplette Java-Infrastruktur steht zur Verfügung
- ▶ JSP sind Servlets sehr ähnlich, nur dass der Code in HTML eingebettet ist
 - und vom Server intern in ein Servlet umgeformt wird.
- ▶ JSP-Technologie etwas neuer als „klassische“ Servlets.

JavaScript - clientseitige Prüfung

Vorteile

- ▶ Benutzer erhält schnell Feedback bei falscher Eingabe
- ▶ Entlastung des Servers beim Erkennen von (un-)beabsichtigten Fehleingaben

Nachteile

- ▶ Script muss auf möglichst vielen Plattformen laufen
- ▶ bei deaktiviertem JavaScript werden Daten ungeprüft übermittelt
- ▶ bestimmte Prüfungen clientseitig gar nicht möglich (z. B. Zugriff auf eine Datenbank zur Überprüfung, ob Benutzer bereits existiert)
- ▶ Umschreiben der Scripte durch den Benutzer möglich
⇒ Prüfung der Daten kann umgangen werden

Kombination der Techniken

- ▶ Formulardaten durch ein Applet oder durch JavaScript vorverarbeiten
⇒ schnelleres Feedback bei Fehleingaben
- ▶ erkannt werden können:
 - ▶ ungewünschte Eingaben
 - ▶ ungewollte Schreib- und Tippfehler oder Ähnliches (z.B. Buchstabe, wo Zahl erwartet wird)⇒ Vermeidung sinnloser Einträge in Datenbanken
- ▶ Daten werden anschließend durch das Servlet auf dem Server geprüft, ohne dass der Benutzer dies umgehen kann (z.B. mit Hilfe regulärer Ausdrücke, um Angriffe auf Datenbank zu verhindern)

JSP - serverseitige Prüfung

Vorteile

- ▶ bessere Plausibilitätsprüfung (z.B. durch Zugriffe auf eine Datenbank)
- ▶ Prüfung kann nicht ohne extremen Aufwand umgangen werden

Nachteile

- ▶ Bedarf an Ressourcen auf dem Server
- ▶ Latenzzeit bis zum Feedback an Benutzer vergrößert sich (Webserver sitzt in der Mitte zwischen Client und Skript)
- ▶ bei fehlerhafter Implementierung wird der Server direkt angreifbar

Kombination der Techniken

Auch wenn eine Vorverarbeitung von Formulardaten durch JavaScript oder Ähnliches bereits beim Client stattfindet, so muss eine serverseitige Prüfung dennoch so gewissenhaft wie möglich durchgeführt werden!

Man sollte sich also trotz Vorprüfung niemals auf bestimmte Dinge verlassen.

Niemals Benutzereingaben ungeprüft weiterverarbeiten!

Vorschläge: E-Mail-Adresse

- ▶ Syntax lässt sich clientseitig überprüfen, wenn JavaScript aktiviert
- ▶ serverseitig kann überprüft werden, ob Domain der E-Mail-Adresse existiert (bei Subdomains teilweise nicht überprüfbar)
Schönes Beispielskript:
PHP Email address validation with Verify probe
<http://www.tienhuis.nl/php-email-address-validation-with-verify-probe>
- ▶ Teil vor dem @-Zeichen lässt sich nie restlos verifizieren

Vorschläge: Kontoverbindung

- ▶ clientseitige Vorverarbeitung mittels JavaScript (nur Ziffern, keine Buchstaben und Sonderzeichen, Mindestlängen, ...)
- ▶ serverseitige Überprüfung durch entsprechende Datenbankabfragen (sensible Daten erfordern erhöhte Sicherheitsvorkehrungen ⇒ Verschlüsselung (SSL/TLS)!)

Vorschläge: Kundennummer

- ▶ basiert Kundennummer ähnlich wie die Matrikelnummer auf einem „Baukastenprinzip“, kann die Syntax mit JavaScript überprüft werden
Beispielhafter Algorithmus:
 1. bilde Quersumme der Kundennummer
 2. falls Quersumme durch 7 teilbar
⇒ Kundennummer gültig
- ▶ Ob Kundennummer wirklich existiert, lässt sich natürlich nur im Zusammenhang mit einer Datenbankabfrage beantworten (JSP), aber Tippfehler und damit verbundene unnötige Datenbankabfragen können so vermieden werden

Prüfungsanmeldesystem

Zu schreiben ist eine Kombination aus HTML-Seite und PHP-MySQL-Datei für ein **elektronisches Prüfungsanmeldesystem am Institut AIFB**.

(Jeder, der sich selbst schonmal bei einer AIFB-Klausur angemeldet hat, kennt das System vom Prinzip her - das kann man im Hinterkopf haben)

Prüfungsanmeldesystem - Funktionsumfang

- ▶ HTML-Maske, über die die Vorlesung auswählbar ist (Select-Box) und Daten bestehend aus Name, Vorname, Matrikelnummer und E-Mail-Adresse eingegeben werden können.
- ▶ PHP-Skript soll prüfen, ob zu Matrikelnummer und gewählter Veranstaltung schon eine Anmeldung vorliegt.
 - ▶ Ist dies der Fall, soll auf dem Bildschirm ausgegeben werden, dass eine Anmeldung schon erfolgte.
 - ▶ Falls noch keine Anmeldung vorliegt, soll ein entsprechender Eintrag in die Tabelle tbl_anmeldung geschrieben werden.
Außerdem erfolgt eine positive Ausgabe am Bildschirm.
- ▶ Zusätzlich zur positiven Bildschirmausgabe schickt das System eine Bestätigungsemail über die erfolgreiche Anmeldung.

klausuranmeldung.html, head

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <title>Klausuranmeldung Institut AIFB</title>
    <style type="text/css">
    <!--
      input, select { width:300px; }
      table { border: 0; border-spacing: 10px; }
      td { padding: 0; }
      tr { text-align: left; vertical-align: top; }
    -->
    </style>
  </head>

  [...]

</html>
```

Vorgehensweise

- ▶ HTML-Formular schreiben
- ▶ „richtige“ Methode für Formular wählen
- ▶ Welche Daten müssen erhoben werden?
- ▶ Welche Daten sollen wo gespeichert werden?
- ▶ Wie sollen die Daten überprüft werden?
- ▶ Was soll wie ausgegeben werden?

⇒ Programmablaufplan aufstellen

siehe <http://de.wikipedia.org/wiki/Programmablaufplan>

klausuranmeldung.html, body, 1

```
<!DOCTYPE [...]
<html>
  [...]
  <body>
    <h1>Klausuranmeldung Institut AIFB</h1>
    <form name="formular" action="undaction.php" method="post">
      <table>
        <tr>
          <td width="150">Vorlesung</td>
          <td>
            <select name="vorlesung" size="1">
              <option value="GdInfoI">Grundlagen der Informatik I</option>
              <option value="GdInfoII">Grundlagen der Informatik II</option>
              <option value="AI1">Angewandte Informatik 1</option>
              <option value="AI2">Angewandte Informatik 2</option>
            </select>
          </td>
        </tr>
        [...]
      </table>
    </form>
  </body>
</html>
```

klausuranmeldung.html, body, 2

```
<!DOCTYPE [...]
<html>
  [...]
  <body>
    [...]
    <form name="formular" action="undaction.php" method="post">
      <table>
        [...]
        <tr>
          <td>Name</td>
          <td><input type="text" name="name"></td>
        </tr><tr>
          <td>Vorname</td>
          <td><input type="text" name="vorname"></td>
        </tr><tr>
          <td>Matrikelnummer</td>
          <td><input type="text" name="matrikel"></td>
        </tr><tr>
          <td>E-Mail-Adresse</td>
          <td><input type="text" name="email"></td>
        </tr><tr>
          <td>&nbsp;</td>
          <td><input type="Submit" value="Senden"></td>
        </tr>
      </table>
    </form>
  </body>
</html>
```

undaction.php, 2

```
[...]
// SQL-Befehl, ohne vorher die Benutzereingaben zu prüfen!
$sql_1 = 'SELECT db_matrikel FROM tbl_anmeldung
WHERE db_matrikel = '.$matrikel.'
AND db_vorlesung = '.$vorlesung.'';
// debug: echo('<!- - '.$sql_1.' - ->');
// SQL abschicken
$result_1 = mysql_query($sql_1);
if ($result_1 !== FALSE) {
  // falls Auslesen erfolgreich
  $nr_1 = mysql_num_rows($result_1);
  if ($nr_1 == 0) { // < 1
    // es liegt noch keine Anmeldung vor
    // neuen Datensatz in tbl_anmeldung schreiben
    // SQL-Befehl
    $sql_2 = 'INSERT INTO tbl_anmeldung
(db_vorlesung, db_name, db_vorname, db_matrikel, db_email) VALUES
('.$vorlesung.', '.$name.', '.$vorname.', '.$matrikel.', '.$email.'');';
    // debug: echo('<!- - '.$sql_2.' - ->');
    // SQL abschicken
    $result_2 = mysql_query($sql_2) or die ('Fehler in sql_2');
  }
  [...]
```

undaction.php, 1

```
<?php
// MySQL-Server ansprechen und Datenbank oeffnen
// Verbindungsdaten
$dbhost = '***'; // Hostname des Servers
$dbuser = '***'; // MySQL-Benutzername
$dbpass = '***'; // MySQL-Kennwort
$dbname = '***'; // Datenbank
// Verbindung herstellen
mysql_connect($dbhost, $dbuser, $dbpass)
  or die ('Verbindung zum SQL-Server fehlgeschlagen.');
```

```
// Datenbank auswahlen
mysql_select_db($dbname)
  or die ('Datenbank kann nicht angesprochen werden.');
```

```
// Variablen initialisieren
$matrikel = $_POST['matrikel'];
$vorlesung = $_POST['vorlesung'];
$name = $_POST['name'];
$vorname = $_POST['vorname'];
$email = $_POST['email'];

$output = ' ';
[...]
```

undaction.php, 3

```
[...]
if ($result_2 !== FALSE) {
  // Bildschirmausgabe
  $output .= 'Ihre Anmeldung war erfolgreich.<br>
  Eine Bestätigung per E-Mail erfolgt in Kuerze.';
  // Bestaetigungsemail verschicken
  $betreff = 'Bestaetigung Klausuranmeldung Institut AIFB';
  $mailto = "$vorname $name <$email>";
  $header = "From: Institut AIFB <no-reply@aifb.uni-karlsruhe.de>
X-Mailer: Sismail Web Email Interface
MIME-version: 1.0
Content-Type: multipart/alternative;";
  $text = "
  Vorlesung: $vorlesung \n
  Name: $name \n
  Vorname: $vorname \n
  Matrikelnummer: $matrikel \n
  E-Mail: $email \n
  ";
  mail($mailto, $betreff, $text, $header);
}
[...]
```

undaction.php, 4

```
[...]
} else if ($nr_1 == 1) { // > 0
    // es liegt eine Anmeldung vor
    $output .= 'Eine Anmeldung ist bereits erfolgt.';
} else {
    $output .= 'Dieser Fall existiert nicht.';
}
} else {
    $output .= 'Fehler in sql_1: Die Datenbank kann nicht angesprochen werden.';
}
?>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head><title>Klausuranmeldung Institut AIFB</title></head>
<body>
    <?php echo($output); ?>
</body>
</html>
```

WSDL

► Was heißt WSDL?

Web Services Description Language

► Was ist WSDL?

- „plattform-, programmiersprachen- und protokollunabhängige XML-Spezifikation zur Beschreibung von Netzwerkdiensten (Web Services) zum Austausch von Nachrichten“
- „WSDL wird häufig in Kombination mit SOAP und dem XML-Schema verwendet, um Web Services im Internet anzubieten. Ein Client, der einen Webservice aufruft, kann WSDL lesen, um zu bestimmen, welche Funktionen auf dem Server verfügbar sind. Alle verwendeten speziellen Datentypen sind in der WSDL-Datei in XML-Form eingebunden. Der Client kann nun SOAP verwenden, um eine in WSDL gelistete Funktion letztlich aufzurufen.“

siehe auch <http://de.wikipedia.org/wiki/WSDL>

WSDL-Beschreibung, 1

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:s1="http://microsoft.com/wsdl/types/"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:s="http://www.w3.org/2001/XMLSchema"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:tns="http://aifb.de/ai2/"
    xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    targetNamespace="http://aifb.de/ai2/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
<wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://aifb.de/ai2/">
        <s:import namespace="http://microsoft.com/wsdl/types/" />
        <s:element name="GetPositionOfSymbolInText">
            <s:complexType>
                <s:sequence>
                    <s:element minOccurs="0" maxOccurs="1" name="text" type="s:string"/>
                    <s:element minOccurs="1" maxOccurs="1" name="symbol" type="s1:char"/>
                </s:sequence>
            </s:complexType>
        </s:element>
        [...]
    </wsdl:types>
```

WSDL-Beschreibung, 2

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions [...]
    [...]
    <s:element name="GetPositionOfSymbolInTextResponse">
        <s:complexType>
            <s:sequence>
                <s:element minOccurs="0" maxOccurs="1"
                    name="GetPositionOfSymbolInTextResult" type="tns:ArrayOfInt"/>
            </s:sequence>
        </s:complexType>
    </s:element>
    <s:complexType name="ArrayOfInt">
        <s:sequence>
            <s:element minOccurs="0" maxOccurs="unbounded" name="int" type="s:int"/>
        </s:sequence>
    </s:complexType>
</s:schema>
<s:schema elementFormDefault="qualified"
    targetNamespace="http://microsoft.com/wsdl/types/">
    <s:simpleType name="char">
        <s:restriction base="s:unsignedShort"/>
    </s:simpleType>
</s:schema>
</wsdl:types>

[...]
```

WSDL-Beschreibung, 3

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions [...]
 [...]

<wsdl:message name="GetPositionsOfSymbolInTextSoapIn">
  <wsdl:part name="parameters" element="tns:GetPositionsOfSymbolInText"/>
</wsdl:message>
<wsdl:message name="GetPositionsOfSymbolInTextSoapOut">
  <wsdl:part name="parameters" element="tns:GetPositionsOfSymbolInTextResponse"/>
</wsdl:message>
<wsdl:portType name="AI2-UebungsServiceSoap">
  <wsdl:operation name="GetPositionsOfSymbolInText">
    <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">
      Liefert Positionen eines Zeichens in einem Text</documentation>
    <wsdl:input message="tns:GetPositionsOfSymbolInTextSoapIn"/>
    <wsdl:output message="tns:GetPositionsOfSymbolInTextSoapOut"/>
  </wsdl:operation>
</wsdl:portType>
[... ]
</wsdl:definitions>
```

Aufgabe

Signatur der beschriebenen Methode(n) in der typischen Java/C#/C++ Notation angeben:

```
ReturnType MethodName
([ParamType ParamName
 [, ParamType ParamName]*])
```

Angaben, welche Daten von einem Client zum Server und wieder zurück übertragen werden, wenn als Übertragungsprotokoll SOAP/HTTP zum Einsatz kommt (TCP/IP-Protokolldaten können selbstverständlich weggelassen werden!).

WSDL-Beschreibung, 4

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions [...]
 [...]
<wsdl:binding name="AI2-UebungsServiceSoap" type="tns:AI2-UebungsServiceSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
  <wsdl:operation name="GetPositionsOfSymbolInText">
    <soap:operation soapAction="http://aifb.de/ai2/GetPositionsOfSymbolInText"
      style="document"/>
    <wsdl:input>
      <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>

<wsdl:service name="AI2-UebungsService">
  <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">
    Dieser Dienst dient dazu, den Studis der Vorlesung AI2
    die Protokolle WSDL und SOAP näherzubringen
  </documentation>
  <wsdl:port name="AI2-UebungsServiceSoap" binding="tns:AI2-UebungsServiceSoap">
    <soap:address location="http://ws-server.de/AI2Service/UebungsService.asmx"/>
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

Was konkret zu tun ist

Dienst mit folgenden Daten aufrufen:

- ▶ „Angewandte Informatik macht Spass!“ und
- ▶ „a“

(Annahme: Der Dienst funktioniert korrekt.)

Funktionsaufruf

Der gesuchte Funktionsaufruf hat folgende Signatur:

```
public int[] GetPositionsOfSymbolInText  
(string text, char symbol)
```

In unserem Fall also:

```
GetPositionsOfSymbolInText  
(\"Angewandte Informatik macht Spass!\",  
 \"a\")
```

Request

```
POST /AI2Service/UebungsService.asmx HTTP/1.1  
Host: ws-server.de  
Content-Type: text/xml; charset=utf-8  
Content-Length: 423  
User-Agent: some great client  
SOAPAction: \"http://aifb.de/ai2/GetPositionsOfSymbolInText\"  
  
<?xml version=\"1.0\" encoding=\"utf-8\"?>  
<soap:Envelope  
  xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"  
  xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\"  
  xmlns:soap=\"http://schemas.xmlsoap.org/soap/envelope/\">  
  <soap:Body>  
    <GetPositionsOfSymbolInText xmlns=\"http://aifb.de/ai2/\">  
      <text>Angewandte Informatik macht Spass!</text>  
      <symbol>a</symbol>  
    </GetPositionsOfSymbolInText>  
  </soap:Body>  
</soap:Envelope>
```

Funktionsaufruf bei Webservices

```
GetPositionsOfSymbolInText  
(\"Angewandte Informatik macht Spass!\",  
 \"a\")
```

Wie spricht man nun den Webservice konkret an?

→ Funktionsaufruf „in XML verpacken“

Response

```
HTTP/1.1 200 OK  
Content-Type: text/xml; charset=utf-8  
Content-Length: 519  
  
<?xml version=\"1.0\" encoding=\"utf-8\"?>  
<soap:Envelope  
  xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"  
  xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\"  
  xmlns:soap=\"http://schemas.xmlsoap.org/soap/envelope/\">  
  <soap:Body>  
    <GetPositionsOfSymbolInTextResponse  
      xmlns=\"http://aifb.de/ai2/\">  
      <GetPositionsOfSymbolInTextResult>  
        <int>5</int>  
        <int>17</int>  
        <int>23</int>  
        <int>30</int>  
      </GetPositionsOfSymbolInTextResult>  
    </GetPositionsOfSymbolInTextResponse>  
  </soap:Body>  
</soap:Envelope>
```

Zusammenfassung

- ▶ **Webanwendungen:** Grundlagen der Formularverarbeitung erklären können
Warum ist Überprüfung von Benutzereingaben wichtig?
Technologien zur Überprüfung kennen und erklären können
- ▶ **Sicherheit:** Einfache Fragen zum Thema Sicherheit beantworten können
- ▶ **Information Retrieval:** Recall und Precision erklären und berechnen können

Zusammenfassung

- ▶ **HTTP:** Die Methoden GET und POST gegeneinander abgrenzen können
Vorteile und Nachteile erklären können
- ▶ **PHP:** praktische Umsetzung eines formularverarbeitenden Skriptes
Wichtig ist die Idee, wie man so ein System implementiert
- ▶ **Webservices:** WSDL-Datei lesen und verstehen können