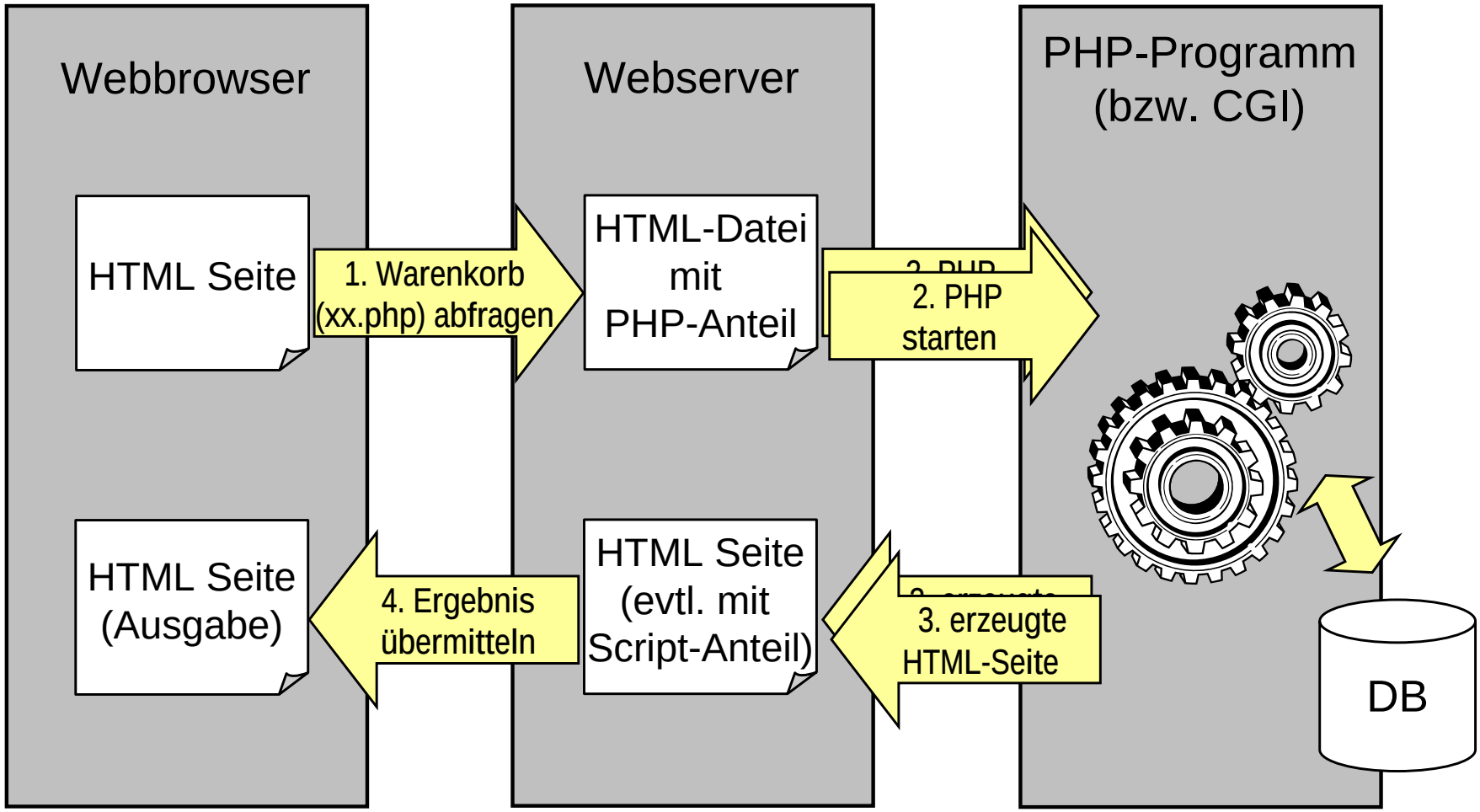
Entwicklung webbasierter Anwendungen

11. Kapitel: Technologien im Zusammenhang

Quellenhinweis:
Viele Folien dieser Vorlesung entstammen der gleichnamigen Vorlesung von Prof. B. Kreling

Prinzip


Webbrowser
Webserver
PHP-Programm
(bzw. CGI)
HTML Seite
1. Warenkorb
(xx.php) abfragen
HTML-Datei
mit
PHP-Anteil
2. PHP
starten
HTML Seite
(Ausgabe)
4. Ergebnis
übermitteln
HTML Seite
(evtl. mit
Script-Anteil)
3. erzeugte
HTML-Seite
DB

Beispiel

Beispiel: Warenkorb (bei BOL)

Mit PHP erzeugte HTML-Seite

Eintrag aus einer Datenbank

erste Überprüfung der Eingabe mit Javascript

Layout mit CSS

Entwicklung webbasierter Anwendungen

12. Kapitel: Sessionverwaltung

Quellenhinweis:
Viele Folien dieser Vorlesung entstammen der gleichnamigen Vorlesung von Prof. B. Kreling

Zusammenhängende Webseite - Beispiel

- Formular-Folge mit mehreren Seiten (z.B. Shop, Warenkorb, Bestellung,…)

⇨ Woher weiß der Webserver / ein CGI-Skript, dass Aufrufe zusammen gehören?

⇨ Was passiert, wenn die URL parallel in mehreren Browsern geöffnet wird?

Die Problematik I

Dieses Problem wird in der Vorlesung nicht weiter behandelt.
(⇨ Semaphor und Shared Memory Funktionen in PHP)

- mehrere Aufrufe desselben Skripts können gleichzeitig ausgeführt werden (quasi-parallel, nebenläufig)

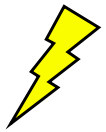
  Sequenz (im Prinzip):

  Client1: $no=get_no_of_entries()

  Client2: delete entry[0]

  Client1: for (i=0 to $no-1) entry[i]=…

- Lösung: Skript muss reentrant sein
  - ⇨ Verwendung einer Datenbank-Tabelle, die Transaktionen unterstützt (z.B. InnoDB oder BerkeleyDB, aber nicht MyISAM)
  - ⇨ Zugriff auf Dateien/Datenbanken muss als kritischer Abschnitt / Transaktion gesichert sein

Die Problematik II

- jeder Aufruf eines CGI-Skripts ist ein neuer Aufruf eines selbständigen Programms
  - ⇨ CGI-Skripte können keine Daten über globale Variable austauschen oder in solchen retten
  - ⇨ Wie kann man Daten zwischen Aufrufen austauschen?
  - ⇨ Idee: persistente Speicher: Dateien / Datenbanken
  - ⇨ **…aber der Bezug muss über den Aufruf hergestellt werden!**

- Jede HTTP Aktion ist ein unabhängiger Vorgang
  - ⇨ HTTP ist ein zustandsloses Protokoll; es enthält keinen Bezug auf vorhergehende Aktionen
  - ⇨ aufeinanderfolgende Request/Response-Aktionen sind unabhängig
  - ⇨ **Wie kann man einen Bezug zwischen Aktionen herstellen?**

Grundidee

- **Verwende einen Identifier, der zwischen Client, Webserver und CGI-Anwendung ausgetauscht wird!**

⇨ Biete entsprechende Methoden zum Erstellen, Löschen und Verwalten solcher IDs

⇨ Biete Methoden für den bequemen Austausch der IDs

⇨ Speicherung beim Client oder (temporär) als Parameter in der URL

Was ist eine Session ?

- Eine zeitweise bestehende Verbindung zwischen einem Server und einem Client
- Zusammenhängende Ausführung mehrerer Aktionen, die dieser Session zugeordnet sind
  - ⇨ z.B. Ausfüllen mehrerer Formulare mit jeweils zugehöriger Rückantwort
  - ⇨ involvierte CGI-Skripte müssen auf Sessiondaten zugreifen können (SessionID, User, User-bezogene Daten)
- Eröffnung durch einen HTTP-Request
  - ⇨ typischerweise "Login"
- Beenden durch einen HTTP-Request
  - ⇨ typischerweise "Logout"
  - ⇨ könnte vom Benutzer vergessen werden
- mehrere Sessions können gleichzeitig offen sein

Datensicherheit ist eine separate Anforderung

SessionID

- wird vom Server beim Login generiert und beim Logout gelöscht
  - ⇨ **eindeutig** soll verschiedene Benutzer unterscheiden
  - ⇨ **zufällig** soll nicht erraten werden können
  - ⇨ **kryptisch** verdeckt das Bildungsgesetz
  - ⇨ **mit Erstellungs- oder Verfallszeitpunkt**
    falls ein Benutzer sich nicht abmeldet

- wird zwischen Server und Client hin- und hergereicht
  - ⇨ in HTML-Datei (Formulardaten, href)
    oder HTTP-Header (Cookie, URL)

in jedem Fall
leicht manipulierbar

- wird im Server bei jeder Seite verwendet,
  um den Benutzer zu identifizieren

SessionID als Cookie

- "Cookies" werden client-seitig (evtl. dauerhaft) gespeichert
  - ⇨ sind einer bestimmten Website zugeordnet
  - ⇨ können von Client und Server gelesen und geschrieben werden
- Cookies werden gesetzt
  - ⇨ per Meta-Tag `<meta http-equiv="set-cookie" content="Keksname=Wert;expires=friday, 31-dec-05 23:59:59 gmt;">`
  - ⇨ oder per HTTP-Header (gemäß RFC 2965)

    ```
    Set-cookie: Keksname=Wert; domain=fh-darmstadt.de;
    expires=friday, 31-dec-05 23:59:59 gmt;
    ```
  - ⇨ oder auch temporär mit JavaScript über HTMLDocument:

    ```
    document.cookie = "nochnKeks=xy";
    ```
- CGI kann Umgebungsvariable HTTP_COOKIE abfragen

  ```
  HTTP_COOKIE=Keksname=Wert; nochnKeks=xy
  ```
- Problem: Benutzer kann Cookies abschalten
  - ⇨ Server kann sich also nicht darauf verlassen und sollte - wenn möglich – auch ohne Cookies funktionieren

Cookies unter PHP

natürlich nicht nur für Sessionverwaltung

- Funktionsdeklaration

```
int setcookie (name [, value [, expire
                  [, path [, domain [, secure]]]]])
```

- Cookie setzen
  - ⇨ verfällt nach 1 Stunde

```
setcookie ("SessionID", $Wert, time()+3600);
```

- Cookie löschen
  - ⇨ mit denselben Parametern wie beim Setzen !
  - ⇨ Verfallszeit in der Vergangenheit bewirkt sofortiges Löschen

```
setcookie ("SessionID", "", time()-3600);
```

- Cookie-Wert auslesen

```
if (isset($_COOKIE["SessionID"]))
    $Wert = $_COOKIE["SessionID"];
```

SessionID ohne Cookies

ID Erzeugen mit PHP:
$SID=md5 (uniqid(mt_rand()));

- optimal bei HTML-**Formularen**
  - ⇨ Verstecktes Formularelement mit generierter ID

```
<input type="hidden"
       name="SessionID" value="xyz1234">
```

  - ⇨ Client schickt dieses per GET sichtbar oder

    per POST unsichtbar zurück

- bei formular-losen HTML-Dateien über **Verweisziel**
  - ⇨ Server generiert URL mit angehängtem Parameter

```
<a href="naechsteSeite.htm?SessionID=xyz1234">
```

  - ⇨ Client schickt dieses per GET sichtbar zurück

PHP Sessionverwaltung

- generiert und übermittelt SessionID
- sichert und restauriert persistente Variable

per Cookie oder URL

in Datei oder DB

- Session eröffnen (Login) bzw. restaurieren (Folgeseiten)
  - ⇨ am Anfang des PHP-Programms: `session_start();`
- persistente Variable registrieren

```
session_register("Zustand");
```

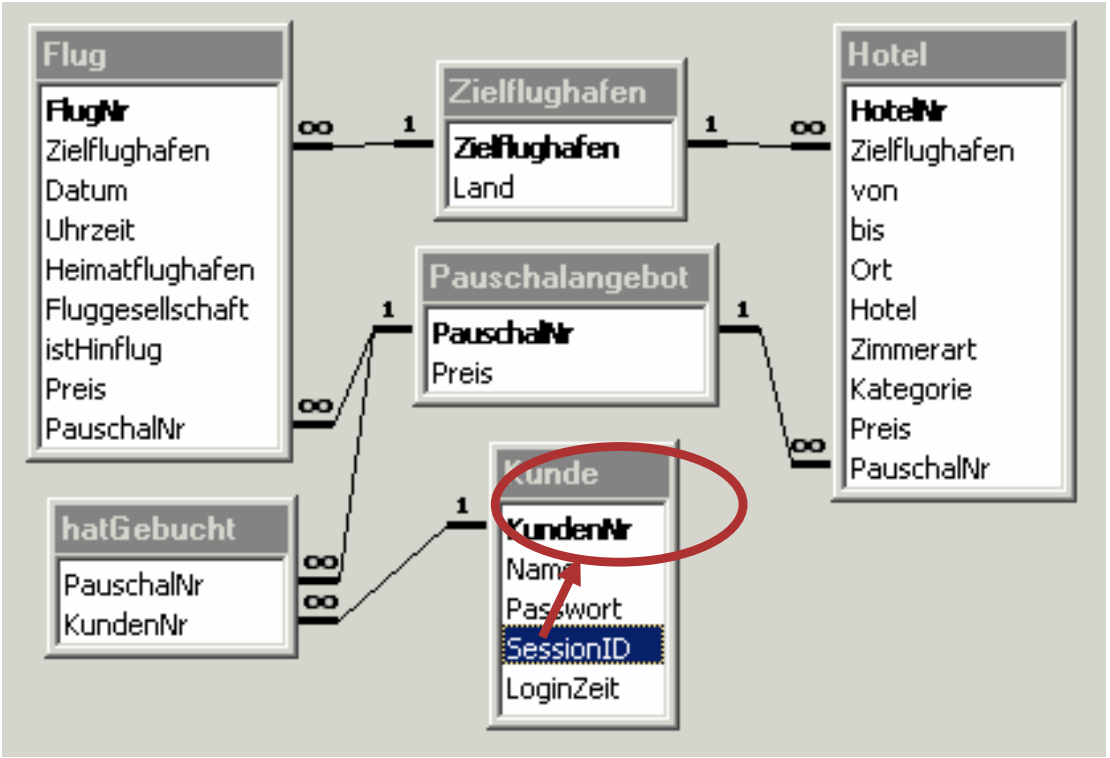
- danach Zugriff auf persistente Variable

```
$_SESSION["Zustand"] = 5;
```

  - ⇨ am Programmende werden persistente Variable autom. gesichert
- Session beenden (Logout)

```
session_destroy();
```

Sessions mit einer Datenbank



- SessionID beim

  Login festlegen

  ⇨ Zeitpunkt speichern

  für Timeout, falls kein

  Logout mehr erfolgt

- Zuordnung SessionID → Kunde in Datenbank speichern
- jeden personenbezogenen Zugriff ergänzen um

```
WHERE ... AND SessionID="$SessionID"
```

- übrigens: die Kunden sind keine Datenbank-User !

  ⇨ Apache/PHP agieren der DB gegenüber als User

SessionID und Datenbank

- beim Login

```
session_start();
$SessionID = session_id();
UPDATE Kunde SET SessionID="$SessionID";
```

- auf Folgeseiten

```
session_start();
$SessionID = session_id();
... WHERE ... AND SessionID="$SessionID";
```

- beim Logout

```
session_start();
$SessionID = session_id();
UPDATE Kunde SET SessionID=Null
       WHERE SessionID="$SessionID";
session_destroy();
```