Datenbank-basierte Webserver

Einführung

Karl Riedling
Institut für Sensor- und Aktuatorsysteme

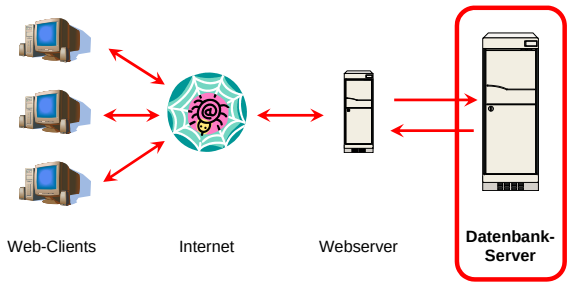
Datenbank-basierte Webserver

- Datenbank mit Web-„Frontend“
  - Datenbank-Funktion steht im Vordergrund
  - Web-Schnittstelle für Eingabe, Wartung oder Ausgabe von Daten
- Website mit Informationen aus einer Datenbank
  - Datenbank läuft im Hintergrund und liefert Daten für bestimmte Seiten einer Website
  - Das Vorhandensein einer Datenbank ist oft transparent für die User

Datenbank-basierte Webserver

- **Datenbank mit Web-„Frontend“**
- Website mit Informationen aus einer Datenbank
- Vorgefertigte Produkte (z.B. CMS) vs. „Eigenbau“
- Moderne Web-Technologien und Datenbanken

Web-Frontend für eine Datenbank



Web-Frontend für eine Datenbank

- Zugriff der Benutzer auf die Datenbank erfolgt über einen konventionellen Web-Browser:
  - Plattformunabhängigkeit: Beliebige Hard- und Software der Benutzerrechner („Clients“) möglich
  - Installation spezieller Software auf den Client-Rechnern ist nicht erforderlich

Web-Frontend für eine Datenbank

- Weitgehende Entkopplung zwischen Server und Clients:
  - Client- und Server-Software sind voneinander (fast) völlig unabhängig
  - Software-Upgrades auf einer Seite erfordern keine Maßnahmen auf der anderen Seite

Web-Frontend für eine Datenbank

- Verwendung des Web-Protokolls (HTTP oder *Secure HTTP* (HTTPS) über TCP/IP) erlaubt bei Bedarf weltweiten Zugriff auf die Datenbank:
  - Im Allgemeinen wenig Probleme mit Routern und Firewalls (im Gegensatz zu z.B. Microsoft Netzwerk oder proprietären Protokollen, z.B. SAP)
  - Einschränkung des Zugriffs auf bestimmte IP-Adressbereiche (z.B. Intranet) ist aber serverseitig möglich

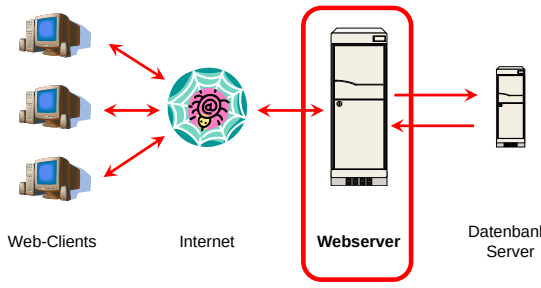
Web-Frontend für eine Datenbank

- Nachteile:
  - „Zustandslosigkeit“ des HTTP(S)-Protokolls (Webseiten sind unabhängig voneinander) erfordert zusätzlichen Aufwand für *Session Management*
  - In vielen Fällen größerer Umfang der zu übertragenden Daten (lokales „Gedächtnis“ zumindest problematisch)
  - Spezieller Aufwand für eine *sichere* Datenübertragung – *Secure HTTP* (HTTPS)

Datenbank-basierte Webserver

- Datenbank mit Web-„Frontend“
- **Website mit Informationen aus einer Datenbank**
- Vorgefertigte Produkte (z.B. CMS) vs. „Eigenbau“
- Moderne Web-Technologien und Datenbanken

Website mit Datenbank-Backend



Website mit Datenbank-Backend

- Dynamische Erstellung von Webseiten:
  - Inhalte stammen nicht (nur) aus statischen Dateien, sondern werden (zumindest zum Teil) automatisch generiert
  - Informationen können leicht modifiziert und für einen bestimmten Zeitraum (z.B. Ankündigungen) oder Benutzerkreis zur Verfügung gestellt werden – typisches Beispiel: *Content Management System* (CMS)

Website mit Datenbank-Backend

- Dynamische Erstellung von Webseiten:
  - Aktualisierung von Inhalten ist einfach und meist ohne HTML-Kenntnisse möglich (meist auch über ein Web-Interface)
  - Bestehende Datenbanken können in die Inhalte der Webseiten eingebunden werden

Website mit Datenbank-Backend

- Beispiel für dynamische Webseiten: Wetter- und Schneeberichte aus einer Region
  - Aufruf über statische URLs, eventuell mit Parametern, z.B.:
    - `http://www.region.com/wetter.php`
    - `http://www.region.com/wetter.php?ort=adorf`
  - Statt gewöhnlicher HTML-Seiten (`.html` oder `.htm`) werden Programme aufgerufen (`.cgi` oder `.asp` oder `.php`)

Website mit Datenbank-Backend

- Interaktive Webseiten:
  - Eingabe von Abfragekriterien für die Suche in *bestehenden* Datenbank-Inhalten – Links oder Web-Formulare
  - Implizite oder explizite Erstellung *neuer* Datenbank-Inhalte (z.B. e-Business) – Web-Formulare
  - Kommunikation (Web-Mail im weitesten Sinn) – Web-Formulare

Website mit Datenbank-Backend

- Interaktive Webseiten:
  - Eingabe von Benutzer-Daten in ein Web-Formular (das oft als dynamische Seite erstellt wurde)
  - Beim Abschicken des Formulars durch den Benutzer werden die im Formular enthaltenen (sichtbaren oder unsichtbaren) Daten an ein auf dem Server laufendes Programm (`.cgi` oder `.asp` oder `.php`) übergeben

Website mit Datenbank-Backend

- Identifikation von Benutzern:
  - Das HTTP(S)-Protokoll ist *zustandslos*, d.h., die Aufrufe von Seiten sind voneinander unabhängig
  - Spezielle Maßnahmen zur Identifikation von Benutzern erforderlich, z.B.:
    - Cookies (werden vom Server ausgegeben, am Client-Rechner verwaltet und bei jedem neuen Seiten-Aufruf an den Server übergeben)
    - Session-IDs (werden vom Server ausgegeben und verwaltet und von Seite zu Seite „weitergereicht“)

Datenbank-basierte Webserver

- Datenbank mit Web-„Frontend“
- Website mit Informationen aus einer Datenbank
- **Vorgefertigte Produkte (z.B. CMS) vs. „Eigenbau“**
- Moderne Web-Technologien und Datenbanken

Vorgefertigte Produkte (CMS)

- Eine große Anzahl vorgefertigter Datenbank-basierter Web-Systeme (z.B. *Content Management Systeme* – CMS) ist – größtenteils kostenlos – verfügbar.
- Vorteile:
  - Websites mit fast beliebiger Komplexität sind ohne großen Aufwand realisierbar
  - Meist extrem komfortable Wartungs-Schnittstellen
- Nachteile:
  - Trotz hoher Komplexität oft nicht genau die benötigte Funktionalität verfügbar

Vorgefertigte Produkte (z.B. CMS)

- Nachteile (Fortsetzung):
  - Das Selbst-Programmieren allenfalls benötigter Erweiterungen ist extrem aufwändig und erfordert tiefgreifende Kenntnisse über die Abläufe in der vorhandenen Software
  - Speziell *Open Source*-Systeme sind wegen ihrer Komplexität und großen Verbreitung häufig ein Ziel von Angriffen – wenn ein vorgefertigtes System verwendet wird, sollten Sicherheits-Updates (möglichst automatisiert) ohne Verzögerungen eingespielt werden!

„Eigenbau“-Systeme

- Vorteile
  - Es kann genau die benötigte Funktionalität, nicht mehr und nicht weniger, implementiert werden
  - Der Programmcode von „Eigenbau“-Systemen ist nicht publik; allfällige Sicherheitslücken (ausgenommen solche, die mit den üblichen Hacker-Tools wie *SQL-Injection* oder *Cross Site Scripting* leicht gefunden werden können) sind vielfach nicht ganz so kritisch

„Eigenbau“-Systeme

- Nachteile
  - Größerer Implementierungsaufwand (speziell, wenn nur Standard-Funktionalität benötigt wird)
- Auch in Eigenbau-Systeme können bei Bedarf vorgefertigte Module (z.B. WYSIWYG-Editoren wie TinyMCE) eingebunden werden. *Vorsicht*: Wenn diese Module auf öffentlich zugänglichen Seiten verwendet werden, gelten die gleichen Überlegungen bezüglich oftmaliger Sicherheits-Updates wie bei vorgefertigten Produkten!

Generell für alle Systeme

- Programmcode, der nach bestem Wissen und Gewissen „sicher“ programmiert wurde, muss das nicht sein!
- Es gibt Organisationen im Web, z.B. OpenBugBounty.org (https://www.openbugbounty.org/), in denen „Edel-Hacker“ tätig sind, die *Vulnerabilities* an bestehenden Websites gegen ein (finanzielles oder auch ideelles) Entgelt lokalisieren – sehr zu empfehlen für sicherheitskritische Anwendungen!

Generell für alle Systeme

- Es werden mit den üblichen Hacker-Tools (z.B. *SQL-Injection* oder *Cross Site Scripting)* Angriffe gegen die Website vorgenommen, bei denen bewusst keine Schäden verursacht werden.
- In vielen Fällen reichen einfache Maßnahmen (*Escapen* von Inhalten, die in SQL-Abfragen verwendet werden, Filtern von HTML-Tags in Eingaben aus der Website und Ähnliches), um Sicherheitslücken zu schließen.
- Die „Edel-Hacker“ leisten zumeist wertvolle Unterstützung bei der Auswahl der Maßnahmen.

Datenbank-basierte Webserver

- Datenbank mit Web-„Frontend“
- Website mit Informationen aus einer Datenbank
- Vorgefertigte Produkte (z.B. CMS) vs. „Eigenbau“
- **Moderne Web-Technologien und Datenbanken**

Moderne Web-Technologien und Datenbanken

- **„Web 2.0“**
- Verbesserte Benutzer-Schnittstellen

Web 2.0-Technologien und Datenbanken

- „Web 2.0“: Schlagwort für eine Reihe unterschiedlicher Technologien mit Zielsetzungen
  - Informations-Abonnements (z.B. RSS)
  - Weblogs
  - Soziale Netzwerke
  - Webservices
  - Kooperativer Informationsaustausch für alle Nutzer

Web 2.0-Technologien und Datenbanken

- Alle diese Technologien erfordern eine dynamische Datenhaltung im Hintergrund, also eine Datenbank (im weitesten Sinne)
- Nicht nur Abruf der Inhalte über eine Web-Schnittstelle, sondern auch ihre Erstellung und Wartung
- Weitere Beispiele für Web 2.0-Technologien:
  - *Smart Browsing*
  - *User Tracking*

Web 2.0-Technologien und Datenbanken

- *Smart Browsing*:
  - Identifikation von Benutzern – meist über *Cookies*
  - Konfiguration einer Website (z.B. einer Suchmaschine) nach den Wünschen des Benutzers
  - Registrierung von Zugriffen auf bestimmte Webseiten (z.B. Unterscheidung zwischen gelesener und nicht gelesener Information)
  - E-Mail-Benachrichtigung z.B. bei Änderungen bestimmter Inhalte

Web 2.0-Technologien und Datenbanken

- *User Tracking*:
  - Verfolgen des Browsing-Verhaltens der Benutzer zu statistischen Zwecken oder zur Unterstützung von *Smart Browsing*
  - Oft Server-übergreifende Nachverfolgung – Datenschutz-Problematik!
  - Adaptive Systeme: Benutzer bekommen je nach Zielsetzung bevorzugt Inhalte angeboten, die sie besonders häufig (oder besonders selten) besucht haben

Moderne Web-Technologien und Datenbanken

- „Web 2.0“
- **Verbesserte Benutzer-Schnittstellen**

Verbesserte Benutzer-Schnittstellen

- „Klassische“ Web-Applikationen werden vielfach als umständlich oder schwerfällig empfunden:
  - Jede User-Interaktion erfordert das Erstellen einer neuen Seite:
    - längere Response-Zeiten
    - bei einem Neu-Aufbau steht die Seite in der Regel wieder an ihrem Anfang
  - User wünschen sich Verhalten wie bei Desktop-Applikationen (Seiten ändern sich dynamisch, ohne jedes Mal neu aufgebaut zu werden)

Verbesserte Benutzer-Schnittstellen

- Verbesserte Usability von Web-Applikationen erfordert *Client-seitige* Software:
  - Client-seitige *Stand alone*-Applikationen (z.B. Java)
  - Verwendung der Browser-eigenen Script-Funktionalität (z.B. JavaScript – *Ajax* (**A**synchronous **J**avaScript **a**nd **X**ML))
- In beiden Fällen kommuniziert die Client-seitige Software im Hintergrund (für den User nicht merkbar) mit dem Server – nur sinnvoll bei komplexem Informations-Angebot!