**Einführung in die Webprogrammierung**

Das Internet ist heute ein zentraler Bestandteil unseres Lebens, und fast jede Anwendung ist in irgendeiner Form webbasiert. Von einfachen statischen Webseiten über datenbankgestützte Webanwendungen bis hin zu hochinteraktiven Single Page Applications (SPAs) bietet die Webentwicklung viele Möglichkeiten.

**Historische Entwicklung**

Die Webentwicklung hat sich seit den frühen 1990er Jahren stark verändert:

* **1990er Jahre** – Statische HTML-Seiten: Nur einfache Dokumente mit Links.
* **2000er Jahre** – Dynamische Webseiten mit PHP, ASP, ASP.NET,...
* **2010er Jahre** – AJAX & moderne JavaScript-Frameworks ermöglichen SPAs.

**Arten von Webanwendungen**

**Statische Webseiten**

Eine statische Webseite besteht aus HTML (und optional CSS und JavaScript).  
**Merkmale:**

* Keine serverseitige Verarbeitung.
* Jeder Nutzer sieht den gleichen Inhalt.
* Änderungen erfordern manuelle Anpassungen im Code.  
  **Beispiele:**
  + Unternehmenswebseiten
  + Portfolio-Seiten
  + Dokumentationen

**Vorteile:**

Schnell zu laden, einfach zu hosten.

**Dynamische Webanwendungen**

Dynamische Webseiten generieren Inhalte basierend auf Benutzereingaben und/ oder Datenbanken.

**MVC (Model-View-Controller) Architektur:**

**Model (M)**: Verwaltet die Daten und Geschäftslogik.

**View (V)**: Präsentation der Daten für den Benutzer.

**Controller (C)**: Vermittelt zwischen View und Model, verarbeitet Anfragen.

**Beispiele:**

* Online-Shops (Amazon, eBay)
* Content-Management-Systeme (WordPress, Joomla)
* Unternehmenssoftware

**Vorteile:**  
Gute Trennung von Logik und Darstellung.  
SEO-freundlich, da HTML vom Server generiert wird.  
**Nachteile:**  
Häufige Seitenneuladungen.  
Höhere Serverlast.

**Single Page Applications (SPAs)**

Moderne Webanwendungen laden eine einzige HTML-Seite und aktualisieren den Inhalt dynamisch über APIs.

**Technologien:**

**Frontend**: React, Angular, Vue.js, Blazor

**Backend**: RESTful APIs oder GraphQL

**Datenbanken**: NoSQL (MongoDB) oder SQL (PostgreSQL, MySQL)

**Merkmale:**

Initial langsamer, aber danach sehr reaktionsschnell.

Kommunikation mit dem Server erfolgt asynchron über API-Calls.

**Beispiele:**

Gmail, Facebook, X

Online-Banking-Apps

Webbasierte Office-Anwendungen (Google Docs, Office 365)

**Vorteile:**  
Fließende Benutzererfahrung (kein Neuladen der Seite).  
Datenübertragung ist effizienter.  
**Nachteile:**  
Komplexe Entwicklung.  
SEO-Optimierung schwieriger.

**Grundlagen des HTTP-Protokolls**

HTTP (Hypertext Transfer Protocol) ist das Kommunikationsprotokoll des Internets.

**Eigenschaften:**

* **Zustandslos**: Jeder Request ist unabhängig.
* **Client-Server-Modell**: Der Browser (Client) sendet Anfragen an den Webserver.
* **Textbasiert**: Kommunikation erfolgt über einfache Textnachrichten.

**Das Request-Response-Modell**

1. Der Browser sendet eine HTTP-Anfrage (Request) an den Server.
2. Der Server verarbeitet die Anfrage und sendet eine Antwort (Response) zurück.

**Aufbau einer HTTP-Anfrage:**

GET /index.html HTTP/1.1

Host: www.example.com

User-Agent: Mozilla/5.0

**Aufbau einer HTTP-Antwort:**

HTTP/1.1 200 OK

Content-Type: text/html

Content-Length: 1234

**HTTP-Methoden**

|  |  |  |
| --- | --- | --- |
| Methode | Bedeutung | Anwendungsfall |
| GET | Holt Daten vom Server. | Eine Webseite aufrufen. |
| POST | Sendet Daten an den Server. | Formular abschicken. |
| PUT | Aktualisiert eine Ressource. | Profil bearbeiten. |
| DELETE | Löscht eine Ressource. | Konto löschen. |

**HTTP-Statuscodes**

**Statuscodes kategorisieren die Antwort des Servers:**

|  |  |  |
| --- | --- | --- |
| Kategorie | Bedeutung | Beispiel |
| 1xx | Informativ | 100 Continue |
| 2xx | Erfolg | 200 OK |
| 3xx | Umleitung | 301 Moved Permanently |
| 4xx | Client-Fehler | 404 Not Found |
| 5xx | Server-Fehler | 500 Internal Server Error |

**Praxisbeispiel:**

GET /seite.html HTTP/1.1

Host: www.example.com

Antwort des Servers:

HTTP/1.1 404 Not Found

Die angeforderte Seite existiert nicht.

**Vergleich der Webtechnologien**

|  |  |  |
| --- | --- | --- |
| Merkmal | Statische Webseiten | MVC-Webanwendungen |
| Dynamik | nein | ja |
| Ladezeit | Schnell | Mittel |
| Server-Interaktion | nein | Hoch |
| SEO-Freundlichkeit | ja | ja |
| Komplexität | Gering | Mittel |

Der netzwerktechnische Ablauf einer HTTP-Anfrage durchläuft mehrere Schritte, die auf verschiedenen Ebenen des OSI-Modells stattfinden. Hier ist eine detaillierte Beschreibung des Prozesses:

**1. DNS-Auflösung (Falls notwendig)**

* Falls der Client nur eine URL (z. B. https://example.com) hat, muss er zunächst die IP-Adresse des Servers ermitteln.
* Der Client fragt dazu einen DNS-Server nach der IP-Adresse der Domain.
* Der DNS-Server liefert die zugehörige IP-Adresse zurück.

**2. Aufbau der TCP- oder TLS-Verbindung**

* Der Client initiiert eine TCP-Verbindung zum Server auf Port 80 (HTTP) oder Port 443 (HTTPS).
* Falls HTTPS verwendet wird, erfolgt nach dem TCP-Handshake ein **TLS-Handshake**:
  + Der Client und der Server tauschen Zertifikate und Schlüssel aus.
  + Eine verschlüsselte Verbindung wird etabliert.

**3. Senden der HTTP-Anfrage**

* Der Client sendet eine HTTP-Anfrage an den Server, z. B.:
* GET /index.html HTTP/1.1
* Host: example.com
* User-Agent: Mozilla/5.0
* Accept: text/html
* Diese Anfrage enthält verschiedene Header, die Informationen über den Client, akzeptierte Formate und Cookies enthalten können.

**4. Verarbeitung der Anfrage durch den Server**

* Der Webserver empfängt die Anfrage und verarbeitet sie.
* Falls die Anfrage Daten aus einer Datenbank benötigt, kann der Server eine zusätzliche Datenbankanfrage stellen.
* Der Server erstellt die HTTP-Antwort und sendet sie zurück.

**5. Empfangen der HTTP-Antwort**

* Der Client empfängt die Antwort, die einen Statuscode (z. B. 200 OK oder 404 Not Found), Header und ggf. den HTML-Inhalt enthält.

**6. Darstellung der Antwort**

* Falls es sich um eine Webseite handelt, verarbeitet der Browser den HTML-Code und lädt ggf. weitere Ressourcen (CSS, JavaScript, Bilder) über separate HTTP-Anfragen nach.

Dieser Ablauf passiert innerhalb von Millisekunden und kann durch Caching-Mechanismen auf verschiedenen Ebenen (Browser, Proxy, CDN) optimiert werden.