WEBENTWICKLUNG

Inhalt

- Dynamische Websites
- Web 2.0
- Mobilgeräteunterstützung
- HTTP
- Angriffsmöglichkeiten

Übersicht

Dynamische Websites

- Überblick
  - CGI
  - PHP
  - ASP
  - JSP
  - Applet versus Servlet

Dynamische Websites

- generieren Inhalte zur Laufzeit auf dem Server
- oft auf Basis von Benutzereingaben oder Datenbankinhalten
- Inhalt kann sich ändern, ohne dass die Seite manuell angepasst werden muss

Technologie	Sprache/Plattform	Serverseitig?	Status	Typische Einsatzzeit
CGI	Beliebig (Perl, Python, C)	✅	Veraltet	1990er
ASP	VBScript / später .NET	✅	Teilweise überholt (Classic ASP), .NET aktuell	1996–heute
JSP	Java	✅	Aktiv (Java EE, Jakarta EE)	Ab 1999
PHP	PHP	✅	Sehr aktiv	Ab 1995

CGI

Common Gateway Interface

- Erstes Modell für dynamische Websites
- CGI-Skripte sind eigenständige Programme, die vom Webserver für jede Anfrage neu gestartet werden
- Sehr ineffizient bei hoher Last
- Typischerweise in Perl oder C geschrieben
- 🔴 Heute kaum noch verwendet

ASP

Active Server Pages

- Entwickelt von Microsoft
- Nutzt VBScript oder später C# (ASP.NET)
- Läuft auf IIS (Internet Information Services)
- Moderne Form: ASP.NET Core – plattformunabhängig, performant
- Wird in vielen Enterprise-Umgebungen genutzt

JSP

JavaServer Pages

- Teil der Java EE-Plattform
- HTML + Java-Code in einem Dokument
- Übersetzt vom Server in ein Servlet
- Starke Einbindung in Java-Ökosystem
- Weniger verbreitet im Vergleich zu PHP oder .NET, aber sehr mächtig

PHP

Hypertext Preprocessor

- Open Source, weit verbreitet
- Einfach zu lernen und direkt im HTML eingebettet
- Läuft auf fast allen Servern
- Große Community, viele CMS (z. B. WordPress, Joomla, Drupal) basieren darauf
- ⚠️ Früher oft für Sicherheitsprobleme bekannt – heute sehr verbessert

Applet versus Servlet

JAVA APPLET

- Java-Programm, das in einem Webbrowser (Sandbox) ausgeführt wird
- läuft clientseitig und benötigt ein Java-Plugin im Browser
- Heute nicht mehr unterstützt

JAVA SERVLET

- serverseitiges Java-Programm
- Läuft auf einem Webserver (Servlet-Container)
- Verarbeitet HTTP-Anfragen, z. B. Formulareingaben oder API-Requests
- Generiert HTML oder JSON als Antwort auf HTTP-Anfragen
- Wird weiterhin aktiv verwendet

Übersicht

Web 2.0

- Web Entwicklungsstufen
- Web 2.0
- Web 3.0
- RIA
- AJAX

Web Entwicklungsstufen

Aspekt	Web 1.0 – Das statische Web	Web 2.0 – Das soziale Web	Web 3.0 – Das semantische & dezentrale Web
Zeitraum	ca. 1990–2004	ca. 2004–2020	ab ca. 2020 (in Entwicklung)
Nutzerrolle	Reine Konsumenten	Prosumenten (Produzenten + Konsumenten)	Eigentümer & kontrollierende Akteure
Inhalte	Statisch (HTML-Seiten)	Dynamisch, nutzergeneriert	Maschinell verknüpft, personalisiert
Interaktivität	Kaum vorhanden	Hoch (Kommentare, Likes, Beiträge)	KI-gestützt, über intelligente Agenten
Technik	HTML, HTTP, Frames	AJAX, RSS, APIs, soziale Plattformen	KI, Blockchain, dApps, RDF, SPARQL
Beispielseiten	Broschüren, Firmen-Websites	Wikipedia, Facebook, YouTube, Twitter	dApps, Web3-Protokolle, Smart Contracts
Datenhoheit	Anbieter	Plattform	Nutzer selbst (dezentrale Kontrolle)
Beziehung Mensch↔Maschine	Einseitig (Lesen)	Zweiseitig (Dialog)	Vernetzt & kontextbezogen (Bedeutung verstehen)
Geschäfts-modell	Werbung, statische Angebote	Plattformökonomie, Datenhandel	Tokenisierung, Blockchain-basierte Ökonomie
Stichworte	„Lesen“	„Lesen & Schreiben“	„Lesen, Schreiben & Eigentum“

Web 2.0

- Begriff wurde um 2004 durch Tim O’Reilly geprägt
- zweite Entwicklungsstufe des World Wide Web
- NutzerInnen sind nicht mehr nur Konsumenten, sondern auch aktive Produzenten von Inhalten
- Nutzerbeteiligung, Vernetzung und Dynamik

Merkmale:

- Interaktivität
- Partizipation
- Vernetzung
- Soziale Software
- Dynamische Inhalte
- Cloudbasierte Dienste
- APIs & Mashups

Web 2.0 Anwendungen

Anwendung	Beschreibung	Web 2.0-Merkmale
Social Networks	Nutzerprofile, Teilen von Beiträgen, Kommentare, Gruppen	Soziale Vernetzung, User-Content
Wikis	Kollaborative Bearbeitung von Texten	Schwarmintelligenz, offene Mitarbeit
Blogs	Persönliche oder thematische Beiträge	Kommentarfunktion, RSS, Leserbindung
Twitter / Microblogging	Kurze Nachrichten, schnelle Verbreitung	Echtzeit-Interaktion, Hashtags
Foren	Diskussion zu Fachthemen oder Interessen	Nutzerfragen und Antworten, Community-Ranking
Podcasts	Audio-Inhalte von Laien oder Profis, regelmäßig verfügbar	Demokratisierung von Medienproduktion

Web 3.0

- Inhalte sind semantisch verknüpft, künstlich intelligent verarbeitet und dezentral gespeichert
- **Ziel:**
  - Web, das smarter, individueller und kontrollierter durch den Nutzer funktioniert
- oft auch als „semantisches“ oder „dezentrales Web“ bezeichnet

Merkmale:

- Dezentralisierung
- Semantik
- Interoperabilität
- Nutzerkontrolle & Datenhoheit
- Tokenisierung
- Smart Contracts
- Künstliche Intelligenz
- Keine zentrale Plattform-Abhängigkeit

Web 3.0 Anwendungen

Anwendung / Projekt	Beschreibung
Ethereum	Plattform für Smart Contracts und dApps
Brave Browser	Datenschutzorientierter Browser mit Belohnungssystem (BAT Token)
Uniswap / Aave	Dezentrale Finanzplattformen (DeFi)
IPFS / Filecoin	Dezentrale Speicherung von Dateien
Steemit	Blockchain-basiertes Blog-Netzwerk
OpenSea	Marktplatz für NFTs
Solid (Projekt von Tim Berners-Lee)	Kontrolle über eigene Daten durch persönliche Datenspeicher (Pods)

RIA

Rich Internet Application

- Webanwendung, die sich wie eine klassische Desktop-Anwendung verhält
- bietet eine interaktive, dynamische Benutzeroberfläche, häufig mit multimedialen Inhalten, ohne dass die Seite ständig neu geladen werden muss
- Läuft im Browser
- benötigt oft zusätzliche Plugins (Flash, JavaFX, Silverlight – heute größtenteils veraltet)
- **Ziel:** Bessere User Experience (wie bei nativen Anwendungen)
- **Beispiele** (historisch): Adobe Flash-Webanwendungen, Java Applets, Silverlight-Anwendungen

AJAX

Asynchronous JavaScript and XML

- Technologie-Ansatz zur Erstellung von asynchronen Webanwendungen, bei dem Daten im Hintergrund vom Server geladen werden, ohne dass die Seite neu geladen werden muss
- kein eigenständiges Produkt, sondern eine Kombination aus:
  - JavaScript (für Interaktivität)
  - XMLHttpRequest (für asynchrone Serveranfragen)
  - HTML/CSS (für Darstellung)
  - JSON/XML (für Datenaustausch)
- heute Kernbestandteil moderner Websites

Übersicht

Mobilgeräteunterstützung

- Anforderungen mobiler Geräte an die Webentwicklung
- Native Apps versus HTML5/JavaScript

Anforderungen mobiler Geräte an die Webentwicklung

Offline-Fähigkeit

- Mobile NutzerInnen erwarten, dass Apps auch bei fehlender Internetverbindung funktionieren.
- ! Wichtig für z. B. Notizen, Kalender, Karten, Medien
- Technologien:
  - Service Worker (für Web-Apps)
  - IndexedDB/LocalStorage (Daten zwischenspeichern)
- Progressive Web Apps (PWA) bieten Offline-Modus wie native Apps

Anforderungen mobiler Geräte an die Webentwicklung

Verschiedene Plattformen

- Mobile Anwendungen sollen plattformübergreifend funktionieren (z. B. Android, iOS, Web)
- **Ziel:** Einmal entwickeln, überall nutzen
- Strategien:
  - Native Entwicklung (pro Plattform) → hoher Aufwand
  - Cross-Platform Frameworks:
    - React Native, Flutter, Xamarin
    - Hybrid-Apps mit HTML5/Cordova
- Spart Zeit und Kosten, aber ggf. Abstriche bei Performance oder UX

Anforderungen mobiler Geräte an die Webentwicklung

Bandbreite

- Mobilgeräte sind oft in Netzen mit instabiler oder langsamer Verbindung (3G, schwaches WLAN)
- **Konsequenzen:**
  - Bilder und Medien komprimieren
  - Lazy Loading, Caching nutzen
  - Minimale Datenmengen übertragen (JSON statt XML)
- **Tools:**
  - Responsive Images
  - Brotli/GZIP-Komprimierung
  - Lighthouse-Analyse (Google)

Anforderungen mobiler Geräte an die Webentwicklung

Auflösungen

- Mobile Displays sind kleiner, hochauflösend und touchbasiert
- Anforderungen an **Design**:
  - Responsive Design
  - Mobile First-Ansatz
  - Große Schaltflächen, einfache Navigation
  - Kein Hover, keine Maus
- **Frameworks**:
  - Bootstrap, Tailwind CSS, Material UI

Native Apps versus HTML5/JavaScript

Kriterium	Native App	HTML5/JavaScript (Web-App oder Hybrid)
Performance	Sehr hoch	Gut bis mittel
Zugriff auf Hardware	Vollständig	Eingeschränkt, aber besser dank Web APIs
Verfügbarkeit	App Stores	Direkt über Browser
Entwicklungsaufwand	Höher (mehrere Plattformen)	Geringer bei Cross-Development

Entwicklungsstrategien

Strategie	Beschreibung
Native Entwicklung	Für jede Plattform einzeln entwickeln – hoher Aufwand, beste Performance
Hybride Entwicklung	Webtechnologien (HTML/CSS/JS) + native Wrapper (z. B. Cordova, Ionic)
Cross-Platform-Frameworks	Ein Codebase, mehrere Plattformen – z. B. mit Flutter, React Native, Xamarin
Progressive Web App (PWA)	Web-App mit App-ähnlichen Funktionen (offline, installierbar)

Übersicht

HTTP

- Methoden
- Konzepte
- Statuscodes

Methoden

Methode	Sicher	Idempotent	Cachefähig
`GET`	Ja	Ja	Ja
`HEAD`	Ja	Ja	Ja
`OPTIONS`	Ja	Ja	Nein
`TRACE`	Ja	Ja	Nein
`PUT`	Nein	Ja	Nein
`DELETE`	Nein	Ja	Nein
`POST`	Nein	Nein	Bedingt*
`PATCH`	Nein	Nein	Bedingt*
`CONNECT`	Nein	Nein	Nein

* POST und PATCH sind cachefähig, wenn Antworten explizit Frische-Informationen (max-age oder expire) und einen passenden Content-Location-Header enthalten

Methoden Begriffsklärung

GET

- fordert eine Darstellung der angegebenen Ressource an
- Anfragen, die GET verwenden, sollten nur Daten abrufen und keinen Anfrage-Inhalt enthalten

HEAD

- verlangt eine Antwort, die mit einer GET-Anfrage identisch ist, jedoch ohne Antwortkörper

POST

- übermittelt eine Entität an die angegebene Ressource
- Verursacht oft eine Zustandsänderung oder Nebeneffekte auf dem Server

PUT

- ersetzt alle aktuellen Darstellungen der Zielressource durch den Anfrage-Inhalt

DELETE

- löscht die angegebene Ressource

CONNECT

- etabliert einen Tunnel zum vom Zielressource identifizierten Server

OPTIONS

- beschreibt die Kommunikationsoptionen für die Zielressource

TRACE

- führt einen Nachrichtenschleifen-Test entlang des Pfads zur Zielressource durch

PATCH

- wendet partielle Modifikationen auf eine Ressource an

Konzepte

SAFE

- Eine HTTP-Methode ist safe, wenn sie keine Veränderungen am Serverzustand verursacht
- Safe-Methoden dürfen gefahrlos mehrfach ausgeführt werden, z. B. durch Crawler oder beim Neuladen der Seite.

IDEMPOTENT

- Eine HTTP-Methode ist idempotent, wenn mehrfache identische Anfragen denselben Effekt haben wie eine einzige

Statuscodes

Klasse	Bereich	Bedeutung
1xx	100–199	Informativ (Request wird weiterverarbeitet)
2xx	200–299	Erfolg (Request war erfolgreich)
3xx	300–399	Weiterleitung (z. B. Umleitung)
4xx	400–499	Clientfehler (Request ist fehlerhaft)
5xx	500–599	Serverfehler (Server kann Request nicht verarbeiten)

1xx – Informativ

- In der Praxis selten relevant für Standard-Webentwicklung

Code	Bedeutung
100	Continue – Der Client kann weitersenden
101	Switching Protocols – Protokollwechsel wird akzeptiert (z. B. zu WebSocket)

2xx – Erfolg

Code	Bedeutung
200	OK – Anfrage erfolgreich, Antwort im Body
201	Created – Ressource wurde erfolgreich erstellt
204	No Content – Anfrage erfolgreich, aber kein Inhalt zurückgegeben

3xx – Weiterleitung

Code	Bedeutung
301	Moved Permanently – Ressource dauerhaft verschoben
302	Found (bzw. „Temporary Redirect“) – Temporäre Weiterleitung
304	Not Modified – Ressource nicht verändert (Cache kann verwendet werden)

4xx – Clientfehler

Code	Bedeutung
400	Bad Request – Ungültige Anfrage (z. B. fehlerhafte JSON-Syntax)
401	Unauthorized – Nicht authentifiziert (aber theoretisch möglich)
403	Forbidden – Zugriff verboten (auch mit Login)
404	Not Found – Ressource nicht gefunden
405	Method Not Allowed – Methode nicht erlaubt (z. B. PUT auf nur-GET-Ressource)
429	Too Many Requests – Ratenlimit überschritten

5xx – Serverfehler

Code	Bedeutung
500	Internal Server Error – Allgemeiner Serverfehler
501	Not Implemented – Methode wird vom Server nicht unterstützt
502	Bad Gateway – Fehlerhafter Proxy oder Gateway
503	Service Unavailable – Dienst temporär nicht erreichbar
504	Gateway Timeout – Antwort vom Backend dauerte zu lange

Übersicht

Angriffsmöglichkeiten

- SQL-Injection
- Cross-Site Scripting
- Cross-Site Request Forgery
- Session Hijacking
- Denial of Service
- Distributed Denial of Service

Angriffsmöglichkeiten

Angriffsart	Was ist das?	Beispielhafte Bedrohung	Schutzmaßnahmen
SQL-Injection	Einschleusen von SQL-Befehlen über Benutzereingaben	Zugriff auf, Veränderung oder Löschung von Datenbankinhalten	- Prepared Statements / Parametrisierte Queries- Eingaben validieren & escapen- Keine direkten SQL-Strings bauen
XSS (Cross-Site Scripting)	Einschleusen von Skriptcode in Webseiten	Angreifer bringt Browser dazu, JavaScript auszuführen (z. B. zum Cookie-Diebstahl)	- HTML/JavaScript escapen- Input/Output filtern- Content Security Policy (CSP) einsetzen
CSRF (Cross-Site Request Forgery)	Missbrauch bestehender Authentifizierung durch ungewollte Aktionen im Namen des Nutzers	Angreifer nutzt eingeloggte Sitzung, um z. B. eine Überweisung auszulösen	- CSRF-Tokens verwenden- Referer prüfen- Nur POST-Requests für kritische Aktionen
Session Hijacking	Diebstahl oder Manipulation einer aktiven Session-ID	Angreifer übernimmt Sitzung des Users, z. B. durch gestohlene Cookies	- HTTPS verwenden- Session-Timeouts- Session-IDs regelmäßig erneuern- Cookies mit Secure und HttpOnly flaggen
DoS (Denial of Service)	Überlastung des Systems durch viele Anfragen von einer Quelle	Server wird unbenutzbar für echte Nutzer	- Ratenbegrenzung- Zeitverzögerungen bei wiederholten Zugriffen- Ressourcen effizient verwalten
DDoS (Distributed Denial of Service)	Gleichzeitige Angriffe von vielen Quellen (z. B. Botnet)	Massive Serverüberlastung, oft bei großen Angriffen (z. B. gegen Unternehmen)	- DDoS-Schutz über Firewall/CDN (z. B. Cloudflare)- IP-Filterung- Lastverteilung (Load Balancer)

SQL-Injection

- Angreifer schleusen manipulierte SQL-Befehle in Formularfelder oder URLs ein, um Datenbankabfragen zu beeinflussen

- **Schutzmaßnahmen:**
  - Prepared Statements/Parameterbindung
  - Eingabevalidierung & Whitelisting
  - ORM-Frameworks verwenden
  - Fehlermeldungen abschalten

Cross-Site Scripting

- Angreifer schleusen JavaScript in eine Webseite ein, das dann im Browser anderer Nutzer ausgeführt wird

- **Schutzmaßnahmen:**
  - Ausgabe escapen (Output Encoding)
  - Content Security Policy (CSP)
  - Keine Benutzereingaben direkt in JavaScript/HTML einbauen
  - Input-Sanitizing
  - HttpOnly-Cookies setzen

Cross-Site Request Forgery

- eingeloggter Nutzer wird durch eine andere Webseite dazu gebracht, unbeabsichtigt eine Aktion im Namen seiner aktiven Sitzung auszuführen

- **Schutzmaßnahmen:**
  - CSRF-Token in Formularen verwenden
  - Nur POST für kritische Aktionen
  - SameSite-Cookie-Flag setzen
  - Referer-/Origin-Header prüfen

Session Hijacking

- Angreifer stiehlt oder errät eine gültige Session-ID (z. B. aus einem nicht verschlüsselten Cookie) und übernimmt damit die Identität eines Nutzers

- **Schutzmaßnahmen:**
  - HTTPS erzwingen (TLS/SSL)
  - HttpOnly- & Secure-Flags für Cookies
  - Session-Timeout & Re-Login
  - Session-Fixation verhindern
  - IP-/User-Agent-Bindung (optional)

Denial of Service

- Angreifer überlastet die Anwendung mit zu vielen Anfragen → Server oder Dienste fallen aus

- **Schutzmaßnahmen:**
  - Rate Limiting
  - Captcha bei Formularen
  - Ressourcen schützen
  - Asynchrone Verarbeitung mit Warteschlangen

Distributed Denial of Service

- Viele Geräte (oft Botnetze) greifen den Server gleichzeitig an → System wird durch „normale“ Anfragen aus verschiedenen Quellen blockiert

- **Schutzmaßnahmen:**
  - CDN mit DDoS-Schutz einsetzen
  - Firewall-Regeln
  - Traffic-Analyse & Frühwarnsysteme
  - Load Balancer & Skalierung