



FH Salzburg

VO Web-Technologien

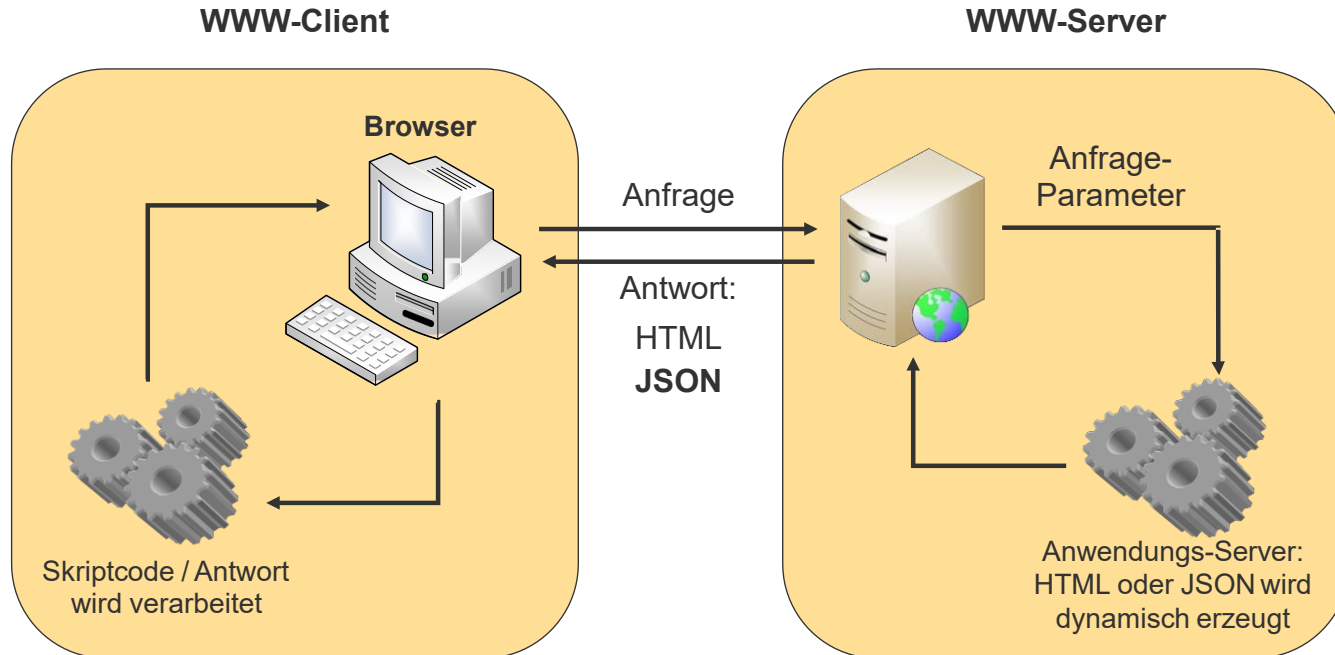
Einheit 6, Oliver Jung

Technik
Gesundheit
Medien

Dynamisch-interaktive Webseiten



Web-Programmierung ist sowohl **clientseitig** als auch **serverseitig** möglich



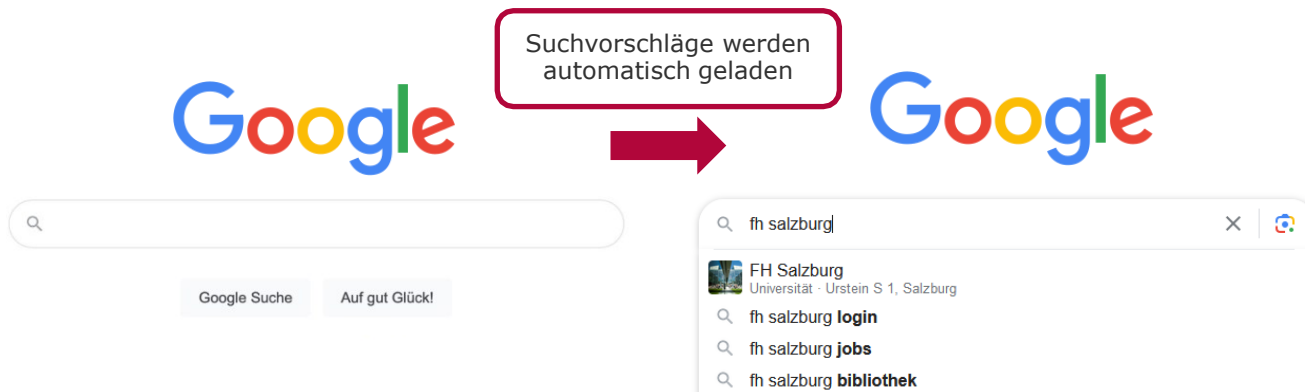
AJAX – Asynchronous JavaScript And XML



Grundidee

„Interner“ Datenaustausch zwischen Client und Server über JavaScript

- Webseite muss bei Änderungen nicht komplett neu geladen, nur ausgewählte, z.B. gerade aktualisierte Bereiche der Webseite werden aktualisiert



AJAX – Asynchronous JavaScript And XML



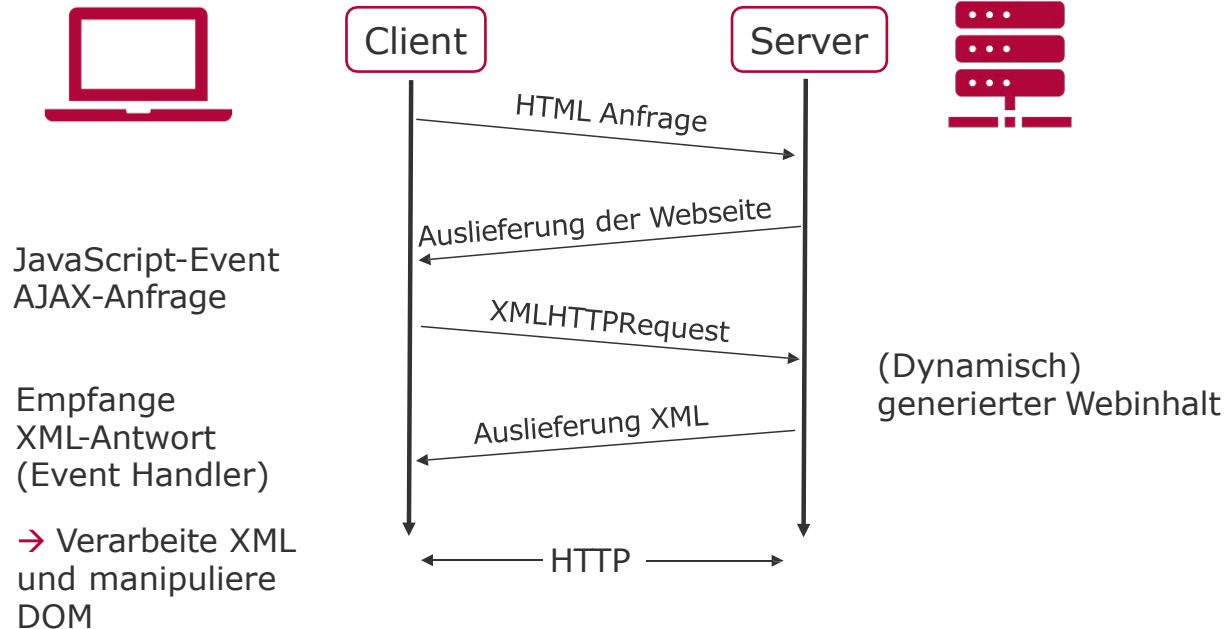
Allgemeines Vorgehen

- Mittels JavaScript wird ein XMLHttpRequest ausgeführt
- Anfragen werden asynchron im Hintergrund verarbeitet
 - Daten werden über einen HTTP-Request nachgeladen ohne die Nutzerinteraktionen mit der Webseite zu beeinträchtigen
 - Nach erfolgreicher Übertragung des XMLHttpRequests wird ein JavaScript-Event ausgelöst
 - Dieses Event kann genutzt werden um Folgeaktionen auszulösen
- Gebräuchliche Datenformate sind XML, JSON oder HTML
- XML/JSON Daten werden durch JavaScript geparkt und durch DOM-Manipulation als zusätzliche Daten in die Webseite eingebettet
- HTML kann bei Bedarf direkt durch DOM-Manipulation in die bestehende Webseite eingefügt werden

AJAX – Asynchronous JavaScript And XML



AJAX Request



AJAX – Asynchronous JavaScript And XML



Vorteile

- Serverlast und Bandbreitennutzung kann reduziert werden
 - Webseite wird nicht bei jeder Veränderung komplett neu angefragt
 - es werden nur bestimmte Bereiche der Webseite aktualisiert
- Schnellere Rückmeldung an den Nutzer
 - verbesserte Nutzbarkeit
- Erstellung von interaktiven Oberflächen möglich
 - Grenze zwischen Web- und Desktop-Anwendung verschwimmt
- Zahlreiche Frameworks/Bibliotheken sind für schnelle und effektive AJAX Programmierung verfügbar, z.B.
 - jQuery
 - React
 - Vue.js

AJAX – Asynchronous JavaScript And XML



Grenzen und Nachteile:

- JavaScript muss im Browser des Nutzers aktiviert sein, Verschlechterung der Barrierefreiheit
- Trennung zwischen Struktur und Layout der Seite verschwimmt
- Web-Crawler von Suchmaschinen führen keinen JavaScript-Code aus
→ Inhalte können nicht indiziert / gefunden werden
- AJAX-Anfragen erscheinen nicht im Browser-Verlauf, Funktionalität des Zurück-Buttons ist aufgehoben

AJAX – Beispiele



Pures JavaScript, auch als Vanilla JavaScript bezeichnet

```
1  var xhttp = new XMLHttpRequest(); // Neuer XMLHttpRequest
2  xhttp.onload = function() {
3    |   console.log(this.responseText); // Ausgabe des Inhalts der JSON-Datei
4  }
5  xhttp.open("GET", "info.json");    // Spezifikation von Anfrage-Type und URL
6  xhttp.send();                     // Absenden des XMLHttpRequests
```

jQuery

```
1  $.get("info.json", function(data, status){
2    |   console.log(data);
3  });
```

JavaScript in modernen Browsern mittels Web-Fetch-API

(ohne direkte Verwendung von XMLHttpRequest)

```
1  fetch("info.json")
2  .then(function(response) { return response.json(); })
3  .then(function(json) { console.log(json); });
```


JSON



JSON – JavaScript **O**bject **N**otation (RFC 4627, ersetzt durch RFC 8259)

- Leichtgewichtiges Datenaustausch-Format als Alternative zu XML
- **Ziel:** Lesbares Format für Mensch und Maschine
- Unterstützt diverse Datenstrukturen, die kompatibel mit den meisten Programmiersprachen sind, insbesondere
 - Arrays: Geordnete Liste von Werten
 - Objekte: Sammlung von Name/Werte-Paaren
 - auch bekannt als struct, dictionary, associative array

JSON vs XML



JSON

```
{
  "course": {
    "title": "Web-Technologien",
    "sections": [{
      "section": {
        "title": "Einheit 6",
        "description":
          "Serverseitige Prog.",
        "items": [{
          "item": {
            "type": "head",
            "title": "JSON vs XML"
          }
        }]
      }
    }]
  }
}
```

XML

```
<?xml version='1.0'?>
<course>
  <title>Web-Technologien</title>
  <sections>
    <section>
      <title>Einheit 6</title>
      <description>Serverseitige
        Prog.</description>
      <items>
        <item type="head">
          <title>JSON vs XML</title>
        </item>
      </items>
    </section>
  </sections>
</course>
```

JSON vs XML



JSON

- Datenformat
- geringerer Speicheraufwand
- leicht für Mensch und Maschine zu lesen
- Bestandteil von JavaScript

XML

- Markup-Sprache (Baumstruktur)
- Eingebaute Unterstützung von Schemata und Validierung
 - selbstbeschreibend
- Unterstützung von Kommentaren, Metadaten und Namespaces
- muss mittels separatem Parser verarbeitet werden

Polling und Long Polling



Grenze von AJAX

- Neue Daten müssen immer *aktiv vom Client* angefragt werden
 - Request-Response-Prinzip
 - keine Möglichkeit für den Server neue Daten an den Client zu geben
- **Polling** als mögliche Lösung
 - Client fragt regelmäßig beim Server nach neuen Daten an
 - erweiterte Variante **Long Polling**: Server antwortet auf die Anfrage des Clients erst, wenn neue Daten vorliegen (oder mit einem Timeout)
 - mittels XMLHttpRequests mit allen gängigen Browsern möglich
 - **Aber**: Deutlich mehr Last auf dem Server durch ständige Anfragen

WebSockets



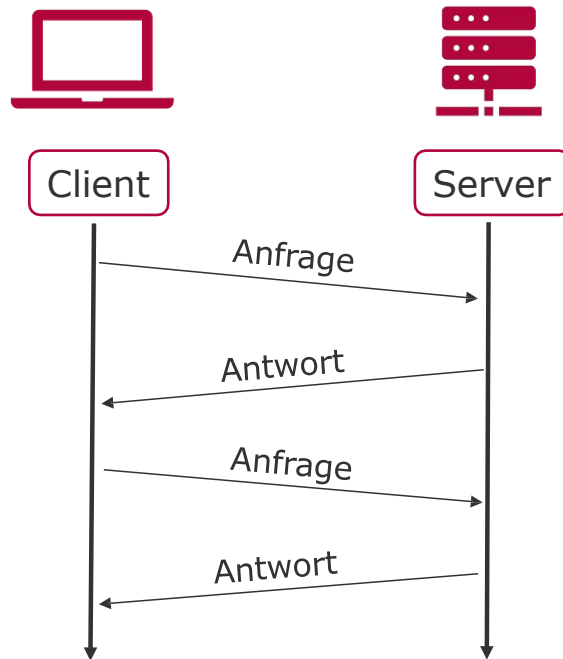
WebSockets (RFC 6455)

- standardisierte Erweiterung zu HTTP
- **Aufbau und Erhalt** einer **einzigen** Verbindung zwischen Client und Server
 - Daten werden mittels **Nachrichten** ausgetauscht
 - **beide Seiten** können Nachrichten senden (bidirektional)
 - beide Seiten können **gleichzeitig** Nachrichten senden (Vollduplex), geeignet z.B. für
 - Spiele, Chat-Anwendungen, Anwendungen mit geringer Latenz
- Funktioniert ohne XMLHttpRequests
 - Header werden nicht jedes Mal – sondern nur einmal – gesendet
 - Gesamtdaten, die zum Server gesendet werden, sind reduziert
 - stabile Verbindung bleibt (über längeren Zeitraum) geöffnet

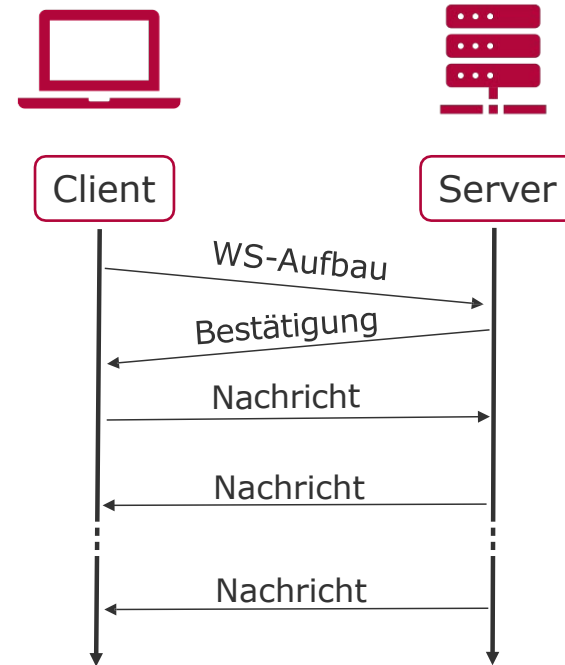
WebSockets



AJAX Request



WebSocket



WebSockets



Unterstützung durch nahezu **alle moderne Browser**

IE	Edge [*]	Firefox	Chrome	Safari	Opera	Safari on iOS [*]	Opera Mini [*]	Android Browser [*]	Opera Mobile [*]
		2-3.6		3.1-4					
		¹ 4-5	¹ 4-14	¹ 5-5.1	10.1	3.2-4.1			
6-9		² 6-10	² 15	² 6-6.1	¹ 11.5	¹ 4.2-5.1		2.1-4.3	¹ 12
10	12-91	11-90	16-91	7-14	12.1-77	6-14.4		4.4-4.4.4	12.1
11	92	91	92	14.1	78	14.7	all	92	64
		92-93	93-95	15-TP					

<https://caniuse.com/?search=websockets>

Und von den **gängigen Servern**, z.B.

- Apache, Nginx, Internet Information Server

WebSockets



Nachteile von WebSockets

- Kein automatisches Caching
- Zusätzliches Protokoll neben HTTP (Implementierungsaufwand)
- Keine automatische Wiederherstellung von Verbindungen
 - manuelle Implementierung erforderlich

Alternative zu WebSockets

- Server-Sent Events
 - Nachrichten können nur vom Server zum Client gesendet werden
 - Event-Loop auf der Client-Seite erforderlich, um Nachrichten zu empfangen
 - Browser-Unterstützung für WebSockets ist jedoch größer

WebSockets



Beispiel

```
1 var socket = new WebSocket("ws://localhost:8000");
2 socket.onopen = function(event) { // Wenn eine neue Verbindung aufgebaut wird
3     socket.send("Hello World"); // Senden der Nachricht "Hello World"
4 };
5 socket.onmessage = function(event) { // Wenn eine Nachricht empfangen wird
6     console.log(event.data); // Ausgabe der Nachricht
7 };
```

- JS-Frameworks, wie Socket.IO, setzen auf Standard-WebSockets auf und erweitern diese um Namespaces, Broadcast-Nachrichten, Automatische Wiederherstellung von Verbindungen, ...
- Nutzung in Kombination mit serverseitigen Frameworks

Entwicklung komplexer Webanwendungen



Problem:

- Häufig sind Dienste auf die Lösung **einer** Aufgabe spezialisiert
- Zur Realisierung von Webanwendungen können externe Abhängigkeiten (andere Webanwendungen) genutzt werden, welche ...
 - Schnittstellen und Verantwortlichkeiten definieren
 - Kommunikationsprotokollen spezifizieren



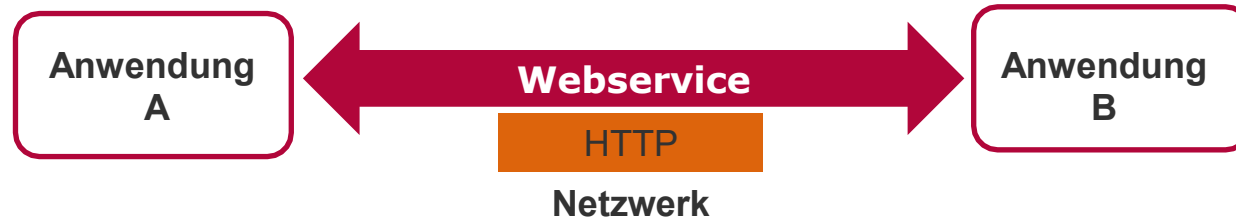
Machine-to-Machine-Kommunikation ist notwendig

Webservices



Webservices sind ein allgemeiner Ansatz für Entwicklung von verteilten Anwendungen im Web

- Zugriff auf Dienste verschiedener Anbieter
 - unabhängig von Betriebssystemen, Programmiersprachen und binären Übertragungsprotokollen
- Ermöglichen Kommunikation zwischen verteilten Anwendungen



Webservices und Schnittstellen



API – Application Programming Interfaces

Die API eines Webdienstes kann von anderen Diensten, Web-Apps im Browser oder mobilen Apps genutzt werden

- Erleichtert Nutzung moderner Web-Frameworks mit dynamischen Inhalten
- Abgrenzung über klar definierte Schnittstelle
- Nutzung und Dokumentation ist entweder privat oder öffentlich

Beispiele: <https://public-api-lists.github.io/public-api-lists/>

Webservices – REST



REST – REpresentational State Transfer

- keine Technologie – sondern Architektur
 - **ROA** – Resource Oriented Architecture
 - benutzt HTTP Vokabular: GET, POST, PUT, ...
 - auszuführende Aktion werden direkt per HTTP definiert
 - Erweiterungsmöglichkeit über Nutzung weiterer HTTP Header
 - native Unterstützung von Caching über HTTP
- **RESTful API**, auch REST API
- Web-APIs in Einklang mit den Design-Prinzipien von REST
 - Nutzung von HTTP Methoden zum Abrufen und Manipulieren von Daten

Webservices – REST



REST - REpresentational State Transfer

- Web-Applikationen leben von **Web-Ressourcen** (erreichbar über ihre URL) und ihrer Ausgestaltung, z.B. User, Content, ...
- REST ermöglicht es, Web-Ressourcen direkt mit Hilfe von HTTP zu manipulieren und zu verknüpfen
 - HTTP *Pfad* bestimmt, *welche* Ressource manipuliert wird
 - HTTP *Methode* bestimmt, *wie* Ressource manipuliert wird:
 - POST – Ressource wird erzeugt (Create)
 - GET – Ressource wird gelesen (Read)
 - PUT – Ressource wird aktualisiert (Update)
 - DELETE – Ressource wird gelöscht (Delete)
- Im HTTP Body werden nur noch die eigentlichen Daten transportiert (POST/PUT)

Webservices – REST



REST Beispiel:

■ Anfrage:

`DELETE /shoppingcart/items/244 HTTP/1.1`
`Host: shop.example.com`

**Auszuführende Aktion
wird direkt mittels der
HTTP-Methode definiert**

■ Antwort:

`HTTP/1.1 204 No Content`
`Date: Wed, 29 Sep 2021 00:00:01 GMT`

**Adressat der Aktion
wird im Pfad
definiert**

Webservices – REST



REST – Design Prinzipien

- Immer passende HTTP Methode (GET, POST, PUT oder DELETE) verwenden
- Kein Erinnerungsvermögen (Statelessness) – alle benötigten Daten müssen mit Request gesendet werden
- URLs orientieren sich an Dateistrukturen
- Service registriert den MIME-Type des Requests und sendet Response im passenden MIME-Format zurück

Idempotenz

- Effekte der Ausführung sind unabhängig von der Wiederholungen einer Anfrage gleich
 - GET, PUT und DELETE sollten idempotent sein
 - POST-Methode im Regelfall nicht idempotent, da für eine Ausführung eine neue Ressource angelegt werden könnte

MVC – Model-View-Controller

Motivation



(Web)-Anwendungen können aus verschiedenen Komponenten bestehen:

- Visuelle Darstellung
- Anwendungslogik (Domainwissen)
- Datenhaltung
- Programmiercode, der alle Teile miteinander verbindet
- ...

Die zunehmende Größe einer (Web)-Anwendung erschwert die Entwicklung:

- steigende Komplexität und schwindende Übersichtlichkeit
- Daten und Quellcode müssen gut strukturiert werden
- Bedarf nach klaren Abgrenzungen und Wiederverwendbarkeit des Quellcodes
- ...

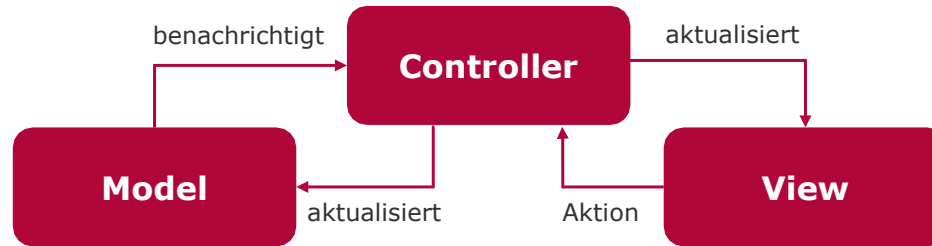
MVC – Model-View-Controller

Grundidee



Trennung nach

- Datenhaltung und Businesslogik (*Model*)
- Kontrolllogik der Applikation (*Controller*)
- Darstellung (*View*)



- Alle Quellcode-Bestandteile werden entsprechend ihrer Funktion eingeteilt
 - Bildung von getrennten Verantwortlichkeiten

MVC – Model-View-Controller

Ziele und Verwendung



Ziele: Bessere Abgrenzung und Wiederverwendbarkeit einzelner Teile



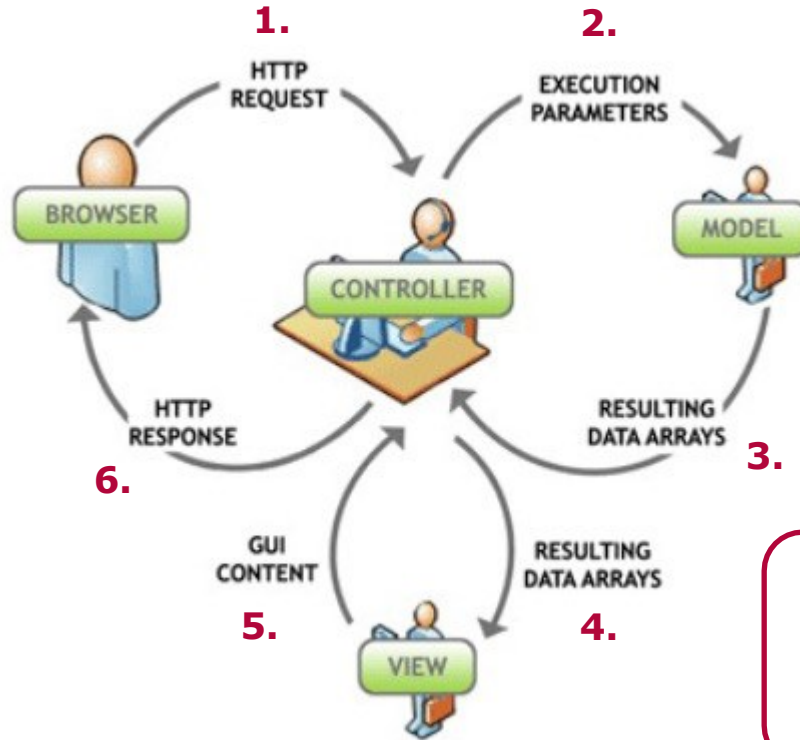
Controller verbindet/koordiniert zwischen *Model* und *View*

- Änderungen in einer Schicht unabhängig von anderen Schichten, z.B.
 - Änderung in der Darstellung (*View*) lässt Controller unangetastet
- Der *Controller* sollte möglichst schlank sein, das Model eher umfangreich
 - auch Code-Auslagerungen in weitere *Controller* ist möglich

Verwendung:

- Beliebtes und simples Software-Design-Muster
- Genutzt in fast allen Web-Frameworks

MVC – Model-View-Controller



Das MVC-Design-Muster kann sowohl clientseitig als auch serverseitig umgesetzt werden

Frameworks: Zu lösendes Problem



Problem

- Umsetzung von MVC-Entwurfsmustern beim Web-Programmieren ist oft ganz ähnlich in verschiedenen Projekten
- Daher häufige Code-Doppelungen und -Wiederholungen

Frameworks als Lösungsansatz („framework“: englisch für *Gerüst*)

- Weitgehende Aufteilung von Verantwortlichkeiten in einem Programm
- Verfügbar in verschiedenen Komplexitätsstufen, von recht einfachen, kleinen Frameworks bis hin zu ausgereiften Frameworks mit vielen Zusatzfunktionen
- Stellen Lösungen bereit für Probleme, die immer wieder gelöst werden müssen
- Enthalten Bibliotheken für typische Anwendungsfälle
- Erlauben beschleunigtes Programmieren
 - Code-Generatoren, Konventionen ...

Frameworks: Vorteile



- Frameworks ermöglichen **einfachere, schnellere** Web-Programmierung
- Bauen auf Erfahrung und Tools anderer auf
- Bieten Lösungen für häufig vorkommende Problemstellungen
 - meist sehr effizient, stabil und **sicher**
- Architektur wird vorgegeben, etwa MVC-Entwurfsmuster
- **Open-Source:** Die meisten Frameworks sind frei verfügbar
 - viele Augen sehen mehr Fehler
 - Verbesserungen sind gern gesehen

Frameworks: Grenzen und Nachteile



- **Frameworks** sind nicht für die Ewigkeit und **können sich ändern**
 - Änderungen können grundlegende Veränderungen des eigenen Codes erzwingen
- Eigener Programmcode oft sehr eng an den Code des Frameworks **gekoppelt**
- Frameworks lösen viele Probleme, aber nicht alle
 - wer außerhalb der vorgegebenen Grenzen gerät, muss oft „gegen das Framework kämpfen“
- Frameworks und „**Magie**“:
 - ohne Kenntnis der Konventionen des Frameworks ist Code oft schwer nachvollziehbar (insbesondere für Neueinsteiger)
 - Was kann die Programmiersprache? Was ist vom Framework?

Frameworks: Empfehlungen



- Erst Programmiersprache lernen, dann ein Framework!
 - Magie des Frameworks von Funktionen der Sprache unterscheiden
 - **Probleme kennen, die das Framework löst, um dessen Lösungen wertschätzen zu können**
- Frameworks garantieren keinen sauberen Code
 - Sinn und Zweck der Architekturmuster kennen
- **Vor- und Nachteile abwägen**
 - Frameworks beschleunigen die Entwicklung (insb. zu Beginn), aber erschweren Quereinstieg und Umstieg
 - Entscheidung sehr lange Zeit gültig: Benutzt man ein Framework, wird man es nur schwer wieder los

Frameworks



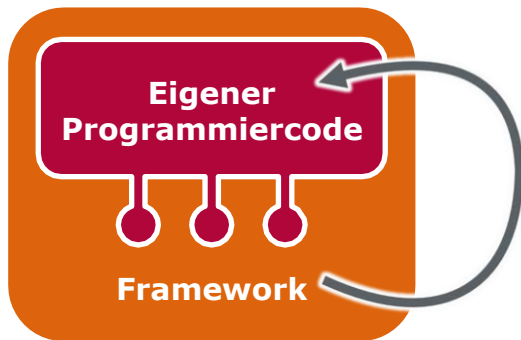
- **Open-Source-Software:** Freie Verwendung unter durch **Lizenzen** geregelten Bedingungen
- **Flexible Lizenzen**
 - **Beispiele:** MIT, BSD, Apache
 - normalerweise untereinander kompatibel
- **Restriktive Lizenzen**
 - **Beispiel:** GPL
 - abgeleitete Software muss ebenfalls unter GPL lizenziert werden
 - Verwendung von GPL-Bibliotheken kann dazu führen, dass eigener Code veröffentlicht werden muss
 - **AGPL:** Schließt Schlupfloch für serverseitige Applikationen
- Mehr Details auf: <https://choosealicense.com/licenses/>

Frameworks und Bibliotheken – Unterschiede: Frameworks



Was ist ein **Framework**?

- bietet Gerüst zur Implementierung der spezifischen Anwendung
 - abstrahiert grundlegende Funktionalitäten
 - konkrete Funktionen werden in eigentlicher Anwendung implementiert
 - gibt meist eine Architektur, wie etwa MVC-Design-Muster vor
 - kann bereits mit Bibliotheken vorkonfiguriert sein
- Eigener Programmiercode „reagiert“ auf Aufrufe aus dem Framework



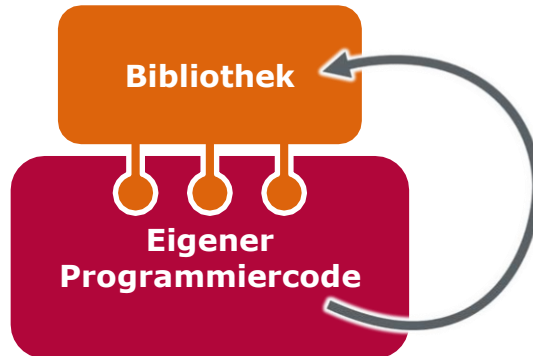
Framework ruft Funktionen aus
der Anwendung auf

Frameworks und Bibliotheken – Unterschiede: Bibliotheken



Was ist eine **Bibliothek**?

- Bibliothek ist eine Sammlung von Funktionen, z. B.
 - String-/Datenmanipulation
 - Visualisierung
 - Kompression
 - Validierung regulärer Ausdrücke
- Anwendung nutzt Ergebnisse der Bibliotheksaufrufe



Die Anwendung ruft Funktionen aus der Bibliothek auf

Serverseitige Web-Programmierung



Dynamische Erstellung von Webseiten auf der Serverseite

- Statt statische HTML-Dateien auszuliefern, können HTTP-Server die dynamische Erstellung einer Website **Anwendungsprogrammen** auf **Anwendungsservern** überlassen, z.B.
 - Suchmaschine, E-Shop, Social Media, ...
- Die von der Anwendung nach Übergabe der übermittelten Nutzerdaten erstellte HTML-Seite wird dann über den HTTP-Server an WWW-Client ausgeliefert
- Anwendung verarbeitet und nutzt:
 - Anfragedaten (wie GET- oder POST-Parameter, Content Negotiation)
 - Sessioninformationen (Benutzer, Berechtigungen, Präferenzen)
 - serverseitig gespeicherte Daten
 - anderen Webservices (über API-Anfragen)

Serverseitige Web-Programmierung

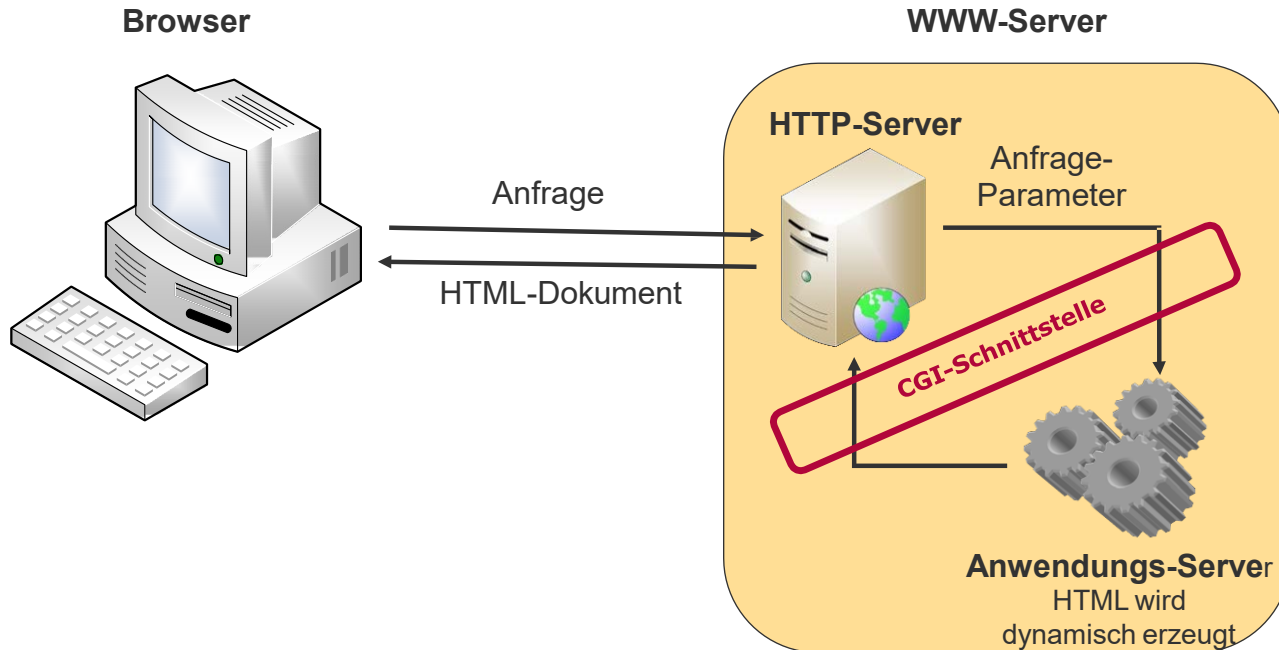


- Zur Übergabe der Nutzerdaten bzw. der erstellten Webseite wird standardisierte Schnittstelle zwischen HTTP-Server und Anwendungsprogrammen auf Serverseite gebraucht:
→ **CGI - Common Gateway Interface**
- Anwendungsprogramme können mit beliebigen Programmiersprachen erstellt werden, z.B.
 - Java: Servlets, Server Pages, Beans, ...
 - Skriptsprachen: ASP, PHP, Python, Ruby, JavaScript, ...
- Web-Frameworks liefern Grundgerüst und erleichtern Bereitstellung üblicher Komponenten solchen Anwendungen, z.B.
 - Ruby on Rails, Django, ...

Serverseitige Web-Programmierung



Dynamische Erstellung von Webseiten auf der Serverseite



Server- vs. clientseitige Web-Programmierung



Serverseitige Web-Programmierung ...

- wird für Synchronisation von Daten zu mehreren Geräten benötigt
- ermöglicht verlässliche Validierung von Daten
 - Clientseitige erzeugte Website kann nur von Nutzer verändert werden

Clientseitige Web-Programmierung ...

- verbessert Benutzerfreundlichkeit und spart HTTP-Requests
- ermöglicht interaktive Webseiten

Serverseitige Web-Anwendungen benötigten mehr Ressourcen des Anbieters

- Verlagerung durch clientseitige Web-Programmierung auf WWW-Client
- Beide Varianten oft in Kombination eingesetzt

Web-Frameworks



Entwicklung serverseitiger Web-Programme

- **CGI:** HTML-Dokument wird vom CGI-Programm generiert, z.B.

```
1 print("<h1>Titel</h1>");
```

- **Serverseitiges Scripting:** Spezielles Skript-Markup innerhalb von (sonst statischen) HTML-Dateien z.B.

```
1 <ul>
2   <?php foreach ($items as $item): ?>
3     <li><?php echo $item->description ?></li>
4   <?php endforeach ?>
5 </ul>
```

- **Problem:** Mischung verschiedener Aufgabenbereiche in der selben Datei
 - unübersichtlicher schwer wartbarer Code
 - Designer und Programmierer arbeiten an derselben Datei → Konfliktpotential

Web-Frameworks



Web Frameworks, seltener Web Application Frameworks ...

- Bieten Routing Pfade, die mit dem verarbeitenden *Controller* verbinden, z.B.
 - `/courses/webtech2023` → **CoursesController** mit ID `webtech2023`
- Abstrahieren häufig verwendete Operationen, wie etwa
 - Session Management mit Cookies
 - Verwaltung von REST-Routen
- Bieten Persistenzschicht für den Datenbankzugriff und für die Abbildung der Anwendungslogik
- Enthalten Template-Engines für die Generierung von HTML
- Integrieren grundlegenden Schutzmaßnahmen gegen Angriffe im Web

Web-Frameworks



Web-Frameworks ...

- Bieten auch native Unterstützung von
 - HTTP-Methoden
 - Request/Response-Zyklen
 - Header-Verarbeitung
 - Middleware
 - ...
- Können erweitert werden durch zusätzliche Bibliotheken für
 - Mail-Versand
 - Caching
 - Verarbeitung von langwierigen Prozessen im Hintergrund (Queues)
 - ...

Ausgewählte Web-Frameworks



Populäre Web-Frameworks

- Python: Django, Flask, Tornado
- Ruby: Ruby on Rails, Sinatra
- PHP: Symfony, Laravel, CakePHP
- Java: Grails, Play, Spring
- .NET: ASP.NET

Persistenz im Web



HTTP ist zustandsloses Protokoll, also ohne Gedächtnis

Aber: Applikationen befinden sich immer in einem bestimmten Zustand

→ Zustände müssen erinnert, beschreibende Daten vorgehalten werden

■ **Clientseitig:** Speicher-Mechanismen beim Browser oder im Arbeitsspeicher

- Vorteile: keine Anforderungen an den Server und geringe Latenz
- Nachteile: zwischen Geräten keine Persistenz oder Synchronisierung

■ **Serverseitig:** Datenbanken und Sessions

- Vorteile: dauerhafte Speicherung
 - Nachteile: effiziente Verwaltung großer Datenmengen erforderlich
- Nutzung eines Datenbanksystems

■ Zustand wird bei Bedarf über Cookies und REST-Anfragen synchronisiert

WebStorage



Erlaubt **clientseitige** Datenpersistenz

- WebStorage umfasst

- localStorage

- 5-10 MB, unbegrenzt gültig, in allen Fenstern/Tabs gültig, wird von JS oder bei leeren des Cache gelöscht

- sessionStorage

- 5-10MB, gültig bis Seite geschlossen wird, in aktuellem Fenster/Tab gültig, wird bei Schließen des Fensters/Tabs gelöscht

- Cookies

- 4KB pro Cookie, unbegrenzt gültig, in allen Fenstern/Tabs gültig, Haltbarkeit wird bei Erzeugung festgelegt

```
// Store
localStorage.setItem("lastname", "Smith");
// Retrieve
document.getElementById("result").innerHTML =
localStorage.getItem("lastname");
// Remove
localStorage.removeItem("lastname");
```

```
document.cookie = "username=John Doe;
expires=Thu, 18 Dec 2013 12:00:00 UTC;
path="/";
```

IndexedDB



Erlaubt **clientseitige** Datenpersistenz

- IndexedDB bringt mächtige Zusatzfunktionalitäten mit sich
 - Beinahe unbegrenzter Speicher
 - Dedizierte Datentypen „Date“ und „Number“ (neben „String“)
 - Unterstützt Service Worker (asynchroner Datenzugriff) und Suchfunktionen
- Nachteile:
 - API weniger intuitiv (Eventhandling hochrelevant für Datenkonsistenz!)
 - Langsamere Datenzugriff

Einsatz von Datenbanken



Ziel: Speicherung und Abfrage von Datensätzen und Informationen

- Spezielle Schnittstelle abstrahiert von zugrundeliegender Persistenz
- Garantierte Konsistenz (Änderungen werden ganz oder gar nicht gespeichert)
- Abruf der gespeicherten Information über APIs oder Abfragesprache
- **Datenbank**, auch Datenbankmanagementsystem (DBMS) genannt

Unterschiedliche Arten und Ansätze von Datenbanken:

- Relationale Datenbanken, z.B. MySQL, PostgreSQL, Oracle, MS SQL, ...
 - bieten **Structured Query Language (SQL)** als Abfragesprache an
- NoSQL-Datenbanken
 - Graph Stores, z.B. Neo4J, Giraph, ...
 - Document Stores, z.B. MongoDB, Elastic, OrientDB, CouchDB, ...
 - Key-Value Stores, z.B. Redis, BerkeleyDB, ...
 - Column Stores, z.B. Hbase, Cassandra, Hadoop, ...

Relationale Datenbanken



■ Tabellenstruktur

- Reihe – Repräsentiert einen Datensatz (z.B. User)
- Spalte – ein Attribut der Datensätze, z.B. Name, Vorname, Email, ...
- Tabelle – Menge von Datensätzen mit denselben Attributen

■ Jeder Datensatz in Tabelle hat eindeutigen Schlüssel: **Primary Key**

- ermöglicht eindeutige Identifikation des Datensatzes
- kann aus mehreren Attributen zusammengesetzt werden
- kann in anderen Tabellen referenziert werden, um Bezüge zwischen Tabellen herzustellen (Foreign Key)

■ Beispiel mit SQL:

```
SELECT first_name, last_name, email FROM users
WHERE city = 'SALZBURG'
ORDER BY given_name ASC;
```

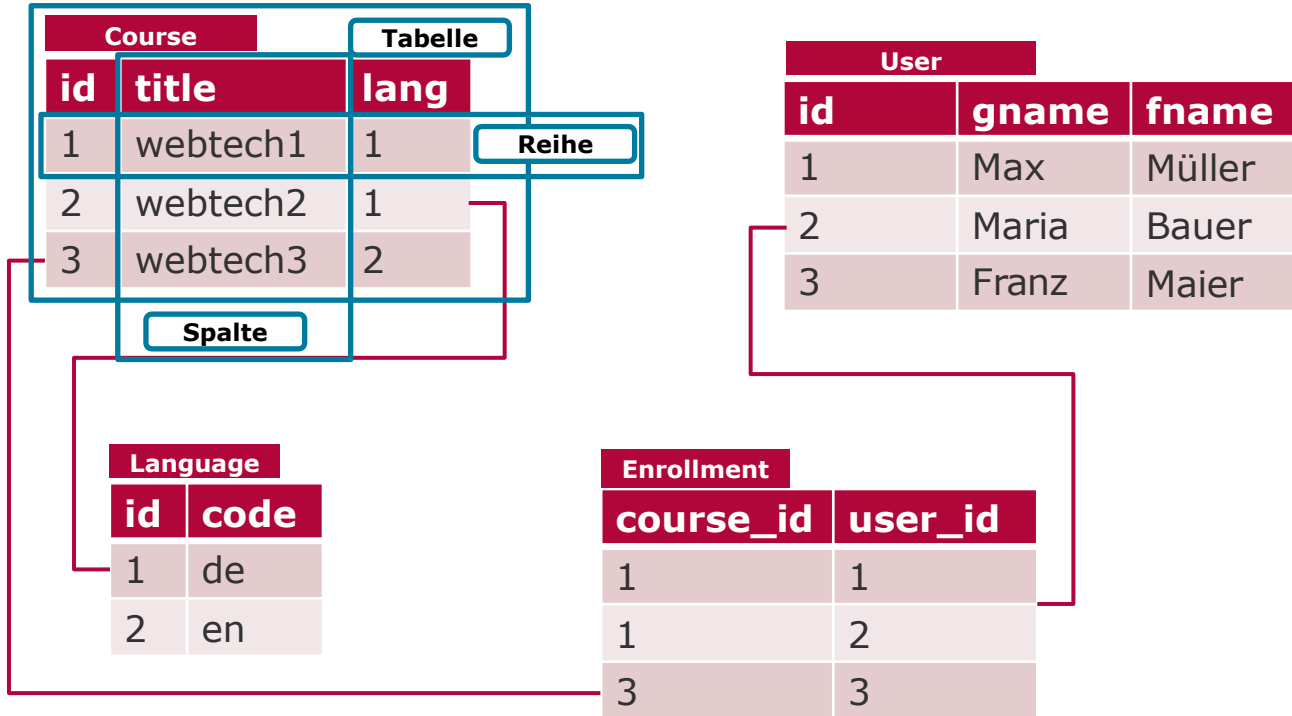
Spalten

Tabelle

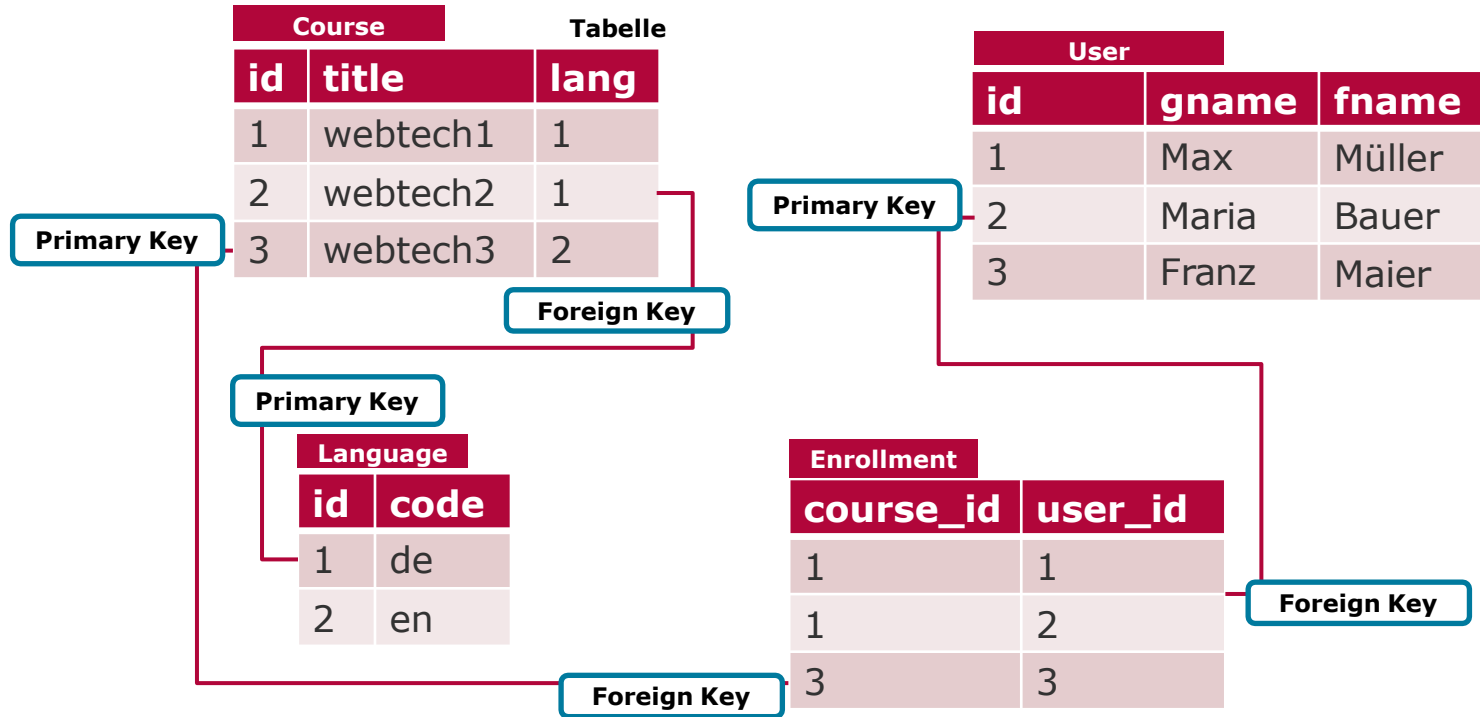
Filter

Alphabetische Sortierung

Relationale Datenbanken



Relationale Datenbanken



Relationale Datenbanken



Course		Tabelle
id	title	lang
1	webtech1	1
2	webtech2	1
3	webtech3	2

User		
id	gname	fname
1	Max	Müller
2	Maria	Bauer
3	Franz	Maier

Language	
id	code
1	de
2	en

Enrollment	
course_id	user_id
1	1
1	2
3	3
2	1

Neue Belegung

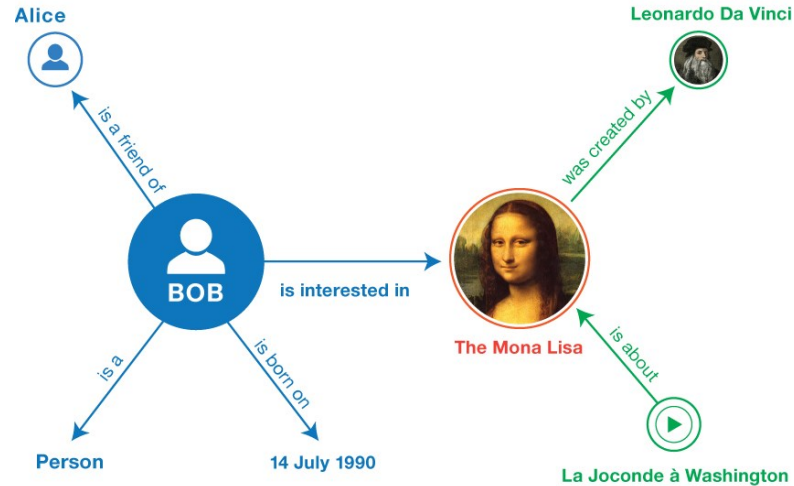
```
INSERT INTO enrollment VALUES (2, 1);
```

NoSQL-Datenbanken

Graph Stores / Graph-Datenbanken



- Basieren auf **Graph-Struktur** → Knoten (Nodes), Kanten (Edges)



Quelle: <https://www.w3.org/TR/rdf11-primer/example-graph.jpg>

- **Query-Sprachen** (ausgelegt auf Graphen): Gremlin, SPARQL, Cypher
- **Anwendung:** Soziale Netze, Semantic Web, etc.

NoSQL-Datenbanken

Document Stores



- Speicherung komplexer Daten, Arrays, Objekte, Key-Value-Paare
- Speicherung der Daten häufig im JSON-Format anstelle von Relationen

Dokument

```
{ "user":  
  {  
    "id": "aaaaaaa-bbbb-cccc-dddd-eeeeeeee",  
    "personal_data": {  
      "last_name": "Müller",  
      "first_name": "Max",  
      "email": "max.mueller@example.com"  
    },  
    "courses": [  
      "webtech1",  
      "webtech2"  
    ]  
  }  
}
```

Abfrage des Vornamens (JSON Pointer)

← /user/personal_data/first_name

- **Query-Sprache:** JSON Pointer (RFC 6901) oder objektorientierte APIs
- **Weitere Alternative:** XML-Datenbanken – **Query-Sprache:** XPath

Datenbanken im Vergleich



Unterschiedliche Datenbanksysteme für unterschiedliche Einsatzzwecke

- Relationale Datenbanken bieten ...
 - + leichtere Sicherstellung der Konsistenz
 - + erzwungene Einhaltung eines Schemas
 - + mächtige Abfragesprache
- NoSQL-Datenbanken bieten ...
 - + höhere Flexibilität in der Datenspeicherung
 - + einfachere horizontale Skalierbarkeit
 - + vielzählige Varianten für spezialisierte Einsatzszenarien



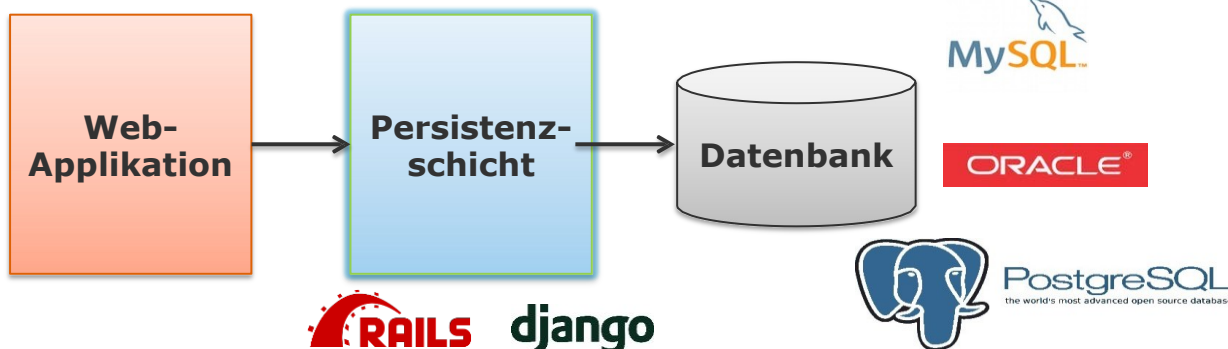
→ **Empfehlung:** Nutzen Sie das zum Zweck der Anwendung passende Werkzeug!

Persistenzschicht



Persistenzschicht

- Abstraktionsebene zwischen Datenbank und Applikation
- **Gründe:**
 - Vendor-Lock-In vermeiden: Abstraktion erleichtert Austausch der Datenbank (Maintainability)
 - Schwieriger: Umstieg von einem DB-Typ auf einen anderen
 - Komfortablerer Zugriff auf Daten für Entwickler (Produktivität)
 - Daten-Zugriffsoptimierung (Performance)



Persistenzschicht



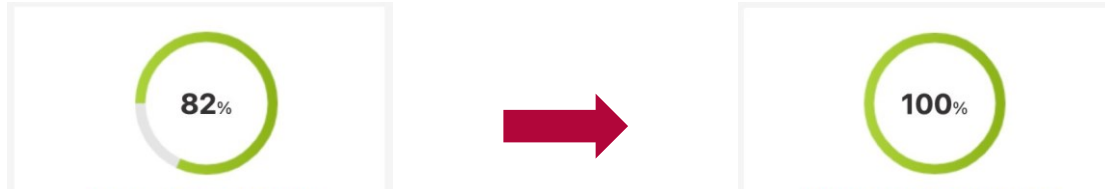
Moderne Web-Applikationen werden in **objektorientierten** Sprachen entwickelt

- Anwendung von Konzepten der objektorientierten Programmierung
 - Kapselung & Wiederverwendbarkeit & Abstraktion
- **Object-Relational-Mapping – ORM:**
 - Abbildung von Objekten auf Datensätze
 - ORM-Tools gibt es für alle verbreiteten Programmiersprachen – oft integriert in Persistenzschicht der Frameworks
 - **Vorteile:**
 - Unabhängigkeit von konkreter Abfragesprache
 - Datenbankverbindung wird durch ORM verwaltet
 - **Nachteile:**
 - Kompliziertere Abfragen können die Performance beeinträchtigen

Persistenzschicht



Beispiel mit ActiveRecord (Bestandteil von Ruby on Rails):



- Speichern eines neuen Prüfungsergebnisses:

```
result = ExamSubmission.create(user: current_user,  
                                exam_id: 1, score: 0.82)
```

Objekt

Variable

- Aktualisieren des vorhandenen Ergebnisses:

```
result.update(score: 1.0)
```

Aufruf der Funktion **update** auf dem Objekt **result**

- Übersetzung zu SQL im Hintergrund:

```
UPDATE exam_submission SET score = 1.0  
WHERE user_id = 123 AND quiz_id = 1;
```



FH Salzburg

VO Web-Technologien

Einheit 6, Oliver Jung