**Web-Grundlagen**

**Basistechnologien:**

* **HTML** (Hypertext Markup Language): definiert Grundstruktur
* **CSS** (Cascading Style Sheets): legt Aussehen fest
* **JavaScript**: Ermöglicht Interaktion mit Benutzer

**Responsive Webdesign:**Webseiten Layout und Interaktion wird auf das Endgerät angepasst.

**Promise Objekt:**Das Promise-Objekt wird **für asynchrone Ausführung von Code** verwendet.  
Es kann sich **in einem von drei Zuständen** befinden:

* **pending**: initial, bis resolved/rejected
* **resolved**: Operation erfüllt
* **rejected**: Operation gescheitert

**Session:**Als Session (dt. Sitzung) bezeichnet man eine **stehende Client- / Server-Verbindung**   
Eine Session beginnt üblicherweise mit Login und endet mit Logout   
Anonyme Sitzungen auch ohne explizites Login möglich

HTTP ist ein zustandsloses Protokoll (jede Anfrage ist in sich abgeschlossen). Für weitere Interaktion müssen Daten hinterlegt werden.

**Cookies:**

* Cookies ermöglichen das **Speichern von Key/Value Paaren**.
* Sie werden **von der Domäne** **gesetzt** ([www.htlwrn.ac.at](http://www.htlwrn.ac.at)). Bei Anfragen an diese Domäne werden **alle zugeordneten Cookies automatisch mitgesendet**.
* Webserver kann aufgrund der mitgesendeten Cookies den **Benutzer wiedererkennen**.
* Sie haben ein **Ablaufdatum**, nachdem der Browser sie löscht.

**HTML5 Web Storage:**

* **Verbesserte Variante von Cookies** in HTML5.
* bietet **mehr Speicherplatz** als Cookies (> 5MB).
* Daten werden **nicht automatisch** bei jedem HTTP-Request **mitgesendet**.
* Daten können mittels JavaScript Aufruf an den Server gesendet werden oder lokal ohne Serverkommunikation verarbeitet werden.

**Verteilte Systeme**

Eine Sammlung von unabhängigen, vernetzen Computern die dem Benutzer wie ein einzelnes zusammenhängendes System erscheinen.

**Warum werden Systeme verteilt?**

* Ressourcen und Services werden mit Benutzern verbunden
* Verfügbarkeit und Ausfallsicherheit
* Performance

**Grundsätzlich gilt:** Nur verteilen, wenn notwendig (Programme werden viel komplexer, fehleranfälliger)

**Trugschlüsse Verteilter Systeme:**

* Das Netzwerk ist verlässlich
* Die Latenzzeit ist null
* Brandbreite ist unendlich
* Das Netzwerk ist sicher
* Die Topologie ändert sich nicht
* Es gibt einen Administrator
* Transportkosten sind null
* Das Netzwerk ist homogen

**Design-Ziele verteilter Systeme:**

* Ressourcen teilen
* Nebenläufigkeit
* Transparenz (Verstecken interner Strukturen und Komplexität)
* Offenheit (Services werden gemäß Standards angeboten)
* Skalierbarkeit
* Fehlertoleranz

**Softwarearchitekturen**

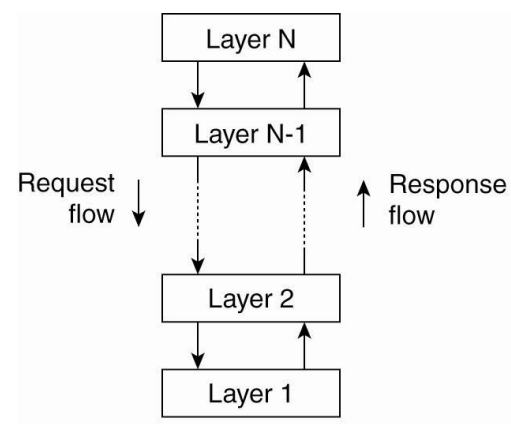
**Separation of Concerns:**

* Einteilung in Client, Server, Service
* Definieren von Schichten (Layering)
* Implementierung aufteilen auf Komponenten

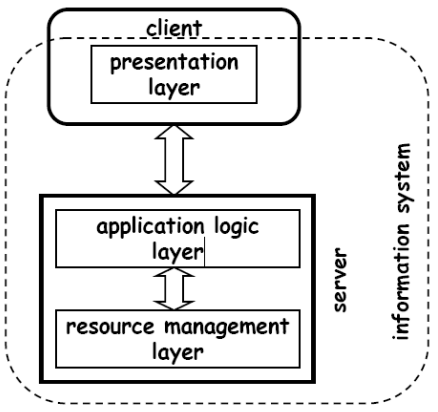
**Datenkapselung (Information hiding):**

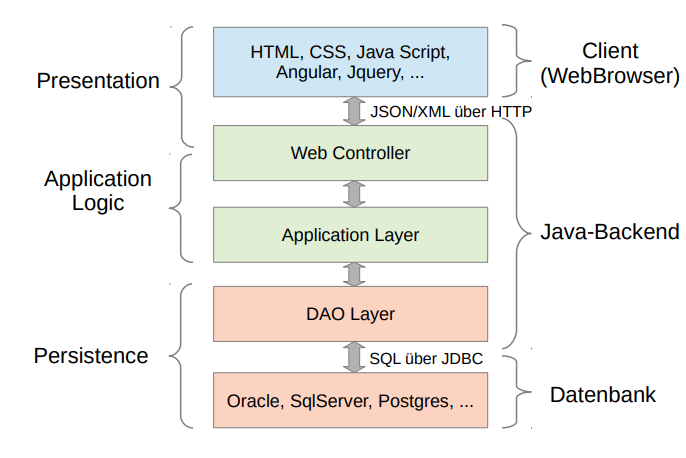
* Zugriff nur über Schnittstellen
* Direkter Zugriff auf Implementierung nicht möglich

**Schichtenarchitektur**



**2-Schichten-Architektur:**



**Schichten im Java-Web-Backend:**

**DAO Layer:**

* steht für Data Access Object
* trennt Application Layer von der Datenbank
* enthält Code für den Datenbank Zugriff

**Datenkapselung über Interfaces:  
Direkter Zugriff** auf Implementierungsklassen wird **vermieden**. **Zugriff erfolgt über Interfaces**.  
**Information Hiding:** Interne Schnittstellen bleiben für Aufrufer der Schnittstelle verborgen.  
**Vorteil:** Die DAO Implementierung kann ausgetauscht werden, ohne sich auf die Serviceschicht auszuwirken.

**Application Layer:**

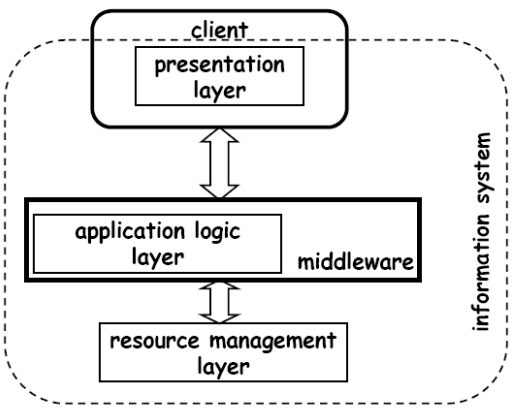
* Enthält **Geschäftslogik**
* Ist unabhängig davon, wo und wie die Daten gespeichert werden
* Ist unabhängig davon, wie die Daten dem Anwender präsentiert werden

**Web Controller:**

* Behandelt die **Kommunikation zwischen Web-Client und Java-Backend**
* Spezifische Schicht im Falle einer Web-App
* Gängige Variante ist REST-API

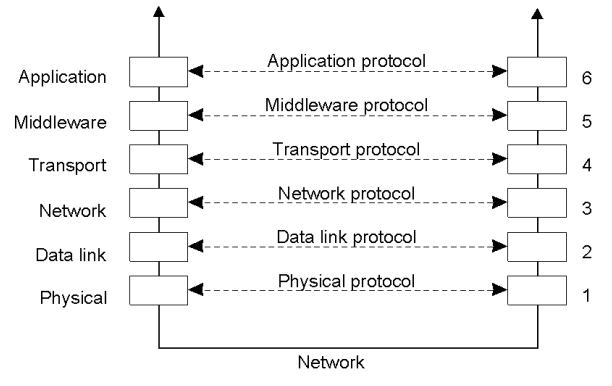
**Middleware und Frameworks**

Typische Geschäftsanwendungen bestehen zu 70-80% aus Applikationsinfrastruktur, die sich von Anwendung zu Anwendung kaum unterscheidet.   
Nur 20-30% Geschäftslogik die spezifisch ist für die jeweilige Anwendung.

**3-Schichten-Architektur mit Middleware:**  


**Middleware Services:**

* Bietet einfache Kommunikationsmöglichkeiten (z.B. Remote Procedure Call)
* Unterstützung für Nebenläufigkeit
* Erleichtert Skalierbarkeit
* Services für Datenbankzugriff
* Unterstützung für Replikation
* Naming Services zum Auffinden der Server
* Security: Authentifizierung, Berechtigungen, …

**Middleware Protokolle:**  


**Frameworks:**Programmiergerüst, dass einen **Rahmen** (engl. Frame) **zur Verfügung stellt**, innerhalb dessen der Programmierer eine Anwendung erstellt.

* Ähnlich wie Middleware sollen Frameworks dem Softwareentwickler das Leben erleichtern
* Häufige, in verschiedenen Anwendungen auftretende Probleme werden gelöst
* Entwickler muss nur die Teile implementieren, die spezifisch für seine Anwendung sind
* Frameworks ersparen Arbeit, geben dafür aber auch teilweise die Architektur der Anwendung **Unterschied zu Libraries:** Libraries können unabhängig von der Softwarearchitektur jederzeit aufgerufen werden

**Logging Frameworks:**

* Vorgänge einer laufenden Software sollen für spätere Auswertung protokolliert werden
* Mehrere Log-level (Error, Warning, Info, Debug, Trace)

**Dependency Injection:**

* Entfernt starre Abhängigkeiten im Programmcode
* Abhängigkeiten werden vom Framework zur Laufzeit erkannt und aufgelöst
* Es können sowohl einfache Datentypen als auch Objekte “injiziert” werden
* Über Konfigurationsdatei können die übergebenen Werte und Objekte definiert werden

**Spring Boot**

* Dependency Injection vieler Komponenten
* Vereinfacht Konfiguration von Applikationen
* Spring Boot sucht automatisch nach Klassen zur Auflösung von Abhängigkeiten

**Weitere Spring Frameworks:**

* Spring Data
* Spring Security
* Spring LDAP
* Spring Batch
* usw.

**Objektrelationale Mapper:**Frameworks zur Ablage von Objekten einer objektorientierten Programmiersprache in eine relationale Datenbank.  
Beispiel: Java Persistence API (JPA)

**HTTP**

HTTP ist ein **zustandsloses Protokoll** (jede Anfrage ist in sich abgeschlossen). Für weitere Interaktion müssen Daten hinterlegt werden.

**HTTP Anfragemethoden:**

* **GET**   
  Gebräuchlichste Methode. Lädt eine durch eine URI adressierte Ressource (z.B. HTML-Datei)
* **POST**  
  schickt Daten zur Verarbeitung zum Server
* **PUT**   
  Hochladen einer Ressource (z.B. Datei) unter Angabe einer Ziel-URI auf den Server
* **DELETE**  
  löscht die angegebene Ressource am Server
* **HEAD**Wie GET, liefert aber nur HTTP-Header (Zur schnellen Überprüfung einer Datei im Cache)
* **TRACE**liefert Anfrage zurück wie vom Server empfangen (Für Debugging von Verbindungen)
* **OPTIONS**liefert Liste, der vom Server unterstützen Methoden

**Uniform Resource Identifier (URI):**  
URI ist eine Zeichenfolge und dient der **eindeutigen Identifikation einer Ressource**.  
Bsp.: http://www.htlwrn.ac.at/images/stories/iconi.png)   
URIs werden **nicht nur von HTTP verwendet**.  
Bsp.: file:///home/sis/NVS/4Jg/Labs/Angabe\_Lab1.pdf

**GET-Anfrage:**

* Hauptbestandteil ist die URL
* Parameter können mittels „?“ angehängt werden

**GET-Antwort:**

* Header enthält Statuscode (z.B. „200 OK“) und Metadaten (z.B. Kodierung mit UTF-8)
* Body enthält die angeforderte Ressource

**HTTP Statuscodes:**

* **Jede HTTP-Anfrage** wird vom Server mit einem **HTTP-Statuscode** beantwortet
* Informiert, ob Anfrage erfolgreich war
* Im Fehlerfall gibt Code Fehlergrund an
* Im Fehlerfall ist der Body meistens leer

**HTTP Statuscode Klassen:**

* 1xx: Information
* 2xx: Erfolgreiche Operation
* 3xx: Umleitung
* 4xx: Client Fehler
* 5xx: Server Fehler

**Markup Languages**

**Extensible Markup Language (XML):**

XML ist eine Sprache zur Darstellung hierarchisch strukturierter Daten.

**Document Type Declarations (DTD):**

* Definiert die erlaubten Tags XML-Dokumente
* Legt fest, welche Tags verwendet werden dürfen und in welcher Reihenfolge
* XML-Dokument kann gegen DTD validiert werden

**XML-Schema:**

* Exaktere Beschreibung der Struktur von XML-Dokumenten
* Spezifiziert, welcher Datentyp der Inhalt eines Tags haben darf
* Konkretes XML-Schema wird als XML-Schema Definition (XSD) bezeichnet

**XML-Namespace:**ermöglicht **mischen von Tags aus verschiedenen XML-Schemata.**  
Zu Beginn werden Schema einem Präfix zugeordnet. Tags werden dann mit Präfix annotiert, um ihm dem jeweiligen Schema zuzuordnen.

**SOAP:**

* XML-gestütztes Netzwerkprotokoll zum Datenaustausch
* Ermöglich Remote Procedure Calls (RPC)
* Anfragen und Antworten mittels XML-Dokumenten

**XPATH:**

XPATH ist eine Abfragesprache für XML-Dokumente. Damit können alle Knoten im Element, die zur Abfrage passen ermittelt werden.

**Kritik an XML:**

* Schlecht als Austauschformat geeignet
* Ineffizientes Parsing
* Regelbasierte Transformation von Dokumenten mit XSLT schwer zu programmieren

**JavaScript Object Notation (JSON):**

* Format für die Serialisierung von Objekten
* Notation hat die Syntax von JavaScript

**YAML:**YAML ist ein gutes Austauschformat. Es verzichtet auf Klammern, Anführungszeichen und Tags. Stattdessen wird Einrückung zur Darstellung hierarchischer Strukturen verwendet.

**REST-Schnittstellen**

**Representational State Transfer** (REST) ist ein Programmierparadigma **für verteilte Systeme**.  
REST ist **besonders geeignet für Webservices.** Es handelt sich um eine **Alternative zu RPC Verfahren.**  
Ziel ist **einheitliche Schnittstelle** für den **Zugriff** auf das **Backend.**

**REST Prinzipien:**

* **Client Server Architektur**
* **Zustandslosigkeit**
  + Jede Anfrage eines Clients ist in sich geschlossen
  + Anfrage enthält alles was Server für die Verarbeitung benötigt
  + Begünstig Skalierbarkeit eines Webservices
  + Voneinander unabhängige Anfragen auf mehrere Server verteilt werden
* **Einheitliche Schnittstelle**
  + Jeder REST-konforme Dienst hat eindeutige URL
  + Verwendung von Standardmethoden zur Manipulation von Ressourcen (GET, POST, …)

**Angular**

**Aus welchen drei Teilen (=Dateien) besteht eine Angularkomponente üblicherweise?**HTML, CSS, TS

**Was ist der Zweck des Interface "OnInit" in Angular?**Lifecyclehook. ngOnInit() wird bei Initialisierung der Komponente aufgerufen.

**Wie können Variableninhalte im HTML-Code angezeigt werden?**  
{{variablenname}}

**Wie funktioniert eine Pipe in Angular?**{{hero.name | uppercase}}. Inhalt der Variable hero.name wird per Pipe an Funktion uppercase weitergereicht.

**Wie funktioniert \*ngIf?**  
Bei \*ngIf=“<<bedingung>>“ muss eine Bedingung stehen, die entweder true oder false ist. Wenn die Bedingung wahr ist, ist die Komponente/das HTML-Tag vorhanden und wird angezeigt, sollte die bedinung false ergeben, wird sie nicht angezeigt.

**Wie funktioniert \*ngFor?**  
Mit \*ngFor kann eine Liste/Array durchlaufen werden z.B. \*ngFor=“let cat of cats“  
Das Element/die Komponenten bei der \*ngFor angegeben ist wird samt Unterelementen wiederholt.

**Wie kann in Angular auf Events reagiert werden?**  
(click)="onSelect()". click ist ein vordefiniertes event. Bei Klick wird die Methode onSelect() aufgerufen.

**Wie können in Angular Inputs an eine Komponente weitergereicht werden?**<myComponent [inputValue]="someValue">. Damit wird die Komponent myComponent angezeigt und ihr der InputValue someValue dem Inputparameter inputValue zugeordnet.

**Was ist two-way binding in Angular?**  
[(ngModel)]="hero.name"  
Die hero.name property wird an die HTML textbox gebunden (binding) damit daten in beide Richtungen fließen können.

**Wofür benötigt man Routing in Angular?**  
Um zwischen verschiedenen Komponenten zu navigieren.

**Wofür benötigt man einen Observable in Angular?**  
Für asynchrone Abarbeitung und Eventhandling. zB für asynchrones Abfragen von Daten vom Backend