Contents

1 Definitionen & Fakten

- „A **distributed system** is a collection of independent computers that appears to its users as a single coherent system.“
- „Ein **Verteiltes System** setzt sich aus mehreren Einzelkomponenten auf unterschiedlichen Rechnern zusammen, die in der Regel nicht über gemeinsamen Speicher verfügen und somit mittels Nachrichtenaustausch kommunizieren, um in Kooperation eine gemeinsame Zielsetzung – etwa die Realisierung einesGeschäftsablaufs – zu erreichen.“
- A **distributed system** is a loosely coupled set of components, which run on different computers' host systems and coordinate by means of message exchange over a communication medium to reach a common goal.
- A **Host system** is an autonomous runtime environment, which provides a component with the necessary execution and communication resources operated by that host system.
- Challenges when implementing a distributed system: Heterogenity, Openness, Security, Scalability, Availability/Error Handling, Concurrency, Transparency
  - these challenges can be solved via system models (fundamental models, architectural models)
- **Architecture** is the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.

  - system = a collection of components organized to accomplish a specific function or set of functions. The term system encompasses individual applications, systems in the traditional sense, subsystems, systems of systems, product lines, product families, whole enterprises, and other aggregations of interest. A system exists to fulfill one or more missions in its environment
  - environment = determines the setting and circumstances of developmental, operational, political, and other influences upon that system
  - mission = is a use or operation for which a system is intended by one or more stakeholders to meet some set of objectives
  - stakeholder = is an individual, team, or organization (or classes thereof) with interests in, or concerns relative to, a system

- **Architectural Styles**:
  - N-tier Architecture
    - Structuring of Distributed Systems into layers/levels, their distribution and interfaces
    - **Client/Server-Model**
      - Client/Server is an example of 2-tier Architecture
      - object oriented model (units of communication & distribution are objects)
      - component-based model (focus on reuse without the OO problems)
    - **Cluster**
      - Set of computers/servers, which are connected to each other over a fast network and can be seen as a single unit from the outside
  - **Service-Oriented Architecture** (**SOA**)
    - Process-oriented view with services as a base concept
    - Cross-platform/-enterprise service delivery
    - Interface reuse and interoperability
    - Composition of services (Orchestration & Choreography)
  - Grid Computing
    - Approach to aggregation and shared use of heterogeneous networked resources, such as computers, databases, scientific tools
  - Peer-to-Peer Architecture
    - peers communicate with each other and offer services to their partners (peers) or use the partners' services
    - unlike typical C/S architectures with a few servers and many clients, P2P architectures have no fixed assignment
    - Communication in P2P
      - Service provisioning and utilization requires peer coordination

- **Communication** is the mechanism of data exchange between components that are executed on host systems
  - **Message Exchange Models**: Direct-Addressing-Model, Queue-Communication-Model, Port-oriented-Communication-Model, Request/Response-Model, Pull/Push-Model

- **Middleware** = "glue" between software components and the network ('/' slash between Client/Server); software platform bridging the heterogeneity of different systems and networks, which simultaneously provide a number of important system services, such as security policies, transaction mechanisms and directory services
  - typical tasks/forms of middleware: RPC, MOM, EAI, Database Middleware

    * RPC (**Remote Procedure Call**)
        · idea: "embedding" a programming language, data exchange stays transparent, RPC is located above UDP or TCP in the protocol stack, client sends RPC-request to server -> server processes and sends RPC-reply
        · enables a synchronous call of functionality offered in separate processes (possibly, on remote machines), where input and output data is exchanged over a narrow channel
        · works well with small messages and is functional foundation for lots of other middleware approaches
        · problems: requires constant encoding/decoding (marshalling), debugging is complex

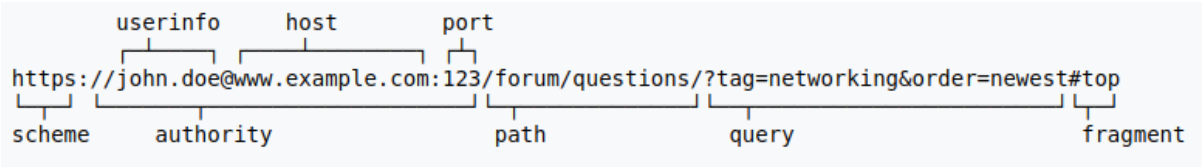
- Markup = Text that is added to the data of a document in order to convey information about it
- Descriptive Markup = Markup that describes the structure and other attributes of a document in a non-system-specific way, independently of any processing that may be performed on it
- Processing Instruction = Markup consisting of system-specific data that controls how a document is to be processed
- **Design Pattern** = "describes a particular recurring design problem that arises in specific design contexts, and presents a wellproven generic scheme for its solution."
- **Web Engineering** = – is the application of systematic, disciplined, and quantifiable approaches to the design, production, deployment, operation, maintenance and evolution of Web-based software products.
- **Web Application** = a distributed application that accomplishes a certain business need based on technologies of the World Wide Web and that consists of a set of Web-specific resources
- **Asynchronous Javascript + XML** (AJAX) while not a technology itself is a term coined in 2005 by Jesse James Garret that describes a "new" approach to using a number of existing technologies together, including HTML or XHTML, CSS, JS, the DOM, XML, XLST and most importantly the XMLHttpRequest object
    – when these technologies are combined in the AJAX model, web applications are able to make quick, incremental updates to the user interface without reloading the entire browser page; this makes the application faster and more responsive to user actions
    – although the 'X' in 'AJAX' stands for XML, JSON is now used more than XML nowadays because of its many advantages such as being lighter and a part of JS
        * both (JS and XML) are used for packaging information in the AJAX model
- **Web-Architektur** = konzeptioneller Aufbau des World Wide Web
    – das WWW oder Internet ist ein sich stetig veränderndes Medium, das einerseits die Kommunikation zwischen unterschiedlichen Nutzern und andererseits das technische Zusammenwirken (Interoperabilität) zwischen verschiedenen Systemen und Subsystemen ermöglicht. Grundlage hierfür sind unterschiedliche Komponenten und Datenformate, die meist in sogenannten Schichten (engl.: tier) angeordnet sind und aufeinander aufbauen. Im Gesamten bilden sie die Infrastruktur des Internets, die durch die drei Kernbestandteile Datenübertragungsprotokolle (TCP/IP, HTTP), Repräsentationsformate (HTML, CSS, XML) und Adressierungsstandards (URI, URL) ermöglicht wird
- Arten von Web-Architekturen:
    – Client-Server Modell = Anfangs war das Web durch das Zwei-Schichten-Modell (engl.: two tier architecture) charakterisiert: Clients und Server teilen sich die Aufgaben und Dienstleistungen, die das System erledigen soll. Der Client kann zum Beispiel einen Dienst vom Server anfordern; der Server beantwortet die Anfrage, indem er den Dienst bereitstellt. Das Abrufen einer Website per URL-Adresse, die auf einen Server verweist, und das Laden der Website im Browser des Clients ist ein Beispiel für das Zwei-Schichten-Modell, das auch als Client-Server-Modell bezeichnet wird.

  - Drei-Schichten-Modell = Drei-Schichten-Modelle sind dadurch gekennzeichnet, dass zwischen Client und Server eine Anwendungslogik tritt, die die Datenverarbeitung übernimmt und ein gewisses Maß an Interaktion ermöglicht. So kann ein Anwendungsserver Daten verarbeiten, während ein Datenbankserver sich ausschließlich der Datenspeicherung widmet. Auf diese Weise können Inhalte dynamisch geladen und gespeichert werden. Die Skriptsprache JavaScript ist häufig für das Verhalten des Clients verantwortlich.
  - Serviceorientierte Architekturen (SOA) = Heute wird das Web für die Vernetzung von global verteilten IT-Strukturen genutzt. Dabei kann jedes IT-System wiederum aus Teilbereichen bestehen, deren einzelne Komponenten über eine feste Struktur bzw. Architektur miteinander verknüpft sind. Man denke an Intranet und interne Unternehmenssoftware. Moderne IT- und Webanwendungen sind wesentlich komplexer als das Client-Server-Modell. Verteilte Webdienste, die als serviceorientierte Architekturen (SOA) aufgesetzt sind, bieten vielerlei Funktionen und modulare Funktionseinheiten, die ergänzt werden können. Mit SOAs lassen sich Geschäftsprozesse automatisieren, indem die beteiligten System - teils ohne menschliche Eingriffe - miteinander kommunizieren und bestimmte Aufgaben erledigen. Beispiele hierfür sind Online-Banking, E-Commerce, E-Learning, Online-Marktplätze oder Business Intelligence Anwendungen. Diese Architekturen sind nicht nur weitaus komplexer, sondern mitunter auch modular erweiterbar. Sie werden als N-tier-Architekturen bezeichnet und kommen bisher vor allem in der Industrie zum Einsatz.
    * zwei Ansätze: WSDL & SOAP vs REST

- **SOAP** = SOAP is a messaging protocol that allows programs that run on disparate operating systems (such as Windows and Linux) to communicate using Hypertext Transfer Protocol (HTTP) and its Extensible Markup Language (XML)
  - since web protocols are installed & available for use by all major operating systems, HTTP & XML provide an at hand solution that allows programs running under different OS's in a network to communicatte with each other. SOAP specifies exactly how to encode an HTTP header and an XML file so that a program in one computer can cann a program in another computer and pass along information. SOAP also specifies how the called program cann return a response. Despite its frequent pairing with HTTP, SOAP supports other transport protocols as well
- **REST** = Representational State Transfer (REST) is a software architecture style consisting of guidelines and best practices for creating scalable web services. REST is a coordinated set of constraints applied to the design of components in a distributed hypermedia system that can lead to a more maintainable architecture. REST efficiently uses HTTP verbs

2 URI

- idea/goal: it must be possible to identify any resources on the internet
- URI is generic term for all textual names/addresses
- URI is URL or URN or URC
- URI = Uniform Resource Identifier is a string of characters that unambiguously identifies a particular resource. To guarantee uniformity, all URIs follow a predefined set of syntax rules,[1] but also maintain extensibility through a separately defined hierarchical naming scheme (e.g. `"http://"`)
  - most common form of URI is the Uniform Resource Locator (URL), frequently referred to informally as a web address
  - generally 5 components: URI = scheme:[//authority]path[?query][#fragment]



- Uniform Resource Locator (URL)
  - The set of URI schemes that have explicit instructions on how to access the resource over the Internet
- Uniform Resource Name (URN)
  - A URI that has an institutional commitment to availability, etc.
  - A particular scheme intended to identify resources
- Uniform Resource Characteristic (URC)
  - A URC provides Meta Information
- reserved characters in URIs: "%" = escape character, "/" = delimiting substrings whose relationship is hierarchical, "#" = hash fragment identifies a fragment in a resource, "?" = query delimiter to delimit the boundary between the URI of a queryable object
- URI Syntax: <URI> ::= <scheme>":"<scheme-specific-part>
- Comparison URN vs URL

	URN	URL
Scope	global	global (abs. URL), local (rel. URL)
Globally Unique	yes	yes (abs. URL), no (rel. URL)
Scalable	yes	yes
Legacy Support	yes	limited
Resolution	not yet determined	partly using DNS

3 HTTP

Hypertext Transfer Protocol (HTTP) is used to exchange resources (such as websites, pictures, JavaScript, other MIMEtypes resources) between a user agent and a server following the Request/Response model.

- developed by W3C
- is a *transfer* (not transport) protocol
- protocol properties:
  - exchange of Request/Response data based on TCP/IP
  - stateless communication between user agent and server
  - two message types: Request and Response
  - messages are ASCII-encoded
  - messages are used to realize methods: GET, POST, HEAD, etc

Generic Structure:

```
Start Line
*Header
CRLF
[Message-Body]
```

Startline ::= Request-Line | Response-Line
Header ::= field-name":"[field-value]CRLF

- field-name = token
- field-value = *(field-content|LWS)
- LWS = Linear White Space
- cookies are also set in the response header: Set-Cookie <cookie-name> = <cookie-value>; Expires = <date>

Message-Body

- must be encoded if exists
- presence signaled by header field with field-name "Content-Length" or "Transfer-Encoding"

HTTP-Request Message:

```
<Method> <URI> <Protocol>
<Headers>
CRLF
[<Data>]
```

Method ::= GET|POST|HEAD|...
Protocol ::= HTTP/1.0 | HTTP1.1 |...
Headers ::= <hName>:<hValue>
Data ::= <TEXT>

Example(GET):

```
GET /hello.html?parameter=value HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.wikipedia.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

Example(POST):

```
POST /guestbook.php HTTP/1.1
From: frog@jmarshall.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 32

parameter=value&parameter2=value2
```

HTTP-Response Message:

```
<Protocol> <Status Code> <Reason-Phrase>
<Headers>
CRLF
[<Data>]
```

Protocol ::= HTTP/1.0 | HTTP1.1 |...
Status-Code ::= DIGIT+
Reason-Phrase ::= <TEXT>
Headers ::= <hName>:<hValue>
Data ::= <TEXT>

Example:

```
HTTP/1.1 200 OK
Date: Sun, 21 Apr 1996 02:20:42 GMT
Server: Microsoft-Internet-Information-Server/5.0
Connection: keep-alive
Content-Type: text/html
Last-Modified: Thu, 18 Apr 1996 17:39:05 GMT
Content-Length: 2543

<HTML> Some data... More and more data</HTML>
```

3.1 Typical HTTP Methods

- GET = deliver the resource addressed by the URI
- POST = request to the server with respect to processing of encoded message body data (processing wrt the URI provided in POST)
- HEAD = Like GET but without the Response Body
- OPTIONS = Request on information submission on communication options
- PUT = Resources encoded in the Body should be saved at the Request URI
- DELETE = Server should remove the resources connected to the Request URI
- TRACE = Methods for development support of the so-called application layer request loop-back; all requests of user agents that the server gets should return to the user agent

3.2 Typical HTTP HEADERS

- Content-Type = media type used
- Expires = date/time from which the response is considers invalid, important for caching
- Host = specifies internet host and port number of the requested resource
- Last-Modified = Date and time when the "variant" (object referenced by the RequestURI) was last modified, important for caching
- Location = Is set in the HTTP Response to notify the user agent of the new location of the requested Request-URI; Very important concept in different protocols which build up on HTTP, for example, in the security area; Is used for implementation of the so-called "Redirects"
- Referrer = Reference to the URI from which the user agent has posted the current Request URI; Useful for maintenance/service tasks
- User-Agent = Information about the user agent, important for personalization and internationalization

Numerous further attributes exist for use for different tasks

3.3 HTTP Status Codes

1XX = Information as intermediate response, 2XX = Successfull operations, 200 = OK, 201 = Created, 3XX = Redirects, 301 = Moved Permanently, 302 = Moved Temporarily, 4XX = Client Error, 400 = Bad Request, 401 = Unauthorized, 403 = Forbidden, 404 = Not Found, 5XX = Server Error

4 MIME

Multipurpose Internet Mail Extension

- Concept
  - MIME messages can consist of many part (-> multi-part messages)
  - message parts can have different types of content (-> Content-Type)
  - each message part has its own headers to describe the content
- Content-Type Syntax (matching of media type and subtype is case-insensitive)
  - content ::= "Content-Type":<type>/<subtype>[";"<parameter>]
  - type ::= discrete-type | composite-type
  - discrete-type ::= "text" | "image" | "audio" | "video" | "application" | extensions-token
  - composite-type ::= "message" | "multipart" | extensions-token
- Example

```
Content-Type: multipart/mixed; boundary="====-1203946231==_============="

====-1203946231==_============
Content-Type: text/plain; charset="us-ascii" ; format="flowed
====-1203946231==_============
Content-Id: <p05100306b83d3caaff11@[139.82.20.12].0.0>
Content-Type: application/vnd.ms-excel; name="WWW2002-Review-Paper-Assign.xls"
; x-mac-type="584C5334"
; x-mac-creator="5843454C"
Content-Disposition: attachment; filename="WWW2002-Review-Paper-Assign.xls"
Content-Transfer-Encoding: base64
```

5 Retrieving Information

Client Side:

1. Prepare Request

- adress the resource (URI)
  - find URI-resolver for scheme in use (eg uri resolver for http)
- URI Resolver: get address of resource (scheme specific), eg. Host: localhost, Resource: /

2. Request Resource

- send request to address (communication)
- depends on scheme of URI and the transmission protocol is defined by scheme

Server Side:
A. Handle Request

- wait for request
  - listen on server port (usually port 80)
  - accept connection
    - iterative server: one request after another (no concurrency)
    - concurrent server: one process/thread per request
  - Server-Loop for example: wait for connection request -> accept connection -> create thread/process -> go to server-loop
    - Thread/Process then -> prepare processing
- prepare processing
  - analyse HTTP request
    - extract URI, Host-Header, Port and analyse if they're supported allowed
    - create/call URI handler or respond with Error code
- process/compute
  - URI handler receives request -> wired componets are executed -> compute response

B. Send Resource

- URI Handler
  - creates header for HTTP response
  - adds response resource
  - return complete response to answering process
- answering process (eg WebServer)
  - may create additional header elements for http response
  - sends response to client

Client Side:
3. Handle Response

- handle protocol eg HTTP 302 Object moved, cookies

4. Process/Render Data of Resource

- for example process header, check content-type, process resource data depending on MIME-type eg render text/html, text/text, image/gif
- or for example store data in the read-only localStorage property which allows to acces a storage object for the Documents origin where the stored data is saved across browser sessions

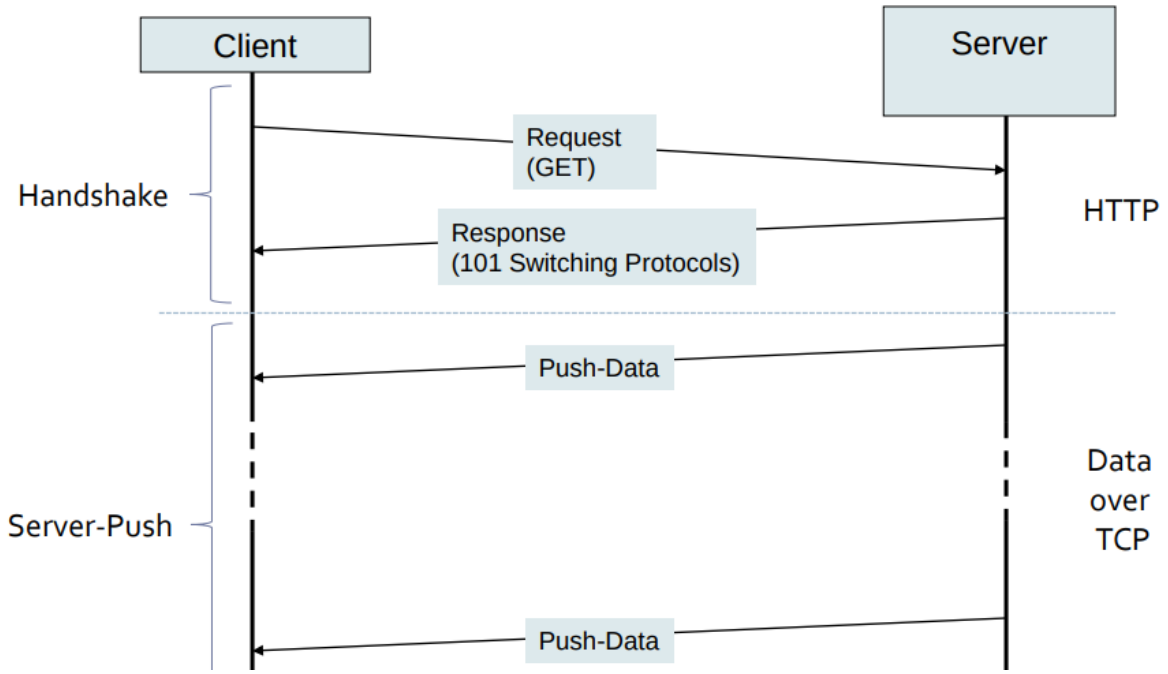
- localStorage is similar to sessionStorage, except that while data stored in localStorage has no expiration time, data stored in sessionStorage gets cleared when a page session ends - that is when the page is closed

```
var myStorage = window.localStorage;
localStorage.setItem('myCat', 'Tom');
var myCat = localStorage.getItem('myCat');
```

6 WebSockets

How can the server send (push) messages to the client if HTTP requires a prior Client request? The solution are WebSockets which is leaving an open TCP connection between Client and Server. WebSockets provide a persistent connection between a client and server that both parties can use to start sending data at any time.

Websocket Principle:



The client establishes a WebSocket connection through a process known as the WebSocket handshake. This process starts with the client sending a regular HTTP request to the server. An Upgrade header is included in this request that informs the server that the client wishes to establish a WebSocket connection.

Here is a simplified example of the initial request headers.

```
GET ws://websocket.example.com/ HTTP/1.1
Origin: http://example.com
Connection: Upgrade
Host: websocket.example.com
Upgrade: websocket
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ== # randomly generated key, processed
    by server
```

If the server supports the WebSocket protocol, it agrees to the upgrade and communicates this through an Upgrade header in the response.

```
HTTP/1.1 101 WebSocket Protocol Handshake
Date: Wed, 16 Oct 2013 10:07:34 GMT
Connection: Upgrade
Upgrade: WebSocket
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo= # processed key recieved from
    Client
```

Now that the handshake is complete the initial HTTP connection is replaced by a WebSocket connection that uses the same underlying TCP/IP connection. At this point either party can starting sending data.

With WebSockets you can transfer as much data as you like without incurring the overhead associated with traditional HTTP requests. Data is transferred through a WebSocket as messages, each of which consists of one or more frames containing the data you are sending (the payload). In order to ensure the message can be properly reconstructed when it reaches the client each frame is prefixed with 4-12 bytes of data about the payload. Using this frame-based messaging system helps to reduce the amount of non-payload data that is transferred, leading to significant reductions in latency.

Advantages:

- Server can actively use the connection
- No HTTP overhead
- No delay due to polling

7 CSS Selectors

Selector	Selects
`a`	selects elements with the `a` tag
`.red`	selects all elements with the 'red' class
`#nav`	selects the elements with the 'nav' id
`div.row`	selects elements with `div` tag and 'row' class
`[aria-hidden="true"]`	selects all elements with the aria-hidden attribute with a value of "true"
`*`	wildcard selector that selects all elements
`li a`	all `a` tags that are a child of `li` tags
`div.row *`	selects all descendants of `div` elements with 'row' class
`li > a`	selects *direct* descendants
`li + a`	selects `a` elements that are immediately preceeded by a `li` element
`li ~ a`	sibling combinator that selects all `a` elements following a `li` element
`li, a`	select all `a` and `li` elements
`p:first-child`	selects every `p` element that is first child of an element
`p:last-child`	selects every `p` element that is last child of an element
`p:nth-child(n)`	selects every `p` element that is nth child of an element
`a:not(.name)`	select all `a` elements that dont have the "name" class

And some common pseudo-classes: `:hover`, `:focus`, `:active`, `:link`, `:visited`

There are more CSS selectors but these are the most common ones.

8 HTML Forms

General:

```
<form
  method= GET |POST
  action= URI (E.g.:mailto:..., http: )
  name= form -id
  enctype= multipart /form-data|...
  target= name of frame If used >
Form Controls and HTML
</form>
```

Example:

```
<p>Name and Age Form:</p>
<form method="POST" action= "mailto:gaedke@example.com">
  <p>Name: <input type="text" name="T1" size="20"></p>
  <p>Age:
    <select size="1" name="D1">
      <option value="15-30">29 and younger</option>
      <option value="age2">30 and above</option>
    </select>
  </p>
  <input type="submit" value="Submit" name="B1">
</form>
```

9 JS Reference

Accessing DOM Elements

```
// Returns a reference to the element by its ID.
document.getElementById('someid');
// Returns an array-like object of all child elements which have all of the
   given class names.
document.getElementsByClassName('someclass');
// Returns an HTMLCollection of elements with the given tag name.
document.getElementsByTagName('LI');
// Returns the first element within the document that matches the specified
   group of selectors.
document.querySelector('.someclass');
// Returns a list of the elements within the document (using depth-first pre-
   order traversal of the document's nodes)
// that match the specified group of selectors.
document.querySelectorAll('div.note, div.alert');
// Get child nodes
var stored = document.getElementById('someid');
var children = stored.childNodes;
// Get parent node
var parental = children.parentNode;
```

Creating & Adding elements to the DOM

```
// create new elments
var newHeading = document.createElement('h1');
var newParagraph = document.createElement('p');
// create text nodes for new elements
```

```
var h1Text= document.createTextNode('This is a nice header text!');
var pText= document.createTextNode('This is a nice paragraph text!');
// attach new text nodes to new elements
newHeading.appendChild(h1Text);
newParagraph.appendChild(pText);
// elements are now created and ready to be added to the DOM.
// grab element on page you want to add stuff to
var firstHeading = document.getElementById('firstHeading');
// add both new elements to the page as children to the element we stored in
   firstHeading.
firstHeading.appendChild(newHeading);
firstHeading.appendChild(newParagraph);
// can also insert before like so
// get parent node of firstHeading
var parent = firstHeading.parentNode;
// insert newHeading before FirstHeading
parent.insertBefore(newHeading, firstHeading);
```

Modify classes

```
firstHeading.classList.remove('foo');
firstHeading.classList.add('anotherclass');
firstHeading.classList.add('foo', 'bar');
firstHeading.classList.remove('foo', 'bar');
// if visible class is set remove it, otherwise add it
firstHeading.classList.toggle('visible');
// will return true if it has class of 'foo' or false if it does not
firstHeading.classList.contains('foo');
```

Events

```
var newElement = document.getElementsByTagName('h1');
newElement.onclick = function() {
  console.log('clicked');
};
var logEventType = function(e) {
    console.log('event type:', e.type);
};
newElement.addEventListener('focus', logEventType, false);
newElement.removeEventListener('focus', logEventType, false);
window.onload = function() {
  console.log('Im loaded');
};
```

Modify styles/CSS

```
myElement.style.backgroundColor = "#D93600";
// or
myElement.style.["background-color"] = "#D93600";
```

9.1 JS + Python Socket Example

```
// Client 1
const socket =  new WebSocket("ws://localhost:8765");
```

```
const changeSpeed = function (newspeed) {
  // send speed to websocket
  if (socket.readyState === 1) {
    socket.send(newspeed);
  }
};
window.onbeforeunload = function () {
  socket.close();
}
// Python Server
import asyncio
import websockets
sockets = []
async def handler(websocket, path):
    sockets.append(websocket)
    async for message in websocket:
        for socket in sockets:
            await socket.send(message)
start_server = websockets.serve(handler, 'localhost', 8765)
asyncio.get_event_loop().run_until_complete(start_server)
asyncio.get_event_loop().run_forever()
// Client 2
const socket =  new WebSocket("ws://localhost:8765");
socket.onmessage = function(event) {
  changeSpeed(event.data);
}
window.onbeforeunload = function () {
  socket.close();
}
changeSpeed = function (newspeed) {
  speed = newspeed * SPEED_FAC;
}
```

9.2 JS XHR Example

```
const baseUrl = "https://vsr.informatik.tu-chemnitz.de/edu/2015/evs/exercises/
   jsajax/guestbook.php";
function loadData() {
const xhr = new XMLHttpRequest();
xhr.addEventListener("load", displayData);
xhr.open("GET", baseUrl);
xhr.send();
}
function displayData() {
  res = JSON.parse(this.responseText);
  const ul = document.getElementsByTagName("ul")[0];
  res.forEach((entry) => {
    addEntry(ul, entry);
  });
}
function addEntry(ul, entry) {
  let li = document.createElement("li");
```

```
  li.id = entry.id;
  li.innerHTML = `<strong>${entry.name}:</strong> ${entry.text} `;
  let a = document.createElement("a");
  a.setAttribute("href", entry.id);
  a.textContent = "(X)";
  li.appendChild(a);
  ul.appendChild(li);
  a.addEventListener("click", function (e) {
    e.preventDefault();
    e.stopPropagation();
    deleteEntry(this.getAttribute("href"));
  });
}
function deleteEntry(id) {
  const xhr = new XMLHttpRequest();
  xhr.open("DELETE", `${baseUrl}?id=${id}`);
  xhr.send();
  xhr.onreadystatechange = function (e) {
    if (xhr.readyState == 4 && xhr.status == 200) {
      res = JSON.parse(this.responseText);
      if (res.message) {
        li = document.getElementById(id);
        li.parentNode.removeChild(li);
      }
    }
  }
}
function sendData() {
  const xhr = new XMLHttpRequest();
  const nameEl = document.getElementById("name");
  const textEl = document.getElementById("text");
  const name = nameEl.value;
  const text = textEl.value;
  const params = `name=${name}&text=${text}`;
  nameEl.value = "";
  textEl.value = "";
  xhr.open("POST", baseUrl);
  xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
  xhr.send(params);
  xhr.onreadystatechange = function (e) {
    if (this.readyState == 4 && this.status == 200) {
      res = JSON.parse(this.responseText);
      const ul = document.getElementsByTagName("ul")[0];
      addEntry(ul, res.entry);
    }
  }
}
```