1. **Die Definition von Websockets**

* Webtechnologie 🡪 Lieferung der Unterstützung von Vollduplexkommunikation über ein TCP basierendes Netzwerkprotokoll
* TCP = Transmission Control Protokoll = Übertragungssteuerungsprotokoll (ist 4. Schicht im OSI-Modell = Transportschicht) 🡪 ist definiert als verbindungsorientiertes, zuverlässiges und paketvermitteltes Transportprotokoll und ermöglicht somit eine bidirektionale Kommunikation
* Dadurch Realisierung des Datenaustausch in Computernetzwerken
* Konzipiert für die Implementation in Webbrowser und Webserver, aber benutzbar für jede Client- und Serverapplikation
* Websockettechnologie liefert API für JavaScript (http://www.w3.org/TR/2011/WD-websockets-20110419/) 🡪 standardisiert durch den W3C
* Außerdem Benutzung eines für WebSockets exklusiven Protokolls, standardisiert durch den IETF (Internet Engineering Task Force) [Zuständig für die Entwicklung von Internet Standards und Koorperation mit W3C und der ISO] <http://tools.ietf.org/html/rfc6455>

1. Allgemeines zu Websockets (Falco)

2. Verwendungen der Websockets (Johannes)

3. Gründe zur Verwendung von Websockets (Johannes)

4. Bestandteile der Websockets

   - Netzwerkprotokoll (Johannes)

   - Standardisierte JavaScript Client-API für Websockets (Falco)

5. Technologien zur Anwendung von Websockets (Falco)

Beispiele zu Websocktes

- NodeJS-Chat (Falco)

- PHP-Beispiel (Johannes)

Zusammenfassung (Falco)

4. **Standardisierte JavaScript Client-API für Websockets**

- Stellen Möglichkeiten zur Kommunikation mit einem Server dar, der Websockets laut W3C <http://www.w3.org/TR/2011/WD-websockets-20110419/> implementiert

- Außen Vorlassen der serverseitige Implementierung von Websockets, da Technologien für Webserver diese bereits implementieren und mit einer eigenen API zur Verfügung stellen

- Clientseitige JavaScript-API

// Erstelle neuen Websocket und Verbinden  
var mySocket = new WebSocket("ws://echo.websockets.org");

- Erzeugen eines JavaScript-Objektes für den Aufbau einer Verbindung zum WebSocket-Server

- Konstruktor 🡪 Entgegennahme der URL des Servers

- Danach Definition von Eventlisteners möglich

// Anfügen von Eventlisteners zum Socket  
mySocket.onmessage = function(event) {alert("Der Server sagt: " + event.data); };  
mySocket.onopen = function(event) {...};  
mySocket.onclose = function(event) {...};  
mySocket.onerror = function(event) {...};

"onopen": Aufruf erfolgt nach dem Herstellen der Verbindung.

"onclose": Aufruf erfolgt, sobald die Verbindung vom Client oder vom Server geschlossen wurde.

"onerror": Aufruf erfolgt im Fehlerfall.

"onmessage": Aufruf erfolgt, sobald Nachrichten vom Server im Client ankommen.

- onmessage = wichtigster Listener, denn zuständig für die Entgegennahme der ankommenden Daten 🡪 Auslesen des standardisierten „data“-Feld des übergebenen event-Objektes

// Send data...  
mySocket.send("Hallo Server!");  
// Close WebSocket  
mySocket.close();

- Ermöglichung der Kommunikation mit dem Server

- Nach erfolgreichem Aufbau der WebSocket-Verbindung 🡪 Senden von Nachrichten über den bidirektionalen Kommunikationskanal mit der JavaScript-Funktion send()

- Beim Wunsch der Verbindungsbeendigung, Aufruf der Methode close() 🡪 onclose-Listener wird aufgerufen beim Schließen der Verbindung

**5. Technologien zur Anwendung von Websockets**

- kurzes Nennen von Servertechnologien zum Hosten einer Websocket-Anwendung

- Bsp.:

Apache httpd (via "mod\\_pywebsocket")

Node.js/Socket.io

jWebSocket

Jetty

Kaazing WebSocket Gateway

Oracle Glassfish 3.1

Netty Project

**Beispiel: NodeJS (Chat-Example)**

**Zusammenfassung**

**- Websockets sind eine standardisierte Technologie für die bidirektionale Client-Serverkommunikation**

**- Ermöglichung einer direkten und schnellen Reaktion auf gesendete Daten über ein Computernetzwerk**

**- profitabel für „Echtzeitanwendungen“, die geringe Latenzzeiten für Nachrichten bedingen 🡪 soziale Netzwerke (Twitter und Facebook), kollaborative Anwendungen (Etherpad) und Spiele mit Echtzeitanforderungen (z.B. Strategiespiele)**