HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN
MÜNCHEN

Bachelorarbeit im Studiengang B.Sc. Wirtschaftsinformatik

Realisierung einer bidirektionalen Chat-Anwendung durch HTML5 WebSockets mit Java und Angular 4 zum Einsatz im seminaristischen Kontext

Andre Weinkötz

15. Februar 2018

Fakultät für Informatik und Mathematik
Hochschule München

Betreuer: Prof. Dr. Mandl

Abstract

Hier steht am Ende was genau eigentlich mit dem Unsinn hier erreicht werden soll.

Inhaltsverzeichnis

1 Einleitung

Die Anforderungen an Webanwendungen haben sich in den vergangenen Jahren stark verändert. Mobile Geräte wie Smartphones oder Tablets ersetzen stationäre Systeme, Webanwendungen sollen zur Kollaboration eingesetzt werden und Buchungssysteme oder Finanzanwendungen verlangen die Bearbeitung tausender Anfragen mit minimaler Verzögerung. Das Hypertext Transfer Protocol (HTTP) bietet hier keine zufriedenstellende Lösung. Um den Anforderungen an moderne Webanwendungen gerecht zu werden, spezifizierten das World Wide Web Consortium (W3C) und die Internet Engineering Taskforce (IETF) das WebSocket-Protokoll mit zugehöriger JavaScript-API.

Die Entwickler des WebSocket Protokolls machten es sich zur Aufgabe, einen Mechanismus zu schaffen, der es browserbasierten Anwendungen ermöglicht bidirektional zu kommunizieren ohne dabei auf mehrere HTTP Verbindungen zu öffnen [FM11]. Dabei sollte auf zusätzliche Methoden - wie beispielsweise den Einsatz von XMLHttpRequests, iframes oder long polling - verzichtet werden.

2 Das WebSocket Protokoll

Die Arbeit an der Spezifikation des WebSocket Protokolls begann bereits 2009, als es von Google in Zusammenarbeit mit Apple, Microsoft und Mozilla im Rahmen ihrer Kooperation WHATWG[1] der Internet Engineering Task Force (IETF) vorgeschlagen wurde. Bis Mai 2010 wurden noch zahlreiche Verbesserungen hinzugefügt, bis es schließlich in Version 76 [Hic10] im August 2010 an die BiDirectional or Server-Initiated HTTP (HyBi) Task Force der IETF zur Weiterentwicklung übergeben wurde [Fet10]. Neben vielen anderen Fortschritten wurde das Protkoll um die Möglichkeit erweitert binäre Dateien auszutauschen, sowie Sicherheitslücken geschlossen, die in Verbindung mit Proxy-Servern entstehen konnten. Im Dezember 2010 wurde das WebSocket Protokoll von der IETF zu dem Request for Comment (RFC) 6455 erklärt [FM11]. Dieser definiert auf über 70 Seiten neben dem Lebenszyklus und den verschiedenen Frames der WebSockets noch zahlreiche weitere technische Details und Defintionen. In diesem Kapitel sollen die wichtigsten Grundlagen der Protokollspezifikation dargelegt werden.

2.1 WebSocket-Frames

Die Spezifikation eines WebSocket-Frames sieht eine Zweiteilung vor, wie sie bei Protokollnachrichten üblicherweise vorgenommen wird. Ein WebSocket-Frame besteht demnach aus einem Header Bereich der Kontrolldaten enthält, sowie dem Payload, welcher die Nutzdaten beinhaltet. Anhand des Aufbaus ist leicht erkennbar, dass die Kommuni-

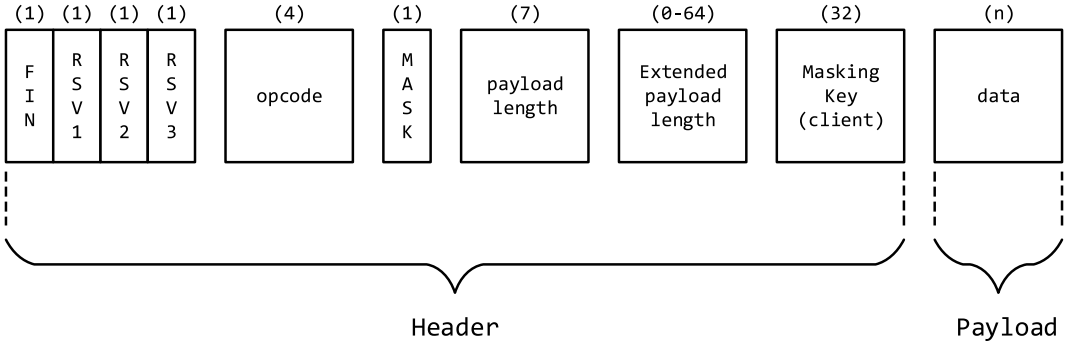

Abbildung 1: Vereinfachte Darstellung eines WebSocket Frames

kation über WebSockets mit deutlich geringeren Paketgrößen arbeiten kann als HTTP. Eine Nachricht mit einfacher Längenangabe, die von Server zu Client geschickt wird, er-

[1] Web Hypertext Application Technology Working Group unter: www.whatwg.org

zeugt durch ihren Header lediglich einen Overhead von zwei Byte. Da WebSocket-Frames auf umgekehrtem Weg zusätzlich noch maskiert werden müssen, entspricht die minimale Länge eines clientseitigen Frames sechs Byte . Im Vergleich dazu beträgt der Overhead bei einer einfachen GET-Anfrage über HTTP/1.1 beim Aufruf des Hosts unter cs.hm.edu 35 Byte. Die Angabe des Hostnamens in HTTP/1.1 ist obligatorisch und trägt maßgeblich zur Größe des HTTP-Frames bei [Lea+99, S. 128].

2.1.1 Header-Felder

Im nachfolgenden Abschnitt werden die Felder sowie deren jeweilige Aufgabe detailliert erläutert:

FIN (1 Bit) Das FIN-Flag gibt an, ob es sich bei dem empfangenen Frame um ein Fragment handelt. Kann der Payload nicht auf einmal übertragen werden, so wird die Übertragung über mehrere Frames gesteuert. Das FIN-Flag aller unvollständigen Fragmente weist den Wert 0 auf, wohingegen das finale Fragment mit einer 1 gekennzeichnet ist. Wenn alle Nutzdaten mit einem einzigen Frame übertragen werden können, so trägt dessen FIN-Flag ebenfalls den Wert 1.

RSV1, RSV2, RSV3 (jeweils 1 Bit Diese Flags sind für zukünftige Implementierung reserviert und finden aktuell keine standardisierte Anwendung. Daher tragen sie meist den Wert 0. Ist eines dieser Flags gesetzt und die empfangende Stelle hat keine Erweiterung, welches das jeweilige Flag interpretieren kann, so muss die WebSocket-Verbindung geschlossen werden [FM11, S. 27].

opcode (4 Bit) Das opcode-Feld enthält die Angabe wie der Payload zu interpretieren ist. Neben den sogenannten Datenframes bzw. Non-Control-Frames, welche die Art der übertragenen Daten kennzeichnen, sind Control-Frames definiert, die zur Steuerung und Überprüfung der Verbindung verwendet werden. Wird ein unbekannter Opcode empfangen, so wird die WebSocket-Verbindung geschlossen.

- **0x0** definiert ein Fortsetzungs-Frame
- **0x1** definiert ein Text-Frame
- **0x2** definiert ein Binary-Frame
- **0x3-7** reserviert für weitere Non-Control-Frames
- **0x8** definiert ein Verbindung beenden-Frame
- **0x9** definiert ein Ping-Frame

- **0xA** definiert ein Pong-Frame
- **0xB-F** reserviert für weitere Control-Frames

MASK (1 Bit) Ist das MASK-Flag gesetzt, so ist der Payload maskiert. Frames, die clientseitig initiert wurden, müssen maskiert sein, wohingegen serverseitige Frames keine Maskierung vornehmen dürfen. Der verwendete Masking-Key muss im gleichnamigen Header-Feld hinterlegt werden.

payload length (7 Bit) Hier wird die Länge der Nutzdaten angegeben. Sind diese nicht größer als 125 Bytes, so können sie mit nur einem Frame übertragen werden. Nutzdaten mit einem höheren Speicherplatzbedarf werden je nach Größe mit dem Wert 126 bzw. 127 angegeben. Diese Angaben führen zur Verwendung des Header-Felds *Extended payload length*.

Extended payload length (16 Bit oder 64 Bit) Überschreitet die Länge des Payloads 125 Byte, so wird dieses Feld verwendet. Je nach Wert im Header-Feld *payload length* umfasst die *Extended payload length* zwei Byte oder acht Byte. Dementsprechend beträgt die maximale Länge der Nutzdaten in einem WebSocket-Frame $2^{64} - 1$ Byte [GLN15, S. 41]

Masking Key (0 oder 32 Bit) Ein Frame, der von einem Client an einen Server gesendet wird, muss mit einem 32-Bit Schlüssel maskiert werden. Dieser wird vom Client generiert und im gleichnamigen Header-Feld abgelegt. Nachrichten, die von einem Server an einen Client gesendet werden, enthalten dieses Feld nicht.

data (n Byte) Hier befinden sich die eigentlichen Nutzdaten. Ihre Größe wird in den entsprechenden Längenfeldern des Headers hinterlegt. Werden lediglich Steuerungsinformationen übertragen, wird kein Payload mitgesendet.

2.1.2 Non-Control-Frames

Durch das Header-Feld *opcode* wird die Art der Nutzdaten angegeben. Es können sowohl textbasierte Daten als auch Binärdaten übertragen werden. Bei der Kommunikation über WebSockets gibt es hier einige zusätzliche Mechanismen, die für den fehlerfreien Ablauf einer Übertragung notwendig sind. Der folgende Abschnitt handelt von der bereits aus anderen Protokollen bekannten Fragmentierung, die WebSocket-Frames zur sequentiellen Übertragung unvollständiger Daten nutzt sowie der aufgrund von Sicherheitsproblemen eingeführten Maskierung clientseitig initierter Nachrichten.

Literatur

[Fet10] Ian Fette. *The WebSocket Protocol*. 31. Aug. 2010. URL: `https://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-01` (besucht am 19. 12. 2017).

[FM11] Ian Fette und Alexander Melnikov. *The WebSocket Protocol*. 1. Dez. 2011. URL: `https://tools.ietf.org/html/rfc6455` (besucht am 19. 12. 2017).

[GLN15] Peter Leo Gorski, Luigi Lo Iacono und Hoai Viet Nguyen. *WebSockets: moderne HTML5-Echtzeitanwendungen entwickeln*. OCLC: 901007083. München: Hanser, 2015. 269 S. ISBN: 978-3-446-44371-6 978-3-446-44438-6.

[Hic10] Ian Hickson. *The WebSocket Protocol*. 6. Mai 2010. URL: `https://tools.ietf.org/search/draft-hixie-thewebsocketprotocol-76` (besucht am 19. 12. 2017).

[Lea+99] Paul J. Leach u. a. *Hypertext Transfer Protocol – HTTP/1.1*. 1. Juni 1999. URL: `https://tools.ietf.org/html/rfc2616` (besucht am 20. 12. 2017).