tgm
Die Schule der Technik

JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

Softwareentwicklung

4BHIT 2017/18, Gruppe A

JS Socket

Laborprotokoll

Christoph Kern

7. Mai 2018

Bewertung:		Version:	1.1
Betreuer:	Michael Borko	Begonnen:	02 Mai 2018
		Beendet:	02 Mai 2018

Inhaltsverzeichnis

1 Einführung

Dieses Protokoll soll dazu dienen, Wissen über JS Sockets zu schaffen um diese bei dem SEW Projekt anwenden zu können.

1.1 Ziele

Hier sollen die wichtigsten Sachen von JS Sockets erläutert werden und alles soll nachvollziehbar sein.

1.2 Voraussetzungen

- JS kenntnisse

1.3 Aufgabenstellung

Dieses Protokoll soll dazu dienen, Wissen über JS Sockets zu schaffen um diese bei dem SEW Projekt anwenden zu können.

2 Konfiguration

2.1 Verbindung herstellen

Ein Websocket wird einfach erstellt, indem der Konstruktor des WebSockets aufgerufen wird.

```
var connection = new WebSocket('ws://html5rocks.websocket.org/echo', ['soap',
↪  'xmpp']);
```

ws: ist das neue URL-Schema für WebSocket-Verbindungen. Außerdem gibt es wss: für sichere WebSocket-Verbindungen, genau wie https: für sichere HTTP-Verbindungen.

Das zweite Argument akzeptiert optionale Subprotokolle. Es kann sich um einen String oder um ein Array von Strings handeln. Jeder String sollte einen Subprotokollnamen darstellen. Der Server akzeptiert nur eines der weitergegebenen Subprotokolle im Array. Das akzeptierte Subprotokoll wird durch den Zugriff auf die protocol-Eigenschaft des WebSocket-Objekts festgelegt.
Die Subprotokollnamen müssen den registrierten Subprotokollnamen in der IANA-Registrierung entsprechen. Zurzeit gibt es nur einen seit Februar 2012 registrierten Subprotokollnamen (soap).

2.2 Überprüfungen

Hier werden wird die Verbindung zu dem Server überprüft.

```
// When the connection is open, send some data to the server
connection.onopen = function () {
  connection.send('Ping'); // Send the message 'Ping' to the server
};

// Log errors
connection.onerror = function (error) {
  console.log('WebSocket Error ' + error);
};

// Log messages from the server
connection.onmessage = function (e) {
  console.log('Server: ' + e.data);
};
```

2.3 Kommunikation mit dem Server

Sobald man mit dem Server vebunden ist, also das 'open-Ereignis' ausgelöst wurde, kann man mit dem Server kommunizieren. Hierzu kann die send('Message') Methode verwendet werden. Sie unterstützte bisher nur Strings, kann aber seit der neusten Spezifikation auch Binärmitteilungen senden. Zum Senden von Binärdaten kann man entweder das Blob- oder das ArrayBuffer-Objekt verwenden.

```
// Sending String
connection.send('your message');

// Sending canvas ImageData as ArrayBuffer
var img = canvas_context.getImageData(0, 0, 400, 320);
var binary = new Uint8Array(img.data.length);
for (var i = 0; i < img.data.length; i++) {
  binary[i] = img.data[i];
}
connection.send(binary.buffer);

// Sending file as Blob
var file = document.querySelector('input[type="file"]').files[0];
connection.send(file);
```

2.4 Antwort vom Server

Der Server kann gleichermaßen jederzeit Mitteilungen senden. Sobald das der Fall ist, wird der onmessage-Rückruf ausgelöst. Der Rückruf empfängt ein Ereignisobjekt. Auf die eigentliche Nachricht kann über die data-Eigenschaft zugegriffen werden.
WebSocket kann in der letzten Spezifikation auch Binärmitteilungen empfangen. Binär-Frames können im Blob- oder ArrayBuffer-Format empfangen werden.

```
// Setting binaryType to accept received binary as either 'blob' or 'arraybuffer'
connection.binaryType = 'arraybuffer';
connection.onmessage = function(e) {
  console.log(e.data.byteLength); // ArrayBuffer object if binary
};
```