Aufgabe 3: Webpack und WebRTC

In dieser Aufgabe soll es darum gehen, die Entwicklung und das Ausliefern des Chat-Clients durch Bundling zu verbessern, sowie einen geheimen Nachrichtenaustausch zwischen Nutzern zu implementieren. Dafür sollen die beiden Technologien *Webpack* und *WebRTC* zum Einsatz kommen.

3.1 Vorbereitung

Als Basis für diese Aufgabe dient Ihre Chat-Client-Implementierung aus der letzten Aufgabe. Um diese für die aktuelle Aufgabe vorzubereiten, befolgen Sie die folgenden Schritte:

1. Kopieren Sie die folgenden Dateien aus dem Ordner der letzten Aufgaben in dem aktuellen Aufgabenordner in den `src`-Unterordner: `index.html`, `index.css`, `client.js`, `giphy.js` und `giphy.css`
2. Installieren Sie alle für die Aufgabe notwendigen Pakete und Addons für Webpack mittels:
   `npm install --save-dev webpack webpack-cli css-loader html-webpack-plugin mini-css-extract-plugin`
   und
3. Installieren Sie die WebRTC Library PeerJs mittels:
   `npm install --save peerjs`

3.2 Bundling

Im Folgenden soll nun die bestehende Lösung mit Webpack gebündelt und ausgeliefert werden. Dafür müssen zunächst einige Änderungen an der aktuellen Lösung getätigt werden:

- Durch den Einsatz von Webpack existiert kein globaler Namensraum mehr. Das bedeutet, dass die Integration der Giphy-API in `giphy.js` ggf. angepasst werden muss, sodass sie auf keine Variablen aus `client.js` zugreift (insbesondere den DOM bzw. das *document*-Objekt). Zudem müssen alle von außen zugreifbaren Funktionen in `giphy.js` exportiert werden (siehe Tafelübung).
- Die `index.html`-Datei muss angepasst werden, sodass keine `.css`-Dateien, und nur die `bundle.js` anstelle der `client.js` und `giphy.js`-Dateien enthalten sind.

Als nächstes soll nun das eigentliche Bundling passieren. Legen sie hierfür die Datei `webpack.config.js`[1] an, die sowohl JavaScript, als auch CSS-Code[2] bündeln soll. Dabei ist zu beachten, dass der Server die auszuliefernden Dateien im Unterordner `dist` erwartet. Abschließend können Sie mit dem Befehl `webpack --config webpack.config.js` die gebündelten Dateien generieren.

Betrachten Sie für das Bundeln die folgenden Plugins:

- `html-webpack-plugin`[3]
- `mini-css-extract-plugin`[4]

Zusätzliche Hinweise:

- Die generierten `package.json` und `package-lock.json` Dateien sollen in das Git-Repository hinzugefügt werden. Der generierte `node_modules`-Ordner, sowie die generierten Bundle-Dateien (z.B. `bundle.js`) hingegen jedoch **nicht**. Legen Sie sich hierfür am besten eine *gitignore*-Datei[5] an.
- Um die Entwicklung angenehmer zu gestalten, empfehlen wir den Bundling-Befehl in ein *npm-Skript*[6] auszulagern.
- Für eine einfachere Fehlersuche empfiehlt sich für Webpack die Verwendung des Development-Modus' sowie von Source-Maps[7].

3.3 WebRTC

Nun soll der *Secret Messaging Modus* (SMM) zu dem bestehenden Chat-Client hinzugefügt werden. Dieser tauscht Nachrichten über WebRTC direkt zwischen Nutzern aus, ohne diese auf dem Server zwischenzuspeichern.

[1] `https://webpack.js.org/guides/getting-started/`
[2] `https://webpack.js.org/guides/asset-management/#loading-css`
[3] `https://webpack.js.org/plugins/html-webpack-plugin/`
[4] `https://webpack.js.org/plugins/mini-css-extract-plugin/`
[5] `https://www.atlassian.com/git/tutorials/saving-changes/gitignore`
[6] `https://docs.npmjs.com/cli/v7/using-npm/scripts`
[7] `https://webpack.js.org/guides/development/`

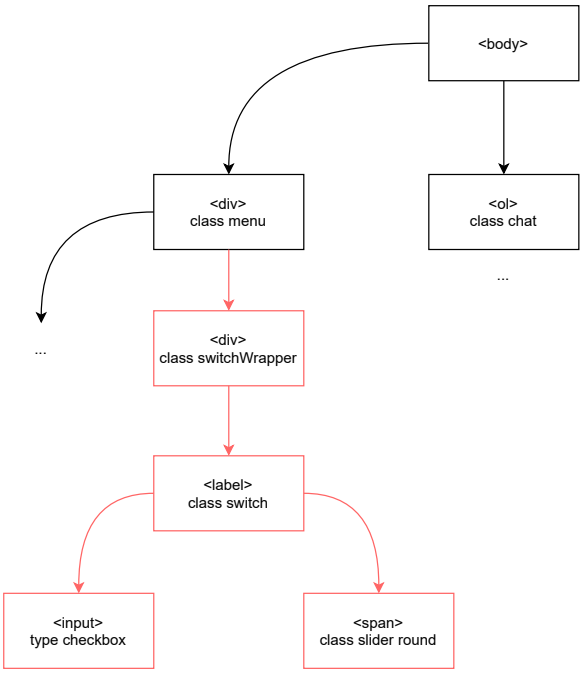

Abbildung 1: Angepasste HTML Struktur

Fügen Sie dazu zuerst die rot markierten Elemente aus Abbildung 1 in das aktuelle HTML-Dokument ein und binden außerdem die ausgeteilte `webrtc.css`-Datei ein. Dadurch wird ein Slider hinzugefügt, mit dem der SMM (de-)aktiviert werden kann.

Implementieren Sie nun den SMM in den folgenden Schritten:

1. Wenn der Slider betätigt wird, verbindet sich der Client mit dem Peer.js-Server um eine ID zu erhalten. Diese wird über die bestehende Websocket-Verbindung mittels einer *SMM*-Nachricht an den Chat-Partner übermittelt.

2. Wird eine *SMM*-Nachricht erhalten, soll der Nutzer gefragt werden, ob der SMM aktiviert werden soll. Wenn dieser akzeptiert, soll mithilfe der ID eine WebRTC-Verbindung aufgebaut und der Slider auf aktiv gesetzt werden. Andernfalls wird ein *noSMM*-Event zurückgesendet, und der Slider auf der Gegenseite deaktiviert. Gleiches gilt, wenn der Slider wieder manuell deaktiviert wird.

3. So lange der SMM aktiv ist, sollen alle Nachrichten über die WebRTC-Verbindung geschickt werden. Alle Nachrichten die über diese Verbindung eingehen, sollen als Elemente der Klasse `self private` oder `other private` an den Chat-verlauf angehängt werden.

Zudem sollen zu jedem Zeitpunkt alle auftretenden Fehler bei der WebRTC-Verbindung **sinnvoll** behandelt werden.

Zusätzliche Hinweise:

- Optional kann auch ein lokaler Peer.js-Server verwendet werden[8].
- Es empfiehlt sich die Behandlung von Nachrichten in eine separate Funktion auszulagern, welche als Handler für Websocket und WebRTC Nachrichten benutzt werden kann.

Letzter Commit bis zum 9. Dezember.
Termine für die Vorstellungen in StudOn buchbar.

Präsentation der fertigen Lösung spätestens am Tag der Abgabefrist in der Rechnerübung!

[8] `https://github.com/peers/peerjs-server`