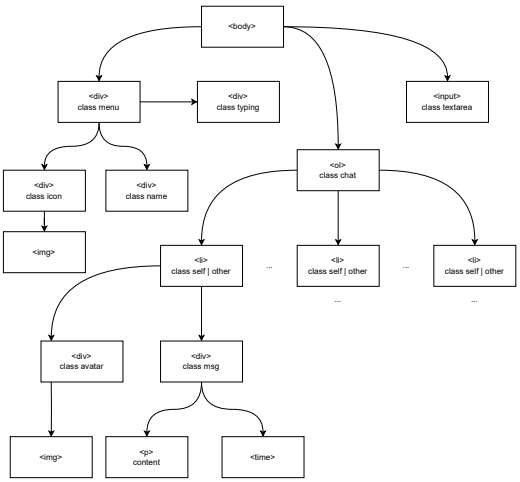
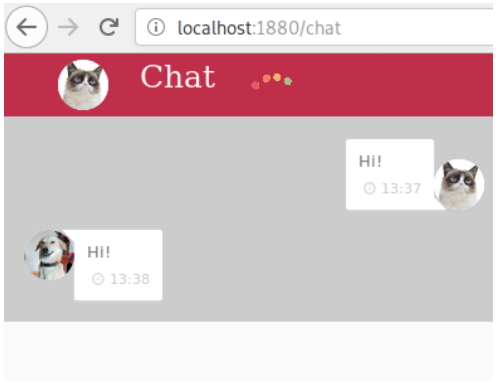
Aufgabe 2: Erweiterung des Chat-Clients

Nachdem Sie in Aufgabe 1 einen rudimentären Chat-Client entwickelt haben, sollen Sie in dieser Teilaufgabe den Client weiterentwickeln. Der Server wird wie in der vorherigen Aufgabe vorgegeben, wurde aber erweitert: Zusätzlich zu der Datei `index.html` werden zwei weitere Dateien ausgeliefert: `client.js` und `index.css`. Alle Teilaufgaben dieses Aufgabenblattes sollen mit den oben genannten Dateien implementiert werden.



(a) HTML-Struktur



(b) UI der Chatanwendungen

Abbildung 1: HTML Struktur und UI des Chatclients

2.1 Statische HTML-Struktur

In Ihrem Repository finden Sie eine größtenteils leere Datei mit dem Namen `index.html`. Nicht wundern: Wenn Sie den Server das erste mal starten, wird die Seite des Chats komplett leer sein. (Um zu überprüfen ob der Server richtig läuft können Sie sich den Quelltext vom Browser anzeigen lassen.) Erweitern Sie diese Datei so, dass der `body` des HTML-Dokuments die in Abbildung 1a gezeigte Struktur enthält. Nutzen Sie `img`-Tags, um die `div`-Tags der folgenden Klassen mit Inhalt zu füllen: `icon`, `typing` und `avatar`. Nutzen Sie dazu die folgenden Ressourcen:

- `https://sys.cs.fau.de/extern/lehre/ws22/wbs/uebung/cat.jpg`
- `https://sys.cs.fau.de/extern/lehre/ws22/wbs/uebung/dog.jpg`
- `https://sys.cs.fau.de/extern/lehre/ws22/wbs/uebung/dots.gif`

Verwenden Sie für den `li`-Tag jeweils einmal die Klasse `self` und `other`. Mit diesen beiden Klassen sollen eigene und Nachrichten von anderen Teilnehmern unterschiedlich dargestellt werden. Füllen Sie auch die `p`- und `time`-Tags mit zunächst beliebigem Inhalt. Am Ende sollte ihr Client wie im Screenshot in Abbildung 1b aussehen. Die Test Nachrichten können nach dieser Teilaufgabe wieder aus dem HTML Dokument entfernt werden. Erweitern Sie das HTML-Dokument später, wenn sie während der Bearbeitung zusätzliche `id`-Attribute benötigen.

2.2 Lokale Verwaltung des Nachrichtenverlaufs

Betrachten Sie nun die vorgegebene Funktion addMessageToHistory in der client.js. Verinnerlichen Sie sich inwiefern `Destructuring`[1] für die Funktionsparameter angewandet wurde und in welcher Form die Funktion

[1] `https://simonsmith.io/destructuring-objects-as-function-parameters-in-es6`

aufgerufen werden kann. Implementieren Sie daraufhin folgendes Verhalten:

- Die Funktion fügt die korrekten HTML-Elemente in das `ol`-Tag ein, sodass der Nachrichtenverlauf wie in 1b dargestellt wird.
- Der Parameter `account` kann die Strings `self` oder `other` enthalten, je nachdem wird HTML für eigene oder fremde Nachrichten eingefügt.
- `time` ist ein Objekt des Typs `Date`[2]. Wird kein Objekt übergeben (also `time ist falsy`[3]), soll die aktuelle Zeit als Zeitstempel eingefügt werden, ansonsten der übergebene Zeitstempel.

Registrieren Sie `onkeydown` Event, welches überprüft welche Taste gedrückt wurde. Wurde Enter gedrückt (siehe Tafelübung) soll der Inhalt des `textarea`-Tags mit `addMessageToHistory` zum lokalen Nachrichtenverlauf hinzugefügt werden.

2.3 Einfache Sitzungsverwaltung

Die Teilnehmer des Chats werden durch ihre Benutzernamen auseinandergehalten. Um auf die Implementierung von Authentifizierungsmechanismen zu verzichten soll die Verwaltung von Sitzungen (engl. *Sessions*) ausschließlich *auf Clientseite* umgesetzt werden. Dazu soll der Chatclient den Benutzernamen ungeprüft speichern. Verwenden sie dazu die in der Tafelübung vorgestellte Technologie *Web Storage*.

- Implementieren Sie folgendes Verhalten, wenn `/chat` im Browser geöffnet wird:
  - Wenn Benutzername nicht gesetzt: Der Benutzer wird zur Eingabe eines Nutzernamens aufgefordert, welcher anschließend gespeichert wird.
  - Wenn Benutzername gesetzt: Der Benutzer wird mit der Ausgabe des gespeicherten Nutzernamens begrüßt.

  Nutzen Sie zur Kommunikation mit dem Nutzer die Methoden `prompt()` und `alert()`, die im `Window`-Objekt[4] definiert sind.

2.4 Bidirektionale Kommunikation

Mit *WebSockets* (siehe Tafelübung) kann ein *bidirektionaler Kommunikationskanal* zwischen Browsern und Webservern hergestellt werden. Der mit der Aufgabe mitgelieferte Server unterstützt dies bereits und stellt den Endpunkt unter `ws://localhost:1880/ws` bereit. Mit Hilfe von WebSockets soll in dieser Teilaufgabe sowohl die Übermittlung von Nachrichten als auch sogenannte *Typing Notifications* umgesetzt werden. Dabei soll zwischen drei Nachrichtentypen unterschieden werden:

- `message`: Um eine vom Benutzer geschriebene Nachricht zu verschicken.
- `typing`: Um zu signalisieren, dass der Nutzer angefangen hat zu schreiben.
- `noLongerTyping`: Um zu signalisieren, dass der Nutzer mit dem schreiben aufgehört hat.

Dies muss innerhalb des eines JSON Objektes als {..., event: Nachrichtentyp, ...} angegeben werden.

Instanziieren Sie das `WebSocket`-Objekt in der Datei `client.js` und implementieren Sie folgendes Verhalten:

Emittieren von Events

- Eine Nachricht vom Typen `message` wird emittiert, wenn der Nutzer eine Nachricht geschrieben hat und Enter drückt.
- Eine Nachricht vom Typen `typing` wird emittiert, wenn der Nutzer eine beliebige Taste drückt. Ausgenommen sind Tasten, die *keinem Zeichen* zugeordnet werden können.
- Eine Nachricht vom Typen `noLongerTyping` wird emittiert, wenn der Nutzer 2 Sekunden lang nicht getippt hat.

[2] `https://developer.mozilla.org/docs/Web/JavaScript/Reference/Global_Objects/Date`
[3] `https://developer.mozilla.org/de/docs/Glossary/Falsy`
[4] `https://developer.mozilla.org/docs/Web/API/Window`

Empfangen von Events

- Fügen Sie neu empfangene Nachrichten dem lokalen Nachrichtenverlauf hinzu (siehe Teilaufgabe 2.2).
- Die Typing Notifications sollen über das Ein- und Ausblenden des `div`-Tags mit der Klasse `typing` umgesetzt werden. Blenden Sie das entsprechende Element beim Empfang des `typing`-Events ein und beim Empfang des `noLongerTyping`-Events aus.

Laden des Chatverlaufs mit HTTP

- Benutzen Sie das HTTP REST Interface unter `/savedMessages`, um den gesamten Chatverlauf bei Neustart des Clients zu laden.

2.5 Desktop-Benachrichtigungen

Der Benutzer Ihres Chat-Clients soll neu eintreffende Nachrichten von anderen Chat-Teilnehmern auch wahrnehmen, wenn das Browser-Fenster nicht geöffnet ist. Nutzen Sie die *Notifications API*[5] um diese Funktion umzusetzen. Behandeln Sie dabei auch den Fall, dass der Nutzer der Anzeige von Benachrichtigungen nicht zustimmt!

Zusätliche Hinweise:

- Beachten Sie, dass die Notifications Api nur Lokal oder mittels HTTPS funktioniert [6].

2.6 Integration der Giphy API

In dieser Teilaufgabe soll der Chatclient um die Zugriffsmöglichkeit auf eine Web API erweitert werden. Zusätzlich kann der gegebene Server auch die Dateien `giphy.js` und `giphy.css` ausliefern. Sorgen Sie zunächst dafür, dass diese beiden Dateien auch vom Browser angefragt werden. Implementieren Sie dann in diesen Dateien die Lösung für diese Aufgabe.

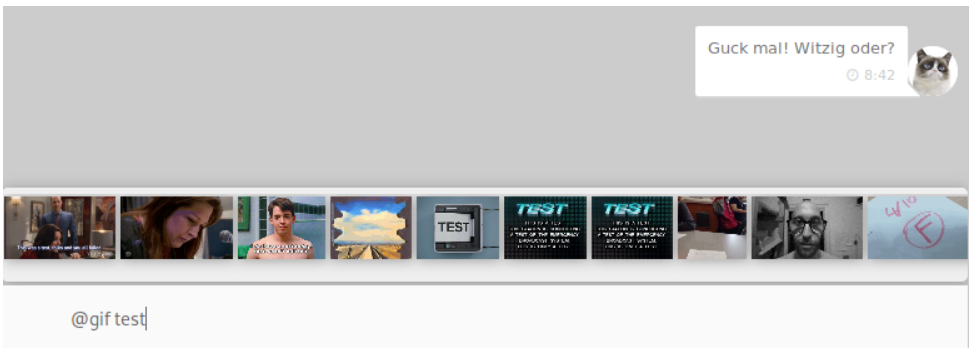


Abbildung 2: Beispielhafte Einbindung von GIFs im Chatclient

Giphy[7] ist eine Suchmaschine und Datenbank für *animierte GIFs*, die seit 2013 existiert. Das Unternehmen bietet eine API[8] an. Diese API ist nur mit einem API Key benutzbar; benutzen Sie den in der Datei `giphy.js` vorgegebenen Key.

- Nutzen Sie die "Search"-Funktion[9] der API um dem Nutzer Ihres Chatclients zu erlauben, GIFs zu suchen: Bei Eingabe des Schlüsselworts `@gif` sollen die darauf folgenden Worte als Ajax GET Request an die Giphy Search API gesendet werden. Dabei soll bei *jedem Tastendruck* ein neuer Request versendet werden, um wiederholt Vorschläge zur aktuellen Eingabe zu erhalten.
- Implementieren Sie eine an Abbildung 2 angelehnte Anzeige der Suchergebnisse. Diese Anzeige soll laufend im Hintergrund parallel zur Nutzereingabe aktualisiert werden. Klickt der Nutzer auf eines der GIFs soll dies als Nachricht der Form

```
<img src="[GIF URL aus der API Response]" draggable="false">
```

[5] `https://developer.mozilla.org/docs/Web/API/notification`
[6] `https://developer.mozilla.org/en-US/docs/Web/API/Notifications_API/Using_the_Notifications_API`
[7] `https://giphy.com/`
[8] `https://developers.giphy.com/docs/api`
[9] `https://developers.giphy.com/docs/api/endpoint#search`

in den Chat eingefügt werden. Diesmal wird kein HTML oder CSS vorgegeben, es ist Ihre Aufgabe ein entsprechendes Layout zu implementieren. Um weniger Bandbreite zu belegen und die GIFs schneller zu laden sollen zur Anzeige während der Eingabe *nicht bewegte* und *herunterskalierte* GIFs verwendet werden. Versenden Sie aber schließlich *bewegte*, größere Darstellungen des GIFs.

Zusätzliche Hinweise:

- Der API-Zugriff unterliegt gewissen Beschränkungen: Es findet ein sogenanntes *Rate Limiting* statt, das bedeutet, dass nur eine (unbekannte) Anzahl von Anfragen pro Zeiteinheit zugelassen wird. Beachten Sie dies, falls während der Bearbeitung der Aufgabe Fehler auftreten.
- Leerzeichen in einem Suchbegriff müssen durch das Zeichen `+` ersetzt werden.

2.7 Sicherheit von Webanwendungen

Als letzte Teilaufgabe sollen Sie die Sicherheit der von Ihren entwickelten Webanwendung bewerten.

- Ist sie anfällig für *Cross-Site-Scripting* Angriffe (XSS, siehe Tafelübung)?

  Ja: Führen Sie einen entsprechenden Angriff aus. Überlegen Sie sich effektive Gegenmaßnahmen (diese müssen *nicht* implementiert werden).

  Nein: Begründen Sie, warum der Angriff nicht möglich ist.

Letzter Commit bis zum 27. November.
Termine für die Vorstellungen per Terminklick ab 27. November.

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