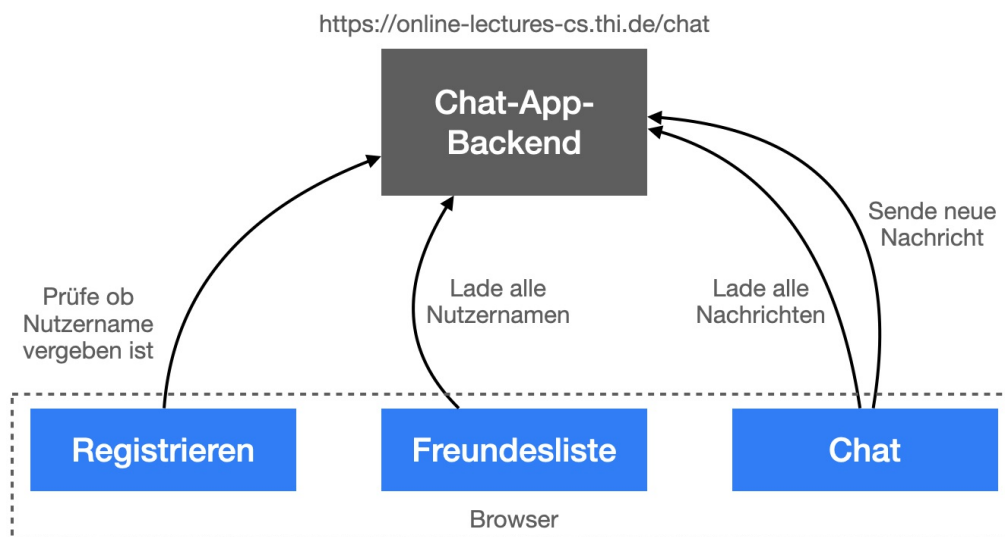


Aufgabe 3: Chat-App JS

Mit Aufgabe 3 sollen für die Chat-App verschiedene dynamische Elemente mittels JavaScript umgesetzt werden: Chatten, Formularvalidierungen und Vorschläge bei der Suche nach Freunden. Ziel ist dabei, dass die Chat-App mit Dummy-Daten an einigen Stellen einen funktionierenden Chat abbildet.

Um dieses Ziel zu realisieren, ist die Einbindung eines externen Dienstes notwendig. Dieser erlaubt das Versenden und Auslesen von Nachrichten für einen bestimmten Nutzer sowie die Suche nach Nutzernamen. In dieser Aufgabe soll unter Verwendung von AJAX-Aufrufen und der DOM-API der Dienst eingebunden und die HTML-Ansicht verändert werden.



Experimente

In diesem Abschnitt finden Sie empfehlenswerte Experimente, welche Sie ausprobieren können, um ein Verständnis über einzelne Mechanismen zu erhalten, die in dieser Aufgabe Verwendung finden.

DOM

Erstellen Sie eine neue HTML-Datei. Erzeugen Sie eine Grundstruktur und ergänzen Sie im Body-Bereich einen Script-Tag. Fügen Sie den folgenden JS-Code in den Tag ein.

```
let element = document.createElement("p");  
element.innerText = "Hallo, Welt!";  
document.body.appendChild(element);
```

Öffnen Sie das Beispiel im Browser. Zu sehen ist, dass im Browser der Text Hallo, Welt! angezeigt wird. Wenn Sie den Text mit dem Inspektor untersuchen (rechte Maustaste > untersuchen), sehen Sie, dass hier ein neues HTML-Element im Body-Tag ergänzt wurde.

Der Aufruf `document.createElement` erlaubt es, mit JavaScript neue HTML-Elemente zu erzeugen. Dies funktioniert für alle aus der Vorlesung bekannten HTML-Elemente (z.B. Listen, Links, Tabellen, usw.). Über den Aufruf von `appendChild` auf einem anderen HTML-Element kann das neu erzeugte Element einem DOM-Knoten hinzugefügt werden. Im Beispiel wurde der globale Objektverweis auf den Body-Tag verwendet, um hier das Element anzufügen.

Versuchen Sie anschließend eine Liste mit JavaScript zu erzeugen. Erzeugen Sie hierfür mit `createElement` die entsprechenden Elemente (`ul` und `li`) und fügen Sie das `ul`-Element in das `body`-Element ein und das `li`-Element in das `ul`-Element.

Erzeugen Sie abschließend einen Link und nutzen Sie die Methode `setAttribute`, um das Attribut `href` am Link zu modifizieren.

CSS

Erstellen Sie eine neue HTML-Datei. Erzeugen Sie eine Grundstruktur und ergänzen Sie im Body-Bereich einen Script-Tag. Fügen Sie den folgenden JS-Code in den Tag ein.

```
let element = document.createElement("p");
element.innerText = "Hallo, Welt!";
element.style.color = "white";
element.classList.add("someclass");
document.body.appendChild(element);
document.body.style.backgroundColor = "gray";
```

Öffnen Sie das Beispiel im Browser. Zu sehen ist, dass im Browser der Text Hallo, Welt! angezeigt wird. Der Text wird in weiß angezeigt, das Fenster selbst ist grau. Wenn Sie den Inspektor nutzen, um das Element zu untersuchen, werden Sie feststellen, dass im HTML-Dokument das Element `<p style="color: red" class="someclass">Hallo, Welt!</p>` ergänzt wurde und das Body-Element ein Style-Attribut erhalten hat.

Sie können erzeugte und existierende HTML-Elemente mit JavaScript jederzeit verändern. Nutzen Sie das Skript oder die Websuche, um entsprechende Attribute zu recherchieren.

AJAX

Erstellen Sie eine neue HTML-Datei. Erzeugen Sie eine Grundstruktur und ergänzen Sie im Body-Bereich einen Script-Tag. Fügen Sie den folgenden JS-Code in den Tag ein.

```
var xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange = function () {
    if (xmlhttp.readyState == 4) {
        if(xmlhttp.status == 200) {
            console.log(xmlhttp.responseText);
        } else {
            console.error("Fehler (" + xmlhttp.status + "): "
                + xmlhttp.responseText);
        }
    }
};
xmlhttp.open("GET", "https://online-lectures-cs.thi.de/chat/test.txt", true);
xmlhttp.send();
```

Das Beispiel lädt die Daten, die über die URL `https://online-lectures-cs.thi.de/chat/test.txt` bereitgestellt werden. Dies könnte ein Text, ein HTML-Dokument, ein Bild oder ein JSON-Dokument sein. In diesem Fall ist es einfacher Text. Sie sollten das Ergebnis des Aufrufs in der Konsole sehen.

Zur Kontrolle kann die URL direkt im Browser eingegeben werden, um zu sehen, dass sich hinter der URL ein einfaches Text-Dokument verbirgt. Diese Mechanik wird im Kontext von JavaScript Asynchronous JavaScript and XML (AJAX) genannt. Sie dient jedoch nicht nur zum Laden von XML-Dokumenten, wie der Name vermuten lässt. Genauer dazu finden Sie im Skript im entsprechenden Kapitel.

Verändern Sie die URL auf `https://online-lectures-cs.thi.de/chat/notfound`. Dies sollte dazu führen, dass einen Hinweis in der Konsole angezeigt wird, der besagt, dass ein Fehler 404 (*http error code*) mit dem Hinweis *Collection not found* aufgetreten ist. Ergänzend wird die Information als JSON-Dokument bereitgestellt.

Ändern Sie die Anfrage auf `https://online-lectures-cs.thi.de/chat/test.txt` und fügen Sie in Ihrem Test-Dokument im Body-Tag einen Div-Container hinzu, der ein Id-Attribut erhält, z.B. `myDiv`. Suchen Sie nach der Zeile `console.log(xmlhttp.responseText);` nach dem HTML-Element im DOM. Hierfür könnten Sie z.B. `document.getElementById()` nutzen. Das Ergebnis ist in diesem Fall ein Objektverweis auf das HTML-Element im DOM. Über das Node-Attribut `innerText` können Sie z.B. den Inhalt des HTML-Elements verändern.

```
// ...
console.log(xmlhttp.responseText);
const element = document.getElementById('myDiv');
element.innerText = xmlhttp.responseText;
// ...
```

Aktualisieren Sie die Webseite mit dem Refresh-Button des Browsers oder der Tastenkombination *Str + R*. Der Test-Text sollte daraufhin nicht nur in der Konsole, sondern ebenso im Browser-Fenster angezeigt werden.

(Optional) Kombinieren Sie Ihr Wissen aus den anderen Experimenten, um den geladenen Text in einen Paragraphen einzufügen und eine Veränderung des Layouts über CSS zu erreichen.

AJAX mit JSON

Erstellen Sie eine neue HTML-Datei. Erzeugen Sie eine Grundstruktur und ergänzen Sie im Body-Bereich einen Script-Tag. Fügen Sie den folgenden JS-Code in den Tag ein.

```
var xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange = function () {
    if (xmlhttp.readyState == 4) {
        if(xmlhttp.status == 200) {
            console.log(xmlhttp.responseText);
            let data = JSON.parse(xmlhttp.responseText);
            console.log(data);
        } else {
            console.error("Fehler (" + xmlhttp.status + "): "
                + xmlhttp.responseText);
        }
    }
};
xmlhttp.open("GET", "https://online-lectures-cs.thi.de/chat/test.json", true);
xmlhttp.send();
```

Das Beispiel lädt ein JSON-Dokument, welches ein Array von Nachrichten repräsentiert. JSON bedeutet JavaScript Object Notation und ist ein Format wie Daten in lesbarer Form für die Übertragung über HTTP repräsentiert werden können. Mithilfe des Aufrufs `JSON.parse` kann der JSON-Text verarbeitet werden, um damit ein JavaScript-Objekt zu erzeugen. Die Notation, die sich hinter JSON verbirgt, ist die Notation, die in JavaScript verwendet werden kann, um Objekte zu erzeugen. Sie können dies ausprobieren, indem Sie die URL im Browser öffnen und den angezeigten Text kopieren und in einem Script-Tag verwenden, zum Beispiel:

```
var data = ... hier den Inhalt der URL ...;
console.log(data);
```

Das Ergebnis ist ein valider JavaScript-Code.

Verarbeiten Sie anschließend das Array in der Variable `data` und erzeugen Sie damit eine HTML-Liste (vgl. DOM-Experiment).

AJAX und Formular-Validierung

Erstellen Sie eine neue HTML-Datei. Erzeugen Sie eine Grundstruktur und ergänzen Sie im Body-Bereich ein Formular mit Eingabe-Feld und Button wie im folgenden Beispiel.

```
<form id="my-form">
  <input name="check">
  <button>Submit</button>
</form>
```

Ziel des Experiments ist es, die Eingabe über AJAX zu prüfen, bevor das Formular abgesendet wird. Hierbei kommt die besondere Schwierigkeit hinzu, dass die Antwort immer zeitlich verzögert eingeht. Für den Test verwenden wir eine Test-URL, die eine Eingabe abgleicht: `https://online-lectures-cs.thi.de/chat/test/<expected>/<current>`. Ersetzen Sie `expected` durch einen erwarteten Wert, z.B. `abc`, und `current` durch den eingegebenen Wert. Der Server antwortet mit 200, wenn die Eingabe gleich ist und mit 404, wenn die Eingabe ungleich ist. Eine einfache `checkForm`-Funktion könnte jetzt wie folgt aussehen:

```
function checkForm() {
  const inputValue = document.querySelector('input[name="check"]').value;
  var xmlhttp = new XMLHttpRequest();
  xmlhttp.onreadystatechange = function () {
    if (xmlhttp.readyState == 4) {
      if(xmlhttp.status == 200) {
        console.log("Gleich");
      } else {
        console.log("Ungleich");
      }
    }
  };
  xmlhttp.open("GET",
    "https://online-lectures-cs.thi.de/chat/test/abc/" + inputValue, true);
  xmlhttp.send();
}
```

Um die Eingabe zu überprüfen würden sich verschiedene Ereignisse bei der Formularnutzung anbieten, einige wären:

- Verwenden des `onsubmit`-Attributes: `<form onsubmit="checkInput()">`
- Verwenden des `onclick`-Attributes: `<button onclick="checkInput()">Submit</button>`
- Verwenden des `oninput`-Attributes: `<input name="check" oninput="checkInput()">`

Die Schwierigkeit ist jetzt, wie Sie verhindern, dass das Formular abgesendet wird, wenn die Eingabe ungleich ist. Wenn Sie die Varianten ausprobieren, werden Sie feststellen, dass manchmal Ausgaben in der Konsole sichtbar werden und manchmal nicht. Das Formular wird aber nicht beeinflusst und immer abgesendet. Lösungsansätze für die drei vorgeschlagenen Wege wären:

- Bei Verwendung des `onsubmit`-Attributes: verhindern Sie, dass das Formular abgesendet wird, indem Sie als Ergebnis der `onsubmit`-Behandlung `false` zurückgeben,

z.B. mittels `<form onsubmit="checkInput(); return false">`. Das Formular kann, wenn die Prüfung erfolgreich ist, mittels `document.getElementById('my-form').submit();` in der `checkForm`-Funktion abgesendet werden.

- Bei Verwendung des `onclick`-Attributes: Ändern Sie den Button auf `type="button"`, damit dieser das Formular nicht absendet. Das Formular kann, wenn die Prüfung erfolgreich ist, mittels `document.getElementById('my-form').submit();` in der `checkForm`-Funktion abgesendet werden.
- Verwenden des `oninput`-Attributes: Merken Sie sich das Ergebnis der Prüfung in einer globalen Variable und prüfen Sie im Fall des submits ob die Variable den korrekten Wert hat. Dies kann mit Enable / Disable des Buttons kombiniert werden, damit dieser für den Nutzer entsprechend angezeigt wird.

In allen Varianten wird der zeitlichen Verzögerung begegnet in dem in die normalen Abläufe im Versand eines HTML-Formulars eingegriffen wird und dieses zu einem späteren Zeitpunkt (wenn die Antwort eingetroffen wird) manuell erlaubt, bzw. durchgeführt wird.

Nutzung des Backend-Servers

Der Chat-Server verwaltet eine beliebige Anzahl von Datenbereichen. Jeder Datenbereich hat eine eigene ID - z.B. `b0d6151a-11e4-40e9-8611-f2b677606daa`. Jedes Team sollte einen eigenen Datenbereich nutzen, um eigene Testdaten für die Entwicklung der Anwendung verfügbar zu haben.

Zur Erzeugung eines eigenen Datenbereichs durch den Chat-Server rufen Sie einfach diesen Link auf: [Dummy-Datensatz und Informationen \(https://online-lectures-cs.thi.de/chat/dummy\)](https://online-lectures-cs.thi.de/chat/dummy). Speichern Sie die unter **Test Chat Collection** gegebene Collection ID ab. Mithilfe der ID haben Sie von nun an Zugriff auf ihren eigenen Datenbereich auf dem Server!

Der für Sie erzeugte Datenbereich enthält initial zwei Benutzer **Tom** und **Jerry** mit sog. *Nutzer-Token*. Die Token benötigen Sie für die Bearbeitung der folgenden Aufgaben. Die Nutzer-Token sind Zugangsberechtigungen, mit denen Daten abgerufen werden können.

Zusätzlich werden auf der Seite die erforderlichen HTTP-Anfragen kurz erklärt und ein Beispiel zur Nutzung dargestellt. Für den aktuellen Stand der Aufgabe kann eines der verfügbaren Nutzertokens als konstanter Wert im Quellcode verwendet werden. Es ist empfehlenswert, sämtliche serverseitigen Informationen zentral zu definieren, um die Pflfegbarkeit der Information zu erleichtern. Eine Möglichkeit wäre, in jeder HTML-Seite eine JS-Datei namens `main.js` zu verlinken und in dieser Datei folgende Defintionen abzulegen:
Sie können die API-Dokumentation für Ihre Collection konkretisieren, indem Sie die ID mitgeben: `https://online-lectures-cs.thi.de/chat/dummy/91f60642-f8d4-42ec-a26d-84c7eb95dbc3`.

```
window.backendUrl = "https://online-lectures-cs.thi.de/chat/91f60642-f8d4-42ec-a26d-84c7eb95dbc3";  
window.token = "...z.B. das Token für Tom...";
```

Nutzen Sie dann zum Beispiel in den entsprechenden Aufrufen der XMLHttpRequest-Klasse die global verfügbaren Informationen:

```
const xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange = function () {
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        let data = JSON.parse(xmlhttp.responseText);
        console.log(data);
    }
};
// Chat Server URL und Collection ID als Teil der URL
xmlhttp.open("GET", backendUrl + "/user", true);
// Das Token zur Authentifizierung, wenn notwendig
xmlhttp.setRequestHeader('Authorization', 'Bearer ' + token);
xmlhttp.send();
```

Obiger Code fragt den Server nach den definierten Benutzern, siehe auch <https://online-lectures-cs.thi.de/chat/dummy#list-users>. Nach Ausführung des obigen Codes erhalten Sie in der JS-Console folgende Ausgabe:

```
Array(2)
0: "Tom"
1: "Jerry"
length: 2
```

Für alle nachfolgenden eventuell notwendigen Tests (z.B. Chats) können Sie jederzeit auf einen Helper zurückgreifen, der die wichtigsten Features umsetzt. Hier können Sie z.B. immer einen anderen Nutzer einloggen und Freundschaften erzeugen / akzeptieren / entfernen bzw. Chat-Nachrichten absenden. Den Helper finden Sie unter:

<https://online-lectures-cs.thi.de/chat/helper>

Vorbereitung

Nutzen Sie das Ergebnis aus Aufgabenblatt 2 und öffnen Sie dieses in Visual Studio Code. Die Arbeiten sind im Team aufzuteilen. **Je Team-Mitglied ist eine der folgenden Teilaufgaben umzusetzen und zu präsentieren.** Für Teams mit zwei Mitgliedern entfällt die Teilaufgabe a.

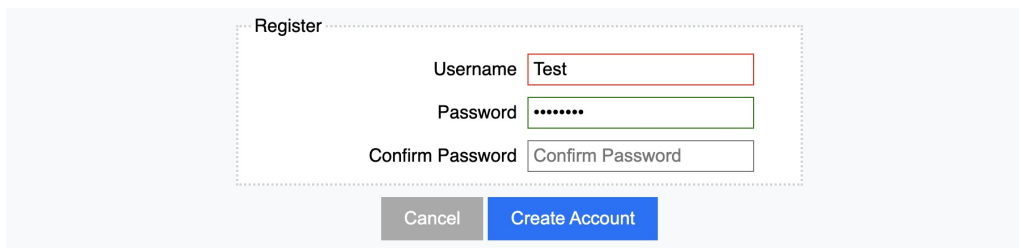
Teilaufgaben a: Registrieren

Das Formular zum Registrieren soll die Eingabemöglichkeiten überprüfen und durch entsprechende Fehlermeldungen visualisieren, ob die Eingabe korrekt ist oder Fehler vorliegen. Solange Fehler vorliegen, soll das Formular nicht abgesendet werden können. Zu überprüfen sind die folgenden Aspekte:

- Der gewählte Nutzernamen soll min. drei Zeichen lang sein

- Der gewählte Nutzername darf noch nicht verwendet worden sein, nutzen Sie hierfür aus den Dummy-Datensatz und Informationen die Server-Funktion User Exists
- Das Passwort muss min. 8 Zeichen haben
- Die Passwort-Wiederholung muss dem Passwort entsprechen

Die visuelle Anzeige, ob die Eingabe korrekt ist oder Fehler vorliegen soll mittels CSS-Klassen-Wechsel am Eingabefeld umgesetzt werden (vgl. Experimente). Sie können hier zum Beispiel die Farbe des Rahmens auf grün bzw. rot wechseln.



The image shows a 'Register' form with three input fields: 'Username' containing 'Test', 'Password' with masked characters, and 'Confirm Password' with the placeholder text 'Confirm Password'. Below the fields are two buttons: 'Cancel' and 'Create Account'. The form is enclosed in a dashed border.

Hinweis: Recherchieren Sie nach Möglichkeiten, um mittels onsubmit-Ereignis das Absenden des Formulars zu verhindern.

Teilaufgaben b: Mögliche Freunde und Freundesliste

In dieser Teilaufgabe sollen folgende Features entwickelt werden:

1. Hinzufügen neuer Freunde durch Eingabe bzw. Auswahl eines existierenden Freund-Namens
2. Dynamisches Aktualisieren der Freundesliste
3. Dynamisches Aktualisieren der Liste der Freundschaftsanfragen

Aufgabe b1: Hinzufügen neuer Freunde

In der Seite mit der Freundesliste soll bei der Eingabe eines Nutzernamens im Formular zum Hinzufügen von neuen Freunden eine Liste mit möglichen Optionen unter der Eingabe angezeigt werden. Diese Liste soll sich mit jedem Tastaturanschlag aktualisieren und entsprechend filtern. Ein Klick auf einen Vorschlag in der Liste soll den Wert im Formularfeld einsetzen. Nutzen Sie hierfür den Dummy-Datensatz und die Informationen der Backend-Funktion List Users.



The screenshot shows a web form with a text input field containing the text 'Tes'. Below the input field is a list of options: 'Test', 'Test1', 'Test2', 'Test3', 'Test4', and 'Test5'. To the right of the input field is a grey button labeled 'Add'.

Zur Vereinfachung Ihrer Implementierung können Sie folgendes Konstrukt nutzen:

```
<input placeholder="Add Friend to List" name="friendRequestName"
      id="friend-request-name" list="friend-selector">
<datalist id="friend-selector">
  <option>Tom</option>
  <option>Jerry</option>
  <!-- weitere Einträge -->
</datalist>
<button type="button">Add</button>
```

Die datalist ist über ihre ID an das input-Element gebunden und muss die erlaubten Nutzernamen enthalten. Das oben geforderte Selektions-Verhalten wird durch den Einsatz einer datalist automatisch gewährleistet. Ihre Aufgabe ist es entsprechend, das datalist-Element per JavaScript mit option-Elementen zu befüllen, nachdem Sie die Namen vom Backend geladen haben.

Achtung: in der Freundesliste angezeigte Benutzer sowie der aktuell eingeloggte Benutzer (z.B. "Tom", wenn Sie dessen Token verwenden) sollten nicht auswählbar sein!

Wenn der Add-Button geklickt wird, sollte das Backend per AJAX aufgerufen werden, um eine Freundschaftsanfrage zu erzeugen - siehe <https://online-lectures-cs.thi.de/chat/dummy#friend-request>. Es ist zuvor zu prüfen, dass der Nutzername in der Nutzerliste existiert, jedoch nicht bereits zu den Freunden gehört und auch nicht dem aktuellen Nutzer entspricht.

Aufgabe b2: Periodische Aktualisierung der Freundesliste und der Freundschaftsanfragen

Regelmäßige Aktualisierungen können in JavaScript mittels der Funktion `window.setInterval(callback, time)` realisiert werden. Die Funktion `setInterval` erwartet eine JavaScript-Funktion die mit dem jeweiligen Zeitereignis ausgeführt werden soll und eine Wiederholungsrate (in Millisekunden). Die erstmalige Ausführung beginnt erst nach erstmaligen Ablauf der angegebenen Zeit. Nutzen Sie folgendes Beispiel, welches jede Sekunde die Freundesliste neu lädt:

```
window.setInterval(function() {  
    loadFriends();  
}, 1000);  
loadFriends(); // erstmaliger Aufruf
```

Die Funktion `loadFriends()` ist von Ihnen zu implementieren. Die Implementierung der Funktion soll die Freundesliste vom Backend (siehe <https://online-lectures-cs.thi.de/chat/dummy#list-friends>) laden, das Ergebnis analysieren und in die eigentliche Freundesliste und Freundschaftsanfragen aufteilen sowie den DOM entsprechend aktualisieren. Dabei gilt, jeder Eintrag mit dem Status `accepted` wird in der Freundesliste angezeigt, jeder Eintrag mit dem Status `requested` wird in der Liste mit Freundschaftsanfragen angezeigt. Obige Funktionalität wird jede Sekunde im Hintergrund ausgeführt. Für die Einträge in der Freundesliste ist eine Verlinkung (wie im HTML-Prototypen vorgesehen) zur Chat-Ansicht notwendig. Implementieren Sie diese Funktion so, dass ein Query-Parameter mit dem angeklickten Nutzernamen übergeben wird und stimmen Sie sich im Team ab, wie dieses Parameter genannt wird, z.B.:

```
a.setAttribute("href", "chat.html?friend=" + friend.username);
```

Testen Sie die Gesamtfunktionalität der Seite z.B. mit folgendem Szenario:

- Start mit leerer Freundesliste, eingeloggter Benutzer ist "Tom" (entsprechendes Token wählen), Nutzer sind "Tom" und "Jerry"
- Die Auswahlliste sollte nur die bestehenden Nutzer enthalten, ohne "Tom"
- Auswahl/Eingabe des Freundes "Tom", Klick auf "Add"-Button → Eingabefeld wird rot umrandet (=Fehler)
- Auswahl/Eingabe von "Jerry", Klick auf "Add"-Button → "Jerry" wird als Freund hinzugefügt (Backend-Aufruf!) und durch die automatische Aktualisierung in der Freundesliste angezeigt
- Check der Auswahlliste: enthält nun keine Einträge mehr!

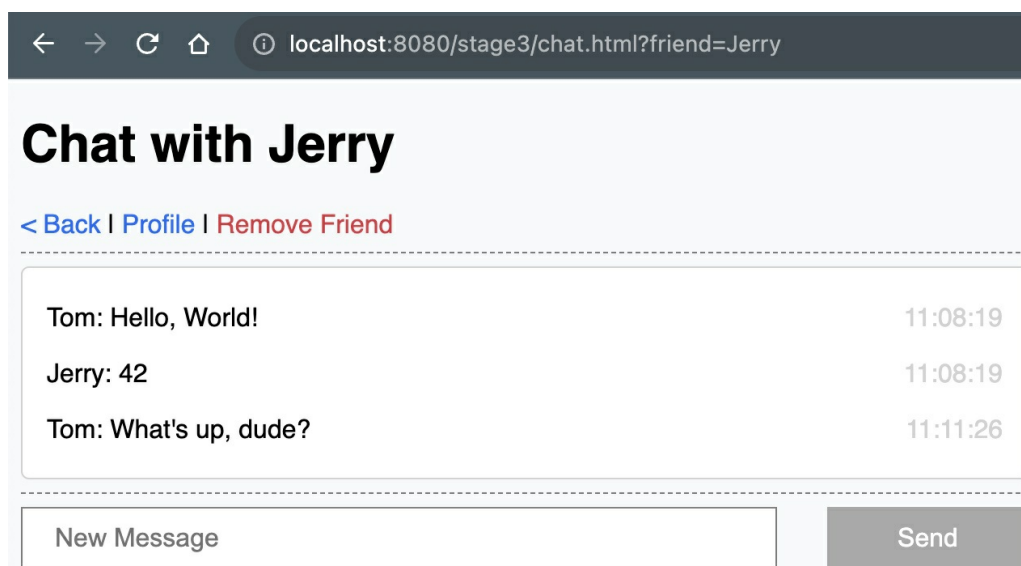
Teilaufgaben c: Chat-Nachrichten laden und versenden

In der Ansicht für den Chat sind die Nachrichten für den Chat-Verlauf zu laden und neue Nachrichten abzusenden. Nutzen Sie hierfür den Dummy-Datensatz und die Informationen der Server-Funktionen `List Messages` und `Send Message`. Der Chat-Verlauf soll darüber hinaus jede Sekunde neugeladen werden, um eventuelle Aktualisierungen zu visualisieren.

Die Freundesliste soll die Chat-Ansicht über einen Link aufrufen, welcher Informationen zu dem Chat-Partner enthält (vgl. Query-Parameter). Stimmen Sie sich über die Benennung ab. In der Chat-Ansicht muss zu Beginn aus dieser aufgerufenen Url der Query-Parameter (z.B. `friend`) gelesen werden, um die Nachrichten zwischen dem aktuellen Nutzer "Tom" und dem gewünschten Chatpartner zu laden. Hierfür können Sie z.B. die folgende Funktion nutzen. Sie gibt den Namen des Chatpartners zurück.

```
function getChatpartner() {  
    const url = new URL(window.location.href);  
    // Access the query parameters using searchParams  
    const queryParams = url.searchParams;  
    // Retrieve the value of the "friend" parameter  
    const friendValue = queryParams.get("friend");  
    console.log("Friend:", friendValue);  
    return friendValue;  
}
```

Wird nun in der Freundesliste "Jerry" zum Chat ausgewählt, so sollte in etwa folgendes Bild erscheinen. Bitte beachten Sie auch die angepasste Überschrift mit dem Namen des Chatpartners. Zunächst sollten nur die ersten beiden Nachrichten angezeigt werden, welche im Dummy-Datensatz automatisch enthalten sind. Die letzte Nachricht wurde zu den beiden im bereits existierenden Nachrichten per Eingabe erzeugt, zum Backend gesendet und durch das periodische Laden der Nachrichten angezeigt.



Achtung: Der "Send"-Button sollte darf kein Submit-Button sein, sonst wird die Seite ggf. ohne den Query-Parameter neu geladen und funktioniert nicht mehr korrekt, da der Chatpartner nicht bestimmt werden kann!

Bewertungskriterien

- Je Team-Mitglied muss eine Funktionalität / Teilaufgabe umgesetzt werden und wie beschrieben funktionieren
- Schreiben Sie ein- oder mehrere JS-Dateien, platzieren Sie die umgesetzten Funktionen nicht in die HTML-Dokumente
- HTML-, Bild-, CSS- und JS-Dateien müssen relativ verlinkt werden