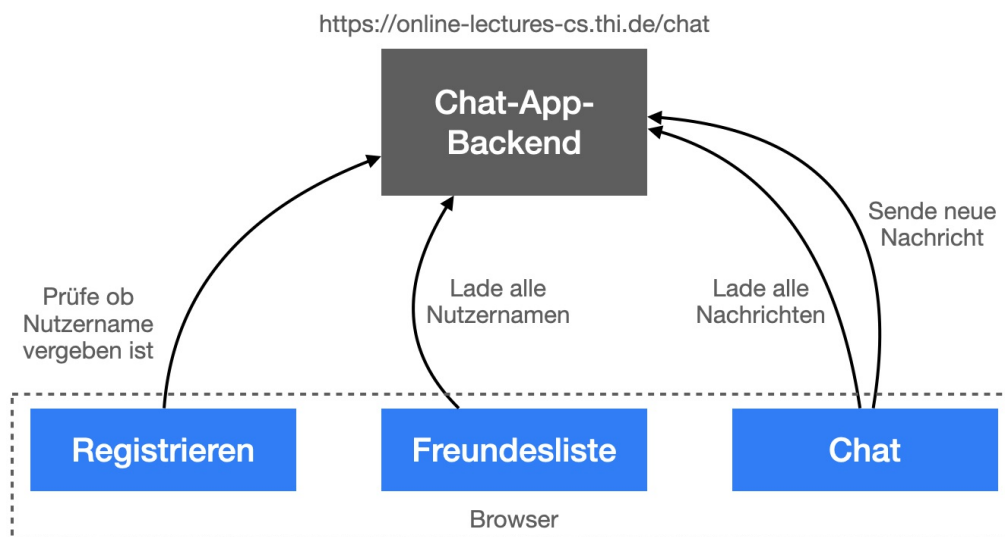


Aufgabe 3: Chat-App JS

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-Element 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).

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.**

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. Später wird dieses Token durch die Angular- (nur INF!) bzw. PHP-Anwendung bereitgestellt, daher empfiehlt es sich in den jeweiligen HTML-Dokumenten ein unabhängiges Skript-Element zu platzieren, welches die Informationen bereitstellt:

```
<script>
  window.chatToken = "...";
  window.chatCollectionId = "...";
  window.chatServer = "https://online-lectures-cs.thi.de/chat";
</script>
```

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

```
var 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", window.chatServer + "/" + window.chatCollectionId +
"/user", true);
// Das Token zur Authentifizierung, wenn notwendig
xmlhttp.setRequestHeader('Authorization', 'Bearer ' + window.chatToken);
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:

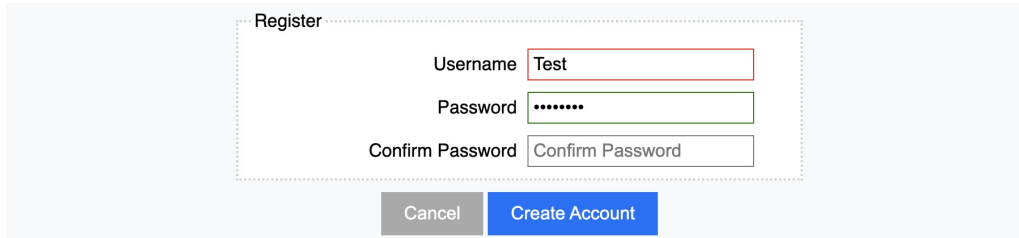
```
Array(2)
0: "Tom"
1: "Jerry"
length: 2
```

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 Nutzername 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' containing seven dots, and 'Confirm Password' which is empty. The 'Username' field has a red border, while the 'Password' field has a green border. Below the fields are two buttons: 'Cancel' and 'Create Account'.

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

Teilaufgaben b: Mögliche Freunde

In der Freundesliste soll bei der Eingabe eines Nutzernames 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 aus den Dummy-Datensatz und Informationen die Server-Funktion List Users.

Es ist darauf zu achten, dass das Formular nur abgesendet werden kann, wenn der Nutzername in der Liste existierte. Für die Erzeugung der Liste sind Methoden zur Manipulation und Erweiterung des DOM notwendig (vgl. Experimente).

Test

Test1

Test2

Test3

Test4

Test5

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 aus den Dummy-Datensatz und Informationen die Server-Funktion List Messages und Send Message. Der Chat-Verlauf soll darüber hinaus jede Sekunde neugeladen werden, um eventuelle Aktualisierungen zu visualisieren.

Damit Nachrichten an die passende Stelle im Dokument eingefügt werden können, bietet es sich an, das gewünschte HTML-Element geeignet zu erweitern, damit mit den Methoden in JavaScript das passende Element gefunden werden kann.

Regelmäßige Aktualisierungen können in JavaScript mittels der Funktion `window.setInterval(callback, time)` realisiert werden. Die Funktion 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 eine Nachricht in die Konsole platziert:

```
window.setInterval(function() {  
    console.log("Hallo, Welt!");  
}, 1000);
```

Chat with Tom

[< Back](#) | [Profile](#) | [Remove Friend](#)

Test4: Test 19:51:13

Test4: Test2 15:24:23

Bewertungskriterien

- Je Team-Mitglied muss eine Funktionalität / Teilaufgabe umgesetzt werden und wie beschrieben funktionieren
- Lagern Sie die Informationen für den Zugriff (Server, Collection ID und Token) aus (nicht in den JavaScript-Dateien)
- 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