AJAX

> AJAX is a developer's dream, because you can:
>
> - Update a web page without reloading the page
> - Request data from a server - after the page has loaded
> - Receive data from a server - after the page has loaded
> - Send data to a server - in the background

AJAX ermöglicht **asynchrones Übertragen** von Daten zwischen Client und Server, ohne dass die Seite neu geladen werden muss.

My AJAX Chatroom

Ziel dieser Übung ist es, den Chatroom des Node.js Express Chatserver mit AJAX umzusetzen.

Anforderungen:

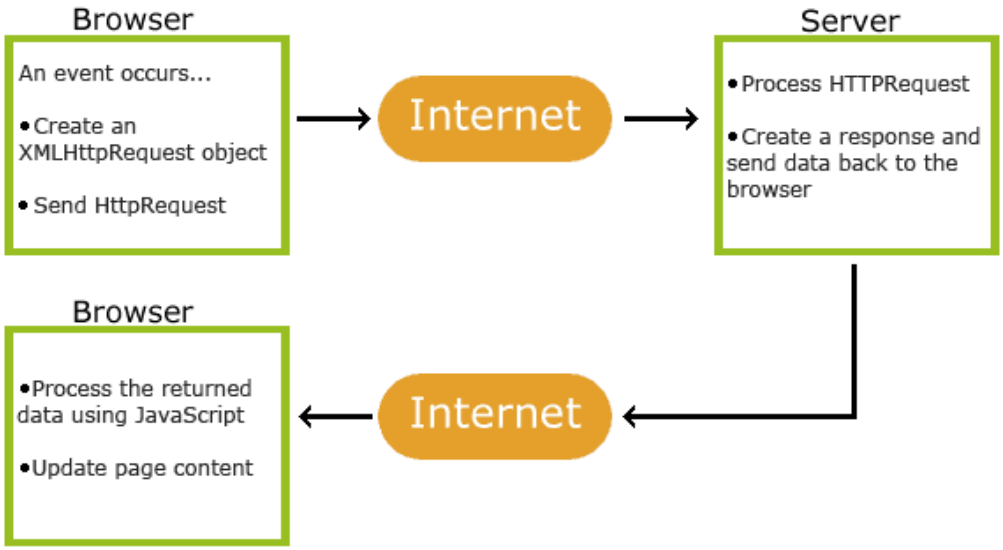
- 1:1 Klon von MyExpressChatServer
- Anforderungsdetails des Chatrooms
  - Das Chatroom Frontend ist wie alle anderen HTML Dateien mittels HTTP GET Request unter Angabe der Dateiendung erreichbar: *localhost:3000/chat.html*. Dafür soll keine separate Route definiert sein, die Datei wird über *express.static* geliefert.
    Es wird somit immer dieselbe HTML Datei mit allen Daten außer den einzelnen Chatnachrichten retourniert (der Chatroom ist also zu diesem Zeitpunkt leer).
  - Die Nachrichten des Chatrooms werden mit AJAX über die API Schnittstelle *localhost:3000/chat* nachgeladen und mit JavaScript in den Chatroom eingefügt.
  - Die Nachrichten werden wie bisher über den HTTP POST Request */chat* gespeichert.
    Dies soll aber ebenfalls mit AJAX passieren (d.h. es wird auf keine andere Seite weitergeleitet)
  - Die Route *localhost:3000/chat/<id>* ist unverändert die Schnittstelle für die REST API.

Verzeichnis:

Tutorial 1: Server für AJAX vorbereiten

Das Ziel dieser Übung ist die Chatnachrichten dynamisch nach dem HTML Seitenaufbau nachzuladen. Dazu verwenden wir AJAX (**A**synchronous **J**avaScript **A**nd **X**ML). AJAX benutzt das XMLHttpRequest Object (das jeder Browser implementiert hat) zum Kommunizieren mit dem Server, sowie JavaScript und DOM zum dynamischen Verändern der HTML Seite. In unserem Fall bedeutet das:

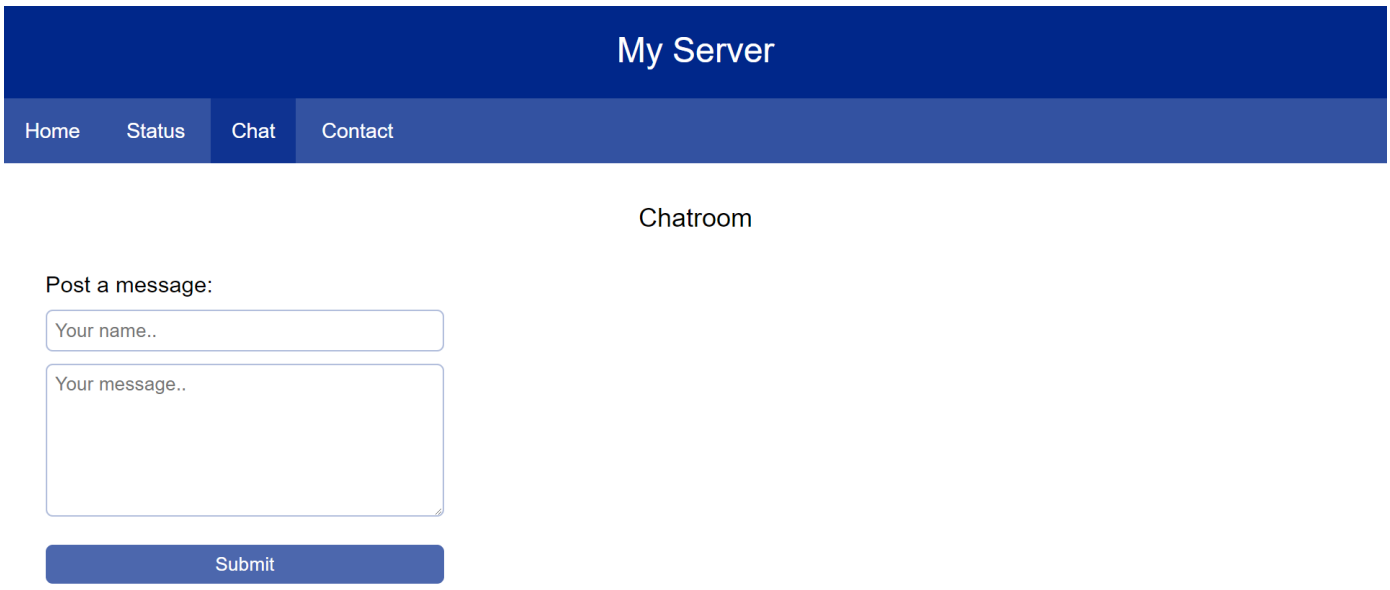
- Mit dem XMLHttpRequest Object werden die Chatnachrichten vom Server geladen
- Mit JavaScript und DOM werden die Nachrichten in die bestehende HTML Seite eingebettet und so dem User angezeigt



https://www.w3schools.com/xml/ajax_intro.asp

Let's begin!

Beim Aufruf von *localhost:3000* soll anfangs nur die HTML-Datei *chat.html* mit all ihren statischen Elementen geliefert werden, d.h. der Header mit Navigation, das Formularfeld und die Chatroom Überschrift. Die einzelnen Nachrichten sind noch nicht in der Datei vorhanden. Die Ausgabe sieht zu diesem Zeitpunkt so aus:



Was müssen Sie dazu tun? Sie benötigen nur mehr ein einziges *chat.html* File (das die obige Ausgabe liefert). Die Datei *chat2.html* wird nicht mehr benötigt.

Testen Sie den Code!
Der Chatroom soll beim Aufruf der URL *localhost:3000/chat.html* erscheinen (noch ohne Nachrichten).

Denken Sie daran, auch die Links in der Navigationsleiste anzupassen.
Der Chatroom ist nunmehr über *localhost:3000/chat.html* erreichbar (und nicht *localhost:3000/chat*)

Als nächsten Schritt müssen Sie die GET Request Route */chat* des Express Servers anpassen.

- Der Chatroom soll über die URL *localhost:3000/chat.html* erreichbar sein. Dazu müssen Sie nichts weiter tun – die Datei ist bereits wegen der Middleware *express.static('public/html')* erreichbar. Das haben Sie eben gerade auch getestet.
- Die Nachrichten sollen über die URL *localhost:3000/chat* nachgeladen werden. Dazu müssen Sie die GET Request Route */chat* etwas umprogrammieren:

  Es darf kein vollständiges HTML File mehr geliefert werden. Stattdessen sollen die Namen und die Nachrichten im JSON Format an den Client übermittelt werden.

Bauen wir also die Route `app.get('/chat', (req, res) => {` neu zusammen:
Der erste Vorteil von AJAX zeigt sich schon bei der ersten Änderung:
Sie müssen nicht mehr mehrere Files synchron einlesen (was der asynchronen Natur von Node.js widerspricht).
Stattdessen müssen Sie nur mehr die Daten aus der Textdatei *chat.txt* auslesen (asynchron!):

```
// chatroom: load posts
app.get('/chat', (req, res) => {
    let data = fs.readFile('./public/data/chat.txt', 'utf8',  function (err, data) {
        if (err) {
            res.status(404).send('Sorry, no messages stored.');
        }
        else {
            // parse data
            // build JSON
            // send JSON
        }
    });
});
```

Der wesentliche Teil, der *else* Block, ist fehlt noch!

Widmen wir uns nun dem *else* Block. Zuerst müssen Sie wieder die Textdaten parsen (d.h. aus der Datei auslesen und in ein leserliches Format konvertieren):

```
// parse data
let names = [];
let messages = [];
let lines = data.split('\n');
for(let i=lines.length-1; i>=0; i--) { // read from end --> newest message is on top
    let elements = lines[i].split( separator: ';');
    if (elements[1] !== undefined) {
        names.push(elements[0]);
        messages.push(elements[1]);
    }
}
```

in diesen Arrays werden die ausgelesenen Daten gespeichert

Die Zeile in Name und Nachricht aufteilen (wie bisher)

die ausgelesenen Daten in den Arrays ablegen

Der nächste Schritt ist schnell erledigt: dazu müssen Sie nur wissen, dass bei den key:value Paaren von JSON Objekten auch Arrays als Value übergeben werden dürfen. Warum? Weil in JavaScript im Hintergrund alles als Objekt implementiert ist (Zitat von Mosh Hamedani: „Everything is an object“).

```
// build JSON
let obj = {"names":names, "messages":messages};
```

Array mit allen Nachrichten

Array mit allen Namen

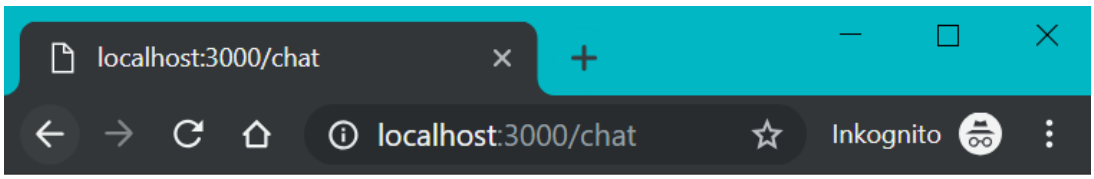
Der letzte Schritt: das fertige JSON Objekt senden.

```
// send JSON
res.send(JSON.stringify(obj));
```

JSON.stringify(obj) wird benötigt, weil über das Netzwerk nur ein String und kein Objekt gesendet werden darf. Das bedeutet:

- Der Server konvertiert das JSON Objekt in einen JSON String
- Der Client parst den JSON String und erhält wieder ein JSON Objekt

Testen Sie den Code! Rufen Sie dazu die URL *localhost:3000/chat* auf.
Als Output müssen Sie im Browser den JSON String mit all Ihren gespeicherten Nachrichten sehen!



{"names":["Susi","Maxi","Susi","Maxi"],"messages":["Thanks, I'm fine.","How are you?","Hello out there!","Hello everyone!"]}

der übermittelte JSON String. Entdecken Sie die beiden Arrays?

Der Server ist fertig – nun liegt es am Client, die Daten abzufragen und anzuzeigen.

Tutorial 2: AJAX im Chatroom implementieren

Der Server ist soweit vorbereitet – jetzt muss der Client die Daten abfragen und in das bestehenden HTML Dokument einfügen.

Dazu benötigen Sie JavaScript. Legen Sie ein neues JS-File *chat_loading.js* an und verlinken Sie die Datei in der HTML Datei *chat.html*:

```
<script src="chat_loading.js" defer></script>
```

> **i** Das Attribut **defer** sorgt dafür, dass das Skript erst ausgeführt wird, nachdem die Seite vollständig geladen wurde und das DOM vollständig aufgebaut wurde.

Widmen wir uns nun der JavaScript Datei:
Definieren Sie eine Funktion *loadData()*. Deren Aufgabe ist mit AJAX die Chatdaten von der Route *localhost:3000/chat* abzufragen, die JSON Daten zu parsen und in das bestehende HTML einzubauen.

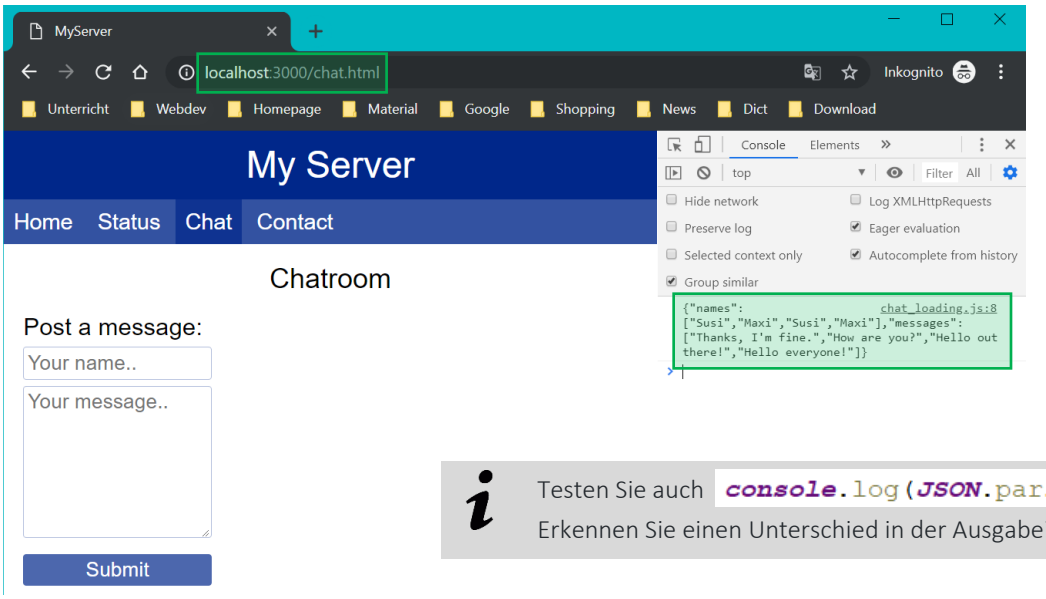
```
function loadData() {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function () {
        if (this.readyState === 4 && this.status === 200) {

            // parse JSON string to JSON object
            // build HTML Code
            // insert into DOM
        }
    };
    xhttp.open( method: "GET", url: "chat", async: true);
    xhttp.send();
}
loadData(); ruft Ihre Funktion auf
```

Callback Funktion:
wird ausgeführt, sobald die Antwort des Servers beim Client angekommen ist

die URL des Requests (hier ein relativer Verweis!)

AJAX unterstützt asynchrones Übertragen!

Testen Sie Ihren AJAX Request!
Fügen Sie dazu in der Callback Funktion ein simples `console.log(this.responseText);` ein.
In der Konsole muss dann der JSON String geloggt werden.



> **i** Testen Sie auch `console.log(JSON.parse(this.responseText));`
> Erkennen Sie einen Unterschied in der Ausgabe?

Kommen wir nun zum Code der Callback Funktion:

Schritt 1: der Server schickt einen JSON String. Dieser muss wieder in ein JSON Objekt konvertiert werden. Verwenden Sie dazu die *JSON.parse* Methode.

```
// parse JSON string to JSON object
let obj = JSON.parse(this.responseText);
```

Schritt 2: aus dem JSON Objekt müssen Sie den vollständigen HTML Code aller Chatnachrichten erstellen. Iterieren Sie dazu durch alle Nachrichten und bauen Sie den HTML Code Schritt für Schritt zusammen.

```
let chat_block = '';
for (i = 0; i < obj.names.length; i++) {
    if (obj.names[i] !== undefined || obj.messages[i] !== undefined) {
        chat_block += `<div><h3>${obj.names[i]}</h3><p>${obj.messages[i]}</p></div>`;
    }
}
```

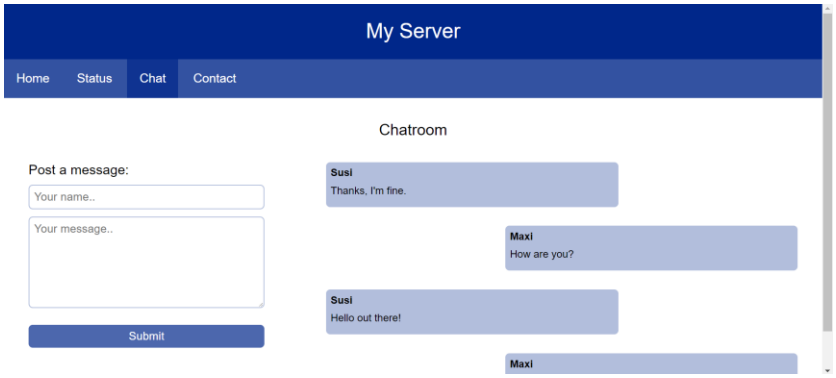
in diesem String wird der HTML Code gespeichert

Für jede Nachricht wird ein neuer Block angelegt, der den Namen und die Nachricht enthält.
Mit der Template String Notation bleibt der Code kompakt!

Schritt 3: den generierten HTML Code in das bestehende HTML Gerüst eingliedern.

```
// insert into DOM
document.getElementById( elementId: "chat").innerHTML = chat_block;
```

Testen Sie Ihren Code. Die Nachrichten müssen im Chatroom erscheinen!



Advanced: Nachrichten mit AJAX senden

Das Posten neuer Nachrichten ist noch etwas „holprig“, da jedes Mal auf eine Unterseite (*confirm* oder *error*) weitergeleitet wird.

Implementieren Sie folgende Verbesserung in einer neuen JS-Datei *chat_sending.js*:

- Neue Nachrichten werden per AJAX an die POST Route gesendet.

  Hilfestellung für Submit Button:

  Sie müssen den „originalen“ Submit Button ausblenden und einen neuen Button mit einem Event Listener einfügen (der beim Click den AJAX Request veranlasst). Verwenden Sie als neuen Button eine einfache Box (auf keinen Fall <button>, dies führt zu Fehler).

  Hilfestellung für einen POST AJAX Request:

```
xhttp.open( method: "POST",  url: "chat",  async: true);
xhttp.setRequestHeader( name: "Content-type",  value: "application/x-www-form-urlencoded");
xhttp.send( body: `name=${name}&message=${message}`);
```

- Die Bestätigung (oder die Fehlermeldung) wird direkt beim Formular eingeblendet.
- Nach dem erfolgreichen Speichern einer Nachricht wird der Chatroom mit der neuen Nachricht aktualisiert (ohne die Seite neu zu laden).

  Tipp: Sie können Ihre Funktion *loadData()* jederzeit wieder aufrufen.