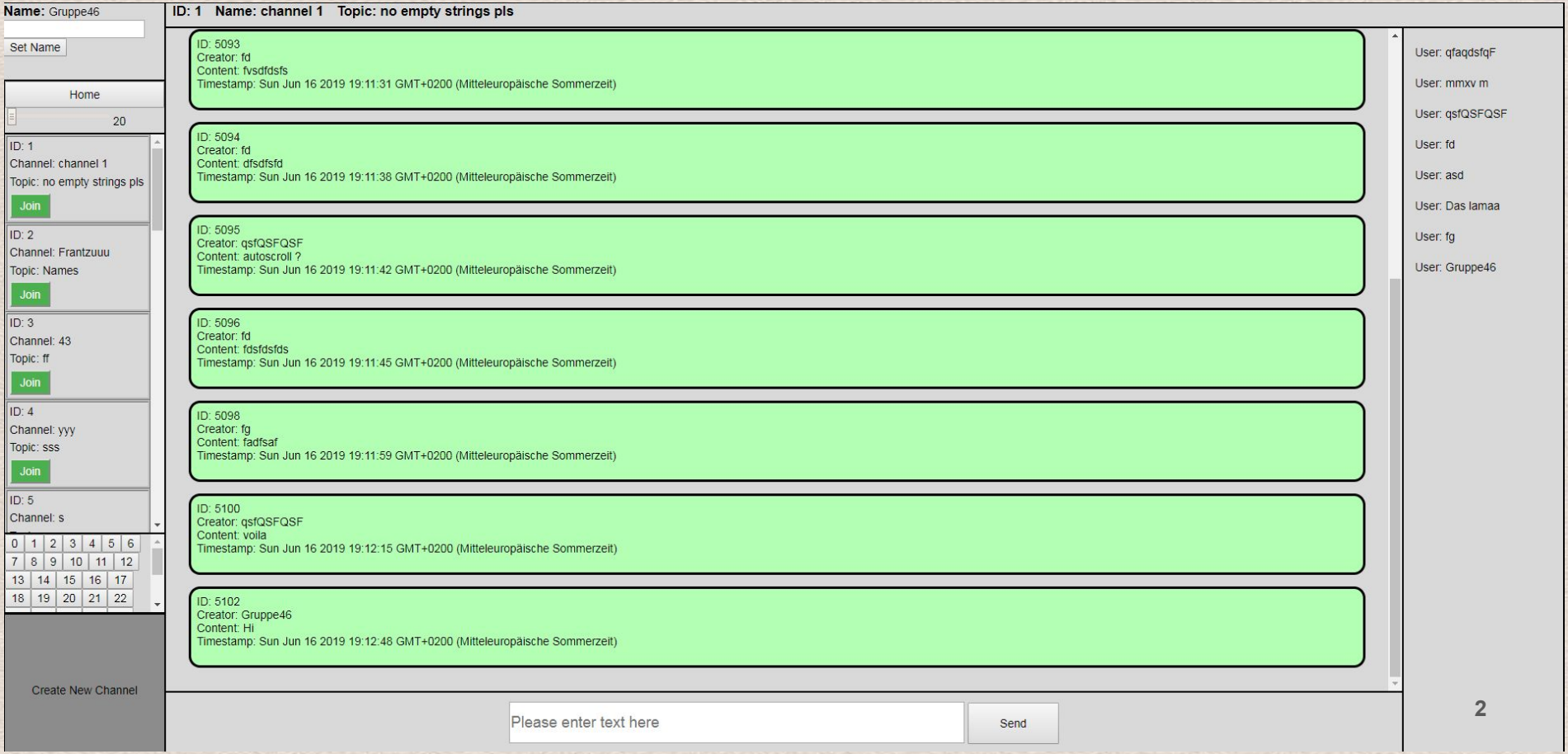
Anwendungssysteme

Programmieraufgabe 1 - WhatsChat

Gruppe 46 :

- Florian Tim Peters
- Erdem Arslan
- Sandra Basche
- Syed Danial Zahir

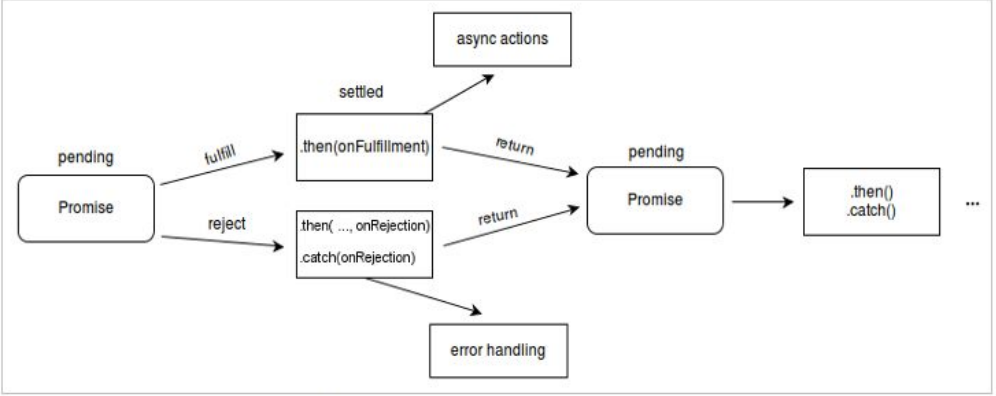
Screenshot der Webseite



Zusätzlich verwendete Objekte / API’s

Promise:

- Das Programm nimmt das "Promise" entgegen und muss nicht auf die Daten warten. Die Funktion kann zu einem späteren Zeitpunkt, sobald die Daten verfügbar sind, die Ausführung fortführen, bzw. stößt auf einen Fehler, falls diese nicht geliefert werden.



https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

Fetch:

- ist ein Web-Request mit ähnlicher Funktionalität wie XMLHttpRequest. Es macht es einfach, Optionen festzulegen und hat ein etwas anderes Error-Handling als jQuery.ajax() (kein reject durch HTTP Status-Codes sondern nur durch Netzwerkfehler). Ein fetch()-Aufruf gibt immer ein Promise zurück mit der eventuellen Antwort der Anfrage.

Aufbau der Seite (HTML/CSS)

- Die HTML-Elemente werden in Größe, Farbe und Anordnung mit Hilfe von CSS angepasst.
- Eine der Designentscheidungen war, dass alle funktionalen Elemente zu jeder Zeit sichtbar sein sollen.
- Dazu haben wir mit “overflow-wrap: break-word; overflow-x: hidden; overflow-y: auto;” dafür gesorgt, dass content mit veränderbarer Größe, wie z.B. die Channel Liste, immer im dafür vorgesehenen Bereich bleibt. Falls zu viele Channels an die Liste angehängt werden, erscheint ein Scrollbar, mit der, der Rest des contents erreichbar ist.
- Um die Elemente auf der Webseite zu positionieren, benutzen wir CSS-Grid, ein zweidimensionales Anordnungssystem, mit dem “grid-areas” definiert werden, denen bestimmte Elemente zugewiesen werden.
- Wir haben unsere Webseite in zwei Bereiche aufgeteilt (left-side, right-side)
- Diese werden in eigenständige div-Elemente definiert, welche im content-wrap enthalten sind

```
<body>
<div id="content-wrap">
    <!-- WRAPPER AROUND ALL PAGE CONTENT-->
    <div id="left-side-content" style="
        display: initial;">
        <!-- Nav menu on the left--> ...
    </div>

    <div id="right-side-content"> <!--
    Content box on the right--> ...
    </div>
<!--CONTENT-WRAP END-->
</div>
</body>
```

```
/*WRAPPER AROUND ALL CONTENT INSIDE BODY*/
#content-wrap {
    width: 100%;
    height: 100%;
    display: grid;
    grid-template-columns: 200px 1fr;
    grid-template-areas: "left right";
}
```

```
#right-side-content {
    width: 100%;
    height: 100%;
    grid-area: right;
    display: grid;
    grid-template-columns: 1fr 200px;
    grid-template-rows: 30px 1fr;
    grid-template-areas:
    "info info"
    "main users";
}
```

```
#left-side-content {
    width: 100%;
    height: 100%;
    grid-area: left;
}
```

Aufruf der Webseite und global deklarierte Variablen

```
// GLOBAL VARIABLES
const url = 'http://34.243.3.31:8080'
const auth = {'X-Group-Token' : '9GAqC9CWYLNH'}
var CL_size = 20
var CL_page = 0
var c_interval = null
var name = "user"
var activeChannel = -1
var m_timeout  = true
var m_interval = null
var ALL_M = {}
//! END GLOBAL VARIABLES

// EXECUTE ON PAGE LOAD
$(document).ready(function() {
    refreshCL(0)
    setUserName(name)
    c_interval = setInterval(refreshCL, 10000)
    refreshM()
});
//! END EXECUTE ON PAGE LOAD
```

Beim Laden der Webseite wird:

- Einmal die Channel Liste geupdatet
- Der Default-Name wird auf “user” festgelegt
- Mit der Funktion setInterval() wird alle 10 Sekunden die Channel Liste erneut geupdated
- Die Funktion refreshM() wird einmal aufgerufen

➔ Zu diesem Zeitpunkt macht refreshM() noch nichts. Der Aufruf startet aber einen Timer, womit die Funktion immer wieder, einmal die Sekunde aufgerufen wird.

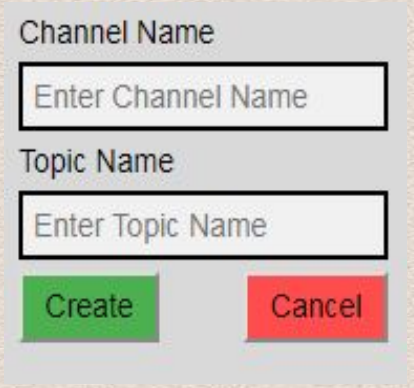
Channels erstellen

```html
<div id="c-create-w"> <!-- Create a new Channel-->
    <button id="toggle-button" onclick="openForm()">Create New Channel</button>
    <div id="toggle-w" style="display: none;">
        <form id="form-w" onsubmit="this.reset(); return false">
            <label for="c-new-n"><b>Channel Name</b></label>
            <input type="text" id="cNewN" placeholder="Enter Channel Name"required>

            <label for="c-new-t"><b>Topic Name</b></label>
            <input type="text" id="cNewT" placeholder="Enter Topic Name" required>

            <button type="submit" id="c-submit" onclick="createChannelA(cNewN.value, cNewT.value)">Create</button>
            <button type="button" id="c-cancel" onclick="closeForm()">Cancel</button>
        </form>
    </div>
</div>
```

```javascript
function createChannelA(name, topic) {
    createChannel(name, topic)
    .then(function(res) {
        closeForm()
        refreshCL()
        joinChannel(res.id)
    }, function(error) {
        // Name is already Taken?
        alert(error)
    })
}
```



- Der Benutzer kann Channels erstellen, indem er auf ‘Create New Channel’ klickt. Daraufhin öffnet sich ein Formular mit 2 Textfeldern, in der der Benutzer den Channel-Name und den Topic-Name eingeben kann.
- Die Funktion createChannelA() wird aufgerufen, sobald der Nutzer auf den Create (Submit) Button geklickt hat. Als Channel-Name und Topic-Name werden die Werte aus den beiden Textfeldern ‘cNewN’ und ‘cNewT’ genommen.
- Mit diesen Werten wird die API-Wrapper Funktion createChannel() aufgerufen

API-Wrapper Struktur

Die API-Wrapper Funktionen sind alle nach dem gleichen Muster aufgebaut :

- Es werden zuerst alle nötigen Optionen in einem Objekt gespeichert (die HTTP-Methode (GET vs. POST), Header und Body).
- Die global gespeicherte Back-End-URL wird mit allen weiteren Parametern (pflicht oder optional) angepasst.
- Dann wird ein fetch()-Request auf diese URL, mit den oben erstellten Optionen, gestartet.
- Auf das von fetch() zurückgegebene Promise wird mit .then() eine Funktion angewendet, die den Statuscode der Antwort prüft.
- Bei Erfolg wird die Antwort des Back-Ends zu einem JSON-Objekt umgewandelt und zurückgegeben.
- Bei einem Fehler wird ein Error geworfen.
- Die Funktion .then() gibt ein Promise zurück, dieses Promise wird letztendlich returned.

Channels erstellen

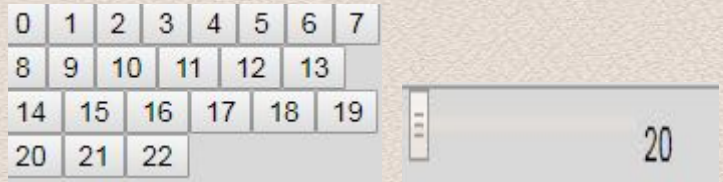
```
function createChannel(name, topic) {
    // POST create a new channel with name and topic
    let opts = {
        method : 'POST',
        headers : new Headers(Object.assign({}, auth, {'Content-Type' : 'application/json'})),
        body : JSON.stringify({'name' : name, 'topic' : topic})
    }
    return fetch(url + '/channels', opts)
    .then(function(res) {
        switch(res.status) {
            case 201:
                return res.json();
                break;
            case 409:
                throw new Error(`Channel Name already exists: ${res.status}`);
                break;
            default:
                throw new Error(`HTTP Status Error: ${res.status}`);
        }
    })
};
```

- Die createChannel() Funktion sendet einen **POST** an das Back-End.
- Wenn die API-Anfrage erfolgreich war, wird das json Objekt zurückgegeben und dann in createChannelA() die Channel Liste geupdatet und dem neuen Channel beigetreten.
- Falls nicht, wird die Fehlermeldung mit alert() ausgegeben.

Channels darstellen

```
function refreshCL(pageNr=CL_page) {
    CL_page = pageNr
    renderCL(getCL(pageNr, CL_size))
}
```

```
function CPLItem(i) {
    var pe        = document.createElement("input")
    pe.type       = "button"
    pe.className  = "p-list-elem"
    pe.value      = `${i}`
    pe.setAttribute("onclick", `refreshCL(${i})`)
    return pe
}
```



Channel Liste

- Um die Channel Liste zu aktualisieren, wird die refreshCL() Funktion aufgerufen. Der “pageNr” Parameter bestimmt welche Seite der Liste man laden möchte. Wenn er nicht angegeben wird, wird die gleiche Seite geladen die das letzte Mal benutzt wurde. Dafür wird die aktuelle Seite immer in der globalen Variable “CL_page” gespeichert.

- Um die aktuelle Seite zu wechseln, gibt es unter der Liste der Channels eine weitere Liste mit Buttons. Jeder Button ruft die refreshCL() Funktion mit jeweils einer anderen “pageNr” auf, sobald der Nutzer auf ihn klickt.

- refreshCL() ruft mit einer “pageNr” und “CL_size” die Funktion getCL() auf. “CL_size” ist eine globale Variable, die durch den Slider über der Channel Liste verändert werden kann. Sobald man den Slider loslässt wird der neue wert in “CL_size” gespeichert und die Channel Liste aktualisiert.
  Die Seite wird auf 0 zurückgesetzt, damit nicht eine Seite aufgerufen wird, welche nicht existiert.

Channels darstellen

Channel Liste

```
function getCL(page=0, size=CL_size) {
    // GET the latest list of channels from the backend
    let opts = {
        headers:new Headers(auth)
    }
    console.log("Updating CL from: " + url + `/channels?page=${page}&size=${size}`)
    return fetch(url + `/channels?page=${page}&size=${size}`, opts)
    .then(function(res) {
        switch(res.status) {
            case 200:
                return res.json();
                break;
            default:
                throw new Error(`HTTP Status Error: ${res.status}`);
        }
    })
};
```

```
function renderCL(pro) {
    pro.then(function(res) {
        // List of Channels
        var div_c = document.createElement("div")
        res._embedded.channelList.forEach(function(c) {
        div_c.append(CLItem(c.id, c.name, c.topic))
        })
        $("#CL").html(div_c.innerHTML)
```

```
        // Change max Slider length
        var tE = res.page.totalElements
        $(".slider").each(function() {
            this.max = (tE + (5 - (tE % 5)))
```

- getCL() ist eine API-Wrapper Funktion und sendet ein GET-Request an das Back-End. Falls die Anfrage nicht funktioniert, wird ein Error geworfen. Falls die Anfrage erfolgreich war, wird die Back-End Antwort an refreshCL() zurückgegeben, welche es dann an renderCL() weitergibt.

- Hier benutzen wir das erste Mal das Pattern, mit der wir fast alle HTML contents auf der Webseite anzeigen. Es wird zuerst ein neues Element erstellt. Dann werden in einer for-each Schleife alle (in diesem Fall Channels) JSON-Elemente in HTML Elemente umgewandelt (in CLItem()), und dann an das vorher erstellte Element angehangen. Sobald die Schleife fertig ist, ersetzt (nur!) der innere HTML-Text des Elements, den ganzen vorherigen Inhalt des Ziel Elements, welches bereits Teil der Website ist. Der Vorteil davon ist, dass es dadurch nur ein einziges Update der Website gibt, was ein “Flackern” verhindert.

- Dieses Pattern wird wiederholt, um auch die Channel-Page-List auf die Seite zu bringen. Als letztes wird noch die maximale Länge des Sliders geupdatet, da jeder Aufruf der Funktion auch eine Veränderung der maximalen Elemente zur Folge haben kann.

Channels darstellen

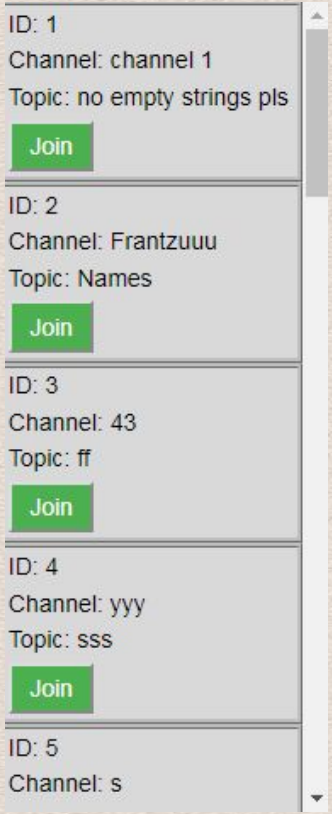
```
function CLItem(id, name, topic) {
    var li            = document.createElement("div")
    li.className      = "c-list-elem"
    var div1          = document.createElement("div")
    div1.className    = "c-id"
    div1.textContent  = `ID:      ${id}`
    var div2          = document.createElement("div")
    div2.className    = "c-name"
    div2.textContent  = `Channel: ${name}`
    var div3          = document.createElement("div")
    div3.className    = "c-topic"
    div3.textContent  = `Topic:   ${topic}`
    var inp           = document.createElement("input")
    inp.type          = "button"
    inp.className     = "c-join"
    inp.value         = "Join"
    inp.setAttribute("onclick", `joinChannel(${id})`)
    li.append(div1, div2, div3, inp)
    return li
}
```

```
<div id="CL-w"> <!-- List of all available Channels-->
    <input type="button" onclick="reset()" value="Home"/>
    <div id="CL-size"> <!-- Slider to change # of channels per page-->
        <input type="range" min="0" max="100" value="20" step="5" class="slider"
        id="CL-S-slider" onchange="setCLSize(this.value)" oninput="setSizeIndicator(this.value)">
        <span id="CL-S-value">20</span>
    </div>
    <div id="CL"> <!-- List of available channels--> </div>
    <div id="CL-PL"> <!-- List of pages for the channel list--> </div>
</div>
```

Channel Liste

- Die Channel Liste besteht aus vielen Elementen (eines für jeden Channel), welche an das Element im HTML mit der ID “CL” angehangen werden.
- Die CLItem() Funktion erstellt die Elemente und Join-Buttons des jeweiligen Channel, und fügt sie einem einzelnen wrapper hinzu.
- Mit .textContent wird sichergestellt, dass keine ungewollten HTML tags in die seite injiziert werden.
- Wir benutzen ein Element, anstatt von <ul> oder <ol>, da ein einfacher zu stylen ist.

Channels darstellen



Regelmäßig aktualisieren

- Sobald die Webseite fertig geladen hat, wird mit der nativen Funktion setInterval() sichergestellt, dass refreshCL() ungefähr alle 10 Sekunden aufgerufen wird.

```
c_interval = setInterval(refreshCL, 10000)
```

- Die Funktion setInterval() garantiert keinen 10 sekündigen Abstand, sondern nur, dass alle 10 Sekunden die Funktion neu aufgerufen wird. Es kann also passieren, dass zwei instanzen der funktion gleichzeitig laufen. Das ist in diesem fall aber nicht wichtig, da die Funktion Idempotent ist und von keinem globalen state abhängig ist der verändert werden kann.

- Der handle zu dem Intervall wird in einer globalen Variable gespeichert, falls man das updaten zu testzwecken stoppen möchte.

Channels bei- und austreten

```
function joinChannel(channelID) {
    getChannelInfo(channelID).then
    (res => setCInfo(res.id, res.name, res.topic))
    refreshUsers(channelID)
    activeChannel = channelID
    displayM(channelID)
}
```

```
<!-- Info about active channel-->
<div id="c-info-header">
    <span id="ci-id"></span>
    <span id="ci-name"></span>
    <span id="ci-topic"></span>
</div>
```

```
function setCInfo(cid, cname, ctopic) {
    $("#ci-id").text(`ID: ${cid}`)
    $("#ci-name").text(`Name: ${cname}`)
    $("#ci-topic").text(`Topic: ${ctopic}`)
}
```

ID: 480 Name: TestChannelName Topic: TestTopicName

- Durch das Aufrufen der joinChannel() Funktion mit einer channelID, kann man einem Channel beitreten. Dazu werden die Channel Informationen abgefragt und diese dann auf der Webseite angezeigt.
  Die Liste der aktiven User wird aktualisiert. Die globale Variable “activeChannel” wird verändert und die Nachrichten des beigetretenen Channels werden angezeigt.

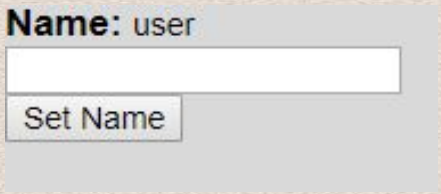
Channel Informationen (ID, Name und Topic)

- Die angezeigten Channel Informationen können jederzeit verändert werden, durch einen Aufruf der setCInfo() Funktion.
- Die HTML elemente befinden sich immer am oberen Rand der Webseite.
- Um Auch hier Injektionen zu verhindern, wird .text() verwendet.

Channels bei- und austreten

```
<div id="account-w"> <!-- User Info (Name, etc.) and change ability-->
    Name: <span id="a-name">user</span>
    <form onsubmit="this.reset(); return false">
        <input id="nname" type="text">
        <input type="submit" onclick="setUserName(nname.value)" value="Set Name">
    </form>
</div>
```

```
function setUserName(NName) {
    name = NName
    $("#a-name").text(NName)
}
```



Nutzername

- Beim Laden der Webseite wird dem Nutzer automatisch ein Default Nutzername “user” zugewiesen. Der Name wird in einer globalen Variable gespeichert und immer oben links auf der Webseite angezeigt. Man kann jederzeit seinen Namen ändern.
- Sobald man auf den Button “Set Name” klickt, wird die setUserName() Funktion mit dem Inhalt des Textfeldes aufgerufen. Die globale Variable wird verändert, ebenso wie der angezeigte Name auf der Webseite. Dieser Name wird automatisch verwendet wenn der Nutzer eine Nachricht sendet.
- Durch .text() werden hier ebenfalls Script-tags escaped.

Channels bei- und austreten

```
function renderUsers(pro) {
    pro.then(res => {
        var div_u  = document.createElement("div")
        for (var key in res) {
            div_u.append(UserItem(res[key]))
        }
        $("#c-member-list").html(div_u.innerHTML)
    })
}
```

User: D2

User: user

User: Hello1

```
function refreshUsers(channelID=activeChannel) {
    renderUsers(getUsers(channelID))
}
```

```
function getUsers(channelID) {
    // GET most recent list of active users of a channel
    let opts = {
        headers:new Headers(auth)
    }
    return fetch(url + `/channels/${channelID}/users`, opts)
    .then(function(res) {
        switch(res.status) {
            case 200:
                return res.json();
                break;
            default:
                throw new Error(`HTTP Status Error: ${res.status}`);
        }
    })
}
```

Nutzer Liste

- Die Funktion refreshUsers() kann mit einer channelID aufgerufen werden, um die aktiven Nutzer eines Channels anzeigen zu lassen.
- Hierzu wird zuerst getUsers() (eine API-Wrapper Funktion) aufgerufen, um die Informationen vom Back-End zu bekommen und dann mit der Antwort, die Funktion renderUsers() aufgerufen.
- renderUsers() benutzt das gleiche Pattern wie die Channel Liste und die Channel-Page-Liste um den Inhalt der Back-End Antwort auf die Webseite zu bringen.
- Nachrichten des aktiven Channels werden angezeigt, sobald ein Channel betreten wird.

Channel-Nachrichten anzeigen

Beispielhafter Ablauf einer Aktualisierung des Nachrichtenverlaufs :

- Die Funktion refreshM() wird aufgerufen. Die Liste der aktiven Nutzer wird refreshed.
- Starte ein Web-Request für die neuesten Nachrichten des aktiven Channels
- Falls schon einmal Nachrichten aus diesem Channel empfangen wurden, benutze den jüngsten verfügbaren Timestamp.
- Rufe mit der Antwort des Web-Requests die Funktion renderM() auf und warte mit der Ausführung von refreshM() bis diese Funktion fertig ist
- renderM() fügt allen Nachrichten ein natives Date-Objekt hinzu, um die Nachrichten der Reihenfolge nach zu sortieren
- Falls nicht bereits ein anderer Aufruf dieser Funktion an diesem Punkt stattfindet, rufe die Funktion updateM() mit dem Array der Nachrichten und der ID des Channels, aus dem die Nachrichten kommen, auf.
- Beim Laden der Seite wird ein neues, leeres, globales Objekt erstellt “ALL_M”. In diesem Objekt werden alle schon einmal gesehenen Nachrichten nach Channels sortiert gespeichert, zusätzlich zu dem Timestamp und ID, der jüngsten Nachricht.
- Die Funktion updateM() überprüft erst ob schon Nachrichten aus diesem Channel gespeichert wurden. Falls ja, werden alle bereits bekannten Nachrichten aus dem übergebenen Array herausgefiltert. Falls nein, wird der Filter-Schritt übersprungen.
- Die neuen Nachrichten werden sortiert, zu HTML-Elementen umgewandelt (falls bereits Nachrichten existieren, werden hiernach die neuen Nachrichten an die alten angehangen) und in einem neuen Objekt gespeichert.
- In dem gleichen Objekt wird zusätzlich noch die ID und der Timestamp der jüngsten Nachricht (für den leichteren Zugriff) gespeichert.
- Das neu erstellte Objekt wird dann unter der Channel-ID zu ALL_M hinzugefügt.
- An diesem Punkt läuft das Programm in refreshM() weiter. Hier wird die Funktion displayM() aufgerufen, welche, falls verfügbar, das erstellte Objekt aus ALL_M ab ruft und die Nachrichten mit demselben Pattern, wie oben, auf die Webseite bringt.

Channel-Nachrichten anzeigen

```
async function refreshM(channelID=activeChannel) {
    if (channelID > 0) {
        try {
            var ts = null
            if (channelID in ALL_M) {
                ts = ALL_M[channelID]['ts']
            }
            refreshUsers(channelID)
            await renderM(getMessages(channelID, ts))
        } catch(error) {
            console.error(error)
        }
        displayM()
    } else {
        // not in a channel
        $("#m-list-w").html("")
        $("#c-info-header").html('<span id="ci-id"></span>\
                                  <span id="ci-name"></span>\
                                  <span id="ci-topic"></span>')
        $("#c-member-list").html("")
    }
    clearTimeout(m_interval)
    if (m_timeout) {
        m_interval = setTimeout(refreshM, 1000)
    }
}
```

Regelmäßig aktualisieren (jede Sekunde)

- Die Funktion refreshM() wird jede Sekunde aufgerufen. Dazu benutzen wir die native setTimeout() Funktion. Zwei Sekunden, nachdem setTimeout() aufgerufen wurde, wird refreshM() aufgerufen. Der handle zu dem Timeout wird in der globalen Variable m_interval gespeichert.

- Direkt bevor ein neues Timeout gestartet wird, wird die native Funktion clearTimeout() auf den global verfügbaren handle aufgerufen. Somit wird sichergestellt, dass immer nur genau eine Version des Timeouts gleichzeitig läuft. Das heißt, keine zwei oder mehrere Schleifen, die sich alle immer wieder neu starten.

ID: 3382
Creator: user
Content: Test
Timestamp: Sat Jun 15 2019 20:56:24 GMT+0200 (Mitteleuropäische Sommerzeit)

Channel-Nachrichten anzeigen

```
function updateM(M, cid) {
    var A = {}
    if (cid in ALL_M) {
        let prev = ALL_M[cid]
        M = M.filter(m => m.id > prev['id'])
        if (M.length <= 0) {
            return
        } else {
            M = M.sort(sortM)
            let ts = M[M.length - 1].ts
            let id = M[M.length - 1].id
            A['M']   = prev['M'].concat(M.map(m => toHTMLItemM(m)))
            A['ts'] = ts
            A['id'] = id
            ALL_M[cid] = A
        }
    } else {
        M = M.sort(sortM)
        let ts = M[M.length - 1].ts
        let id = M[M.length - 1].id
        A['M']   = M.map(m => toHTMLItemM(m))
        A['ts'] = ts
        A['id'] = id
        ALL_M[cid] = A
    }
}
```

Nachrichten Reihenfolge (zeitliche Korrektheit)

- In der Funktion updateM() werden neue Nachrichten mit der Vergleichsfunktion sortM() sortiert, bevor sie evtl. an eine bereits sortierte Liste von Nachrichten angehängt werden.
- Die Funktion sortM() sortiert zuerst aufgrund des Timestamps, indem “Date”-Objekte verglichen werden, welche vorher aus den mitgelieferten Timestamps erstellt wurden.
- Falls der Timestamp gleich ist, wird die ID benutzt, um festzustellen, in welcher Reihenfolge die Nachrichten angezeigt werden sollen.

```
function sortM(a, b) {
    if          (a.d  > b.d) {return 1}
    else if     (a.d  < b.d) {return -1}
    else {return a.id > b.id ? 1 : -1}
}
```

Channel-Nachrichten anzeigen

```
function renderM(pro) {
    return pro.then(function(res) {
        if (res.page.totalElements <= 0) {
            // Channel doesn't have any messages
        } else {
            let M = res._embedded.messageList
            let cid = M[0].channel.id
            M = M.map(function(m) {
                return {
                    "id" : m.id,
                    "d"  : new Date(m.timestamp),
                    "ts" : m.timestamp,
                    "creator" : m.creator,
                    "content" : m.content
                }
            })
            if (lockM) {
                lockM = false
                try {
                    updateM(M, cid)
                } catch(error) {
                    console.error(error)
                }
                lockM = true
            }}}}
```

Zeitpunkt der Veröffentlichung, Author und Nachrichteninhalt

- Die Funktion renderM() verändert die Back-End Antwort, indem zusätzlich zum normalen Inhalt, noch ein Date-Objekt hinzugefügt wird. Unter anderem zum späteren Sortieren.
- Diese neuen Message-Elemente werden dann in der Funktion updateM() zu HTML-Elementen umgewandelt.
- Die Funktion toHTMLItemM() nimmt die relevanten Informationen über eine Nachricht, wandelt das Date-Objekt zu einem String um (durch die native Funktion .toString()) und gibt ein HTML-MessageItem zurück, in welchem jetzt alle wichtigen Informationen als Text gespeichert sind.
- Die HTML-Items werden in dem globalen Objekt “ALL_M” gespeichert und später von der Funktion displayM() auf die Webseite gebracht.

```
function toHTMLItemM(m) {return MessageItem(m.id, m.creator, m.content, m.d.toString())}
```

Channel-Nachrichten anzeigen

```
function MessageItem(id, creator, content, timestamp) {
    var div  = document.createElement("div")
    div.className = "m-list-elem"
    var div1 = document.createElement("div")
    div1.className = "m-id"
    div1.textContent = `ID: ${id}`
    var div2 = document.createElement("div")
    div2.className = "m-creator"
    div2.textContent = `Creator: ${creator}`
    var div3 = document.createElement("div")
    div3.className = "m-content"
    div3.textContent = `Content: ${content}`
    var div4 = document.createElement("div")
    div4.className = "m-timestamp"
    div4.textContent = `Timestamp: ${timestamp}`
    div.append(div1, div2, div3, div4)
    return div

}
```

Nachrichten nicht doppelt anzeigen

- In der Funktion updateM() werden am Anfang alle Nachrichten herausgefiltert, dessen ID niedriger ist, als die höchste ID, die in diesem Channel zuvor gesehen wurde.
- Da ID’s immer größer werden, kann, sobald eine ID x in einem Channel benutzt wurde, danach keine Nachricht mit der ID y gesendet werden, wo y < x.

```
function displayM(cid=activeChannel) {
    if (cid in ALL_M) {
        M = ALL_M[cid]['M']
        var div_m = document.createElement("div")
        M.forEach(function(m) {
            div_m.append(m)
        })
        $("#m-list-w").html(div_m.innerHTML)
    } else {
        $("#m-list-w").html("")
    }
}
```

Channel-Nachrichten schicken

```
async function sendMessageA(msg, channelID=activeChannel) {
    if (channelID <= 0) {return}
    var ts = null
    if (channelID in ALL_M) {ts = ALL_M[channelID]['ts']}
    await renderM(sendMessage(msg, channelID, ts))
    displayM()
}
```

```
<div id="m-form-w"> <!-- Form for Sending new Messages to active channel-->
    <form id="Mform" onsubmit="this.reset(); return false">
        <input  placeholder="Please enter text here"
         title="Please enter text here" id="mNewMsg" name="Message"
         type="text" size="40" maxlength="255"/>
        <button type="submit" id="b-msg"
        onclick="sendMessageA(mNewMsg.value, activeChannel)">Send</button>
    </form>
</div>
```

Nachrichten verfassen und in beigetretenen Channel veröffentlichen

- Wenn der Nutzer auf den “Send”-Button klickt und einem Channel beigetreten ist, dann wird die sendMessageA() Funktion mit dem Wert des Textfeldes als “msg”, und der globalen Variable activeChannel als channelID aufgerufen.
- Die Nachricht wird an die Funktion sendMessage() (eine API-Wrapper Funktion) weitergegeben, welche ein POST-Request an das Back-End schickt.
- Falls aus dem Channel schon einmal Nachrichten empfangen wurden, wird auch der Timestamp der jüngsten Nachricht beim POST mitgegeben.
- Die Nachricht wird in dem aktiven Channel veröffentlicht.

Channel-Nachrichten schicken

```
function sendMessage(msg, channelID=activeChannel, timestamp=null) {
    // POST a new message to a channel
    if (channelID === null || channelID <= 0) {
        throw new Error(`Not a valid Channel ID, was: ${channelID}`)
    }
    let opts = {
        method : 'POST',
        headers : new Headers(Object.assign({}, auth, {'Content-Type' : 'application/json'})),
        body : JSON.stringify({'creator' : name, 'content' : msg})
    }
    args = ""
    if (timestamp !== null) {args += `?lastSeenTimestamp=${timestamp}`}
    return fetch(url + `/channels/${channelID}/messages${args}`, opts)
    .then(function(res) {
        switch(res.status) {
            case 200:
                return res.json();
                break;
            case 404:
                throw new Error(`Channel ID doesn't exist: ${res.status}`);
                break;
            default:
                throw new Error(`HTTP Status Error: ${res.status}`);
        }
    })
}
```

Nachrichtentext und Autor

- Der Nachrichtentext wird wie oben beschrieben aus dem Textfeld genommen.
- Der String wird zusammen mit dem Autor im Request-Body mitgeschickt.
- Der Autor ist der, in der globalen Variable “name” gespeicherte Nutzername (standardmäßig “user”) und wird in der Funktion sendMessage() automatisch dem Request-Body hinzugefügt.

Channel-Nachrichten schicken

Direkt nach dem Senden an Nachrichtenverlauf anhängen

- Damit der Nutzer nicht auf das nächste Timeout von refreshM() warten muss, um seine Nachricht zu sehen, wird mit der Antwort von sendMessage() sofort die Funktion renderM() aufgerufen.

- Die Funktion renderM() ist, wie oben beschrieben, dafür zuständig, die Nachrichten im richtigen Format an updateM() zu geben, welche sie dann, an die bereits existierenden Nachrichten, in der richtigen Reihenfolge und ohne Dopplungen anhängt.

- Um hier eine Race-Condition zu verhindern, hat renderM() um den Aufruf von updateM() ein einfaches “lock”, welches verhindert, dass mehrere Aufrufe von updateM() gleichzeitig passieren. In renderM() ist das nicht nötig, da hier kein globaler state verändert oder gelesen wird. Dies gilt jedoch nicht für die Funktion updateM(), deshalb ist hier Vorsicht nötig.

- Sobald renderM() returned, wird displayM() aufgerufen, dieser sorgt dafür, dass die neu hinzugefügten Nachrichten auf der Webseite angezeigt werden.

Beschreibung der Gruppenarbeit

Wir haben gemeinsam, als Gruppe, an der ganzen Programmieraufgabe umfangreich gearbeitet. Die Funktionalitäten im JavaScript-Code haben wir untereinander aufgeteilt und jeder hat seinen Teil eigenständig erledigt. Zusätzlich zum JavaScript-Teil haben Florian und Erdem gemeinsam den HTML und CSS-Code geschrieben, während Sandra und Danial das ganze Programm ausgiebig auf Funktionalitäten und Anforderungen getestet haben. Gegenseitige Verbesserungsvorschläge, für eine bessere Veranschaulichung, des Codes wurden zur Kenntnis genommen und umgesetzt. Bei Fragen, die bei der Selbstbearbeitung aufgekommen sind, haben wir einander geholfen und sie zusammen geklärt.