

**Gymnasium Norf**

**Schuljahr 2020/2021**

**Webtechnologien: Echtzeitkommunikation  
mit Websockets – implementiert am Beispiel  
einer Zeitplan-Webapplikation für  
Konferenzen**

**Verfasser: Leon Kappes**

**Fach: Informatik**

**Fachlehrer/in: Herr Daniel Kreilmann**

**Kursnummer: IF20**

## Inhaltsverzeichnis

1. Einleitung.....	2
2. Echtzeitkommunikation.....	2
2.1. Definition.....	2
2.2. Client-Server-Model.....	3
2.2.1 Definition.....	3
2.2.2 Wieso Client-Server statt Peer-to-Peer.....	3
2.2.3 Umsetzung im Raum von Webtechnologien.....	4
3. Websockets.....	5
3.1 Erläuterung.....	5
3.2 Ablauf einer Websocket-Verbindung im Implementationskontext.....	5
3.3 Realisierung innerhalb der Implementation.....	6
3.4 Erzielung der Kriterien für eine Echtzeitkommunikation.....	7
4. Implementation.....	8
4.1 Aufbau der Implementierung.....	8
4.4 Gründe für die Umsetzung mit Webtechnologien.....	9
4.5 Ausblicke.....	10
5. Fazit.....	11
6. Literaturverzeichnis.....	12
7. Anhang.....	14
I. Abbildungen.....	14
II. Eigenständigkeitserklärung:.....	17
III. Protokollbogen.....	18
IV. Programmcode.....	19

## 1. Einleitung

Der immer schneller werdende Alltag erfordert immer schnellere Kommunikation. Ein Geschäftsmann in New York muss mit seinem Kollegen in Frankfurt ohne spürbare Latenz kommunizieren können, um geschäftsfähig und wettbewerbs-relevant zu bleiben. Doch selbst in unserem Alltag erwarten wir mit unseren Großeltern, distanzunabhängig, jederzeit in Kontakt stehen zu können.

Unter den Schwerpunkten, des sehr schlecht ausgebauten Breitbandinternets in Deutschland (siehe Abbildung 1) und dem Verlangen immer und überall erreichbar zu sein, ob am privatem Rechner oder unterwegs am Smartphone, bieten sich Webtechnologien als ideale, kostengünstige und einfache Lösung für die zuvor genannten Kriterien an.

Diese Ausarbeitung befasst sich nun also mit der Fragestellung wie und ob sich Echtzeitkommunikation mit Webtechnologien umsetzen lässt. Der Prozess der Lösungsfindung ist an einer Beispielimplementation erläutert und weitere Rahmenbedingungen erläutern sich in den folgenden Abschnitten. Da es sich bei diesen Technologien noch um, in Betrachtung, dass das Web an sich noch als relativ jung gilt, neue Technik handelt, ist die Verfügbarkeit von Literatur noch recht beschränkt und daher werde ich viel auf Fachzeitschriften zurück greifen, welche solche Themen schneller aufgreifen.

## 2. Echtzeitkommunikation

### 2.1. Definition

*Von Real-Time Communications (RTC) oder Echtzeitkommunikation wird bei jedem Modus gesprochen, bei dem Anwender Informationen sofort oder mit zu vernachlässigiger Latenz austauschen können.<sup>1</sup>*

Heißt, um in Echtzeit zu kommunizieren, muss der Informationsaustausch für den Anwender in einer unmerkbaren Geschwindigkeit, also sehr schnell, stattfinden. Im Implementationsbeispiel müssen die Eingaben eines Anwenders nahezu sofort bei den restlichen Anwendern eintreffen oder sie

---

<sup>1</sup> Vgl. <https://whatis.techtarget.com/de/definition/RTC-Real-Time-Communications-Echtzeitkommunikation> zuletzt abgerufen am 07.01.2021

verlieren ihre Relevanz (über ein verschobenes Meeting sollte man vor dem eigentlichen Termin Bescheid wissen).

Dies wird in der Beispielimplementation durch die Kommunikation im Client-Server-Modell erreicht.

## **2.2. Client-Server-Model**

### **2.2.1 Definition**

*Das Client-Server-Modell ist ein Prinzip, um die Verbindung und Aufgaben zwischen einem Client und einem Server zu organisieren und eine weit verbreitete Methode für viele Computeranwendungen.<sup>2</sup>*

Kurzum, ein zentraler Server handelt als Vermittler zwischen allen Anwendern (Clients). Im Implementationsbeispiel wäre die Architektur dann wie folgt umgesetzt:

Anwender A startet einen Konferenzpunkt 10 Minuten früher als geplant und sendet diese Änderung an den Server. Im Modell ist es dann des Servers Aufgabe alle anderen Clients (Anwender) über diese Änderung in Kenntnis zu setzen, beziehungsweise den neuen Datensatz (neuer Konferenzplan) an alle Clients zu übermitteln, sodass diese über eine, sie möglicherweise betreffende Änderung, Bescheid wissen. Eine bildliche Darstellung findet sich in Abbildung 2.

### **2.2.2 Wieso Client-Server statt Peer-to-Peer**

Peer-to-Peer stellt das genaue Gegenstück zum Client-Server-Modell<sup>3</sup> dar. Es definiert sich als *Zusammenschluss von gleichberechtigten Arbeitsstationen in Netzwerken, die den Einsatz von verteilten Anwendungen und den Austausch von Dateien ermöglichen. Ein zentraler Server ist hierfür nicht notwendig.<sup>4</sup>*

Diese sich ergebene Gleichberechtigung und die Möglichkeit eines jeden sich im Netzwerk befindlichen Rechners auf Funktionen/Ressourcen eines anderen zuzugreifen und so die Rollen des Clients und des Servers aus dem Client-

---

2 Vgl. <https://www.melaschuk-medien.de/begriffe-definition-publishing-crossmedia-marketing-it/client-server.html> zuletzt abgerufen am 08.01.2021

3 Vgl. <https://www.fonial.de/wissen/begriff/client-server-modell/> zuletzt abgerufen am 09.01.2021

4 Vgl. <https://wirtschaftslexikon.gabler.de/definition/peer-peer-p2p-42462/version-265810> zuletzt abgerufen am 09.01.2021

Server-Modell einzunehmen<sup>5</sup>, ist in dem gegebenen Sachkontext eine unzureichende Lösung. In einem Netzwerksystem in dem es verschiedene Klassen an Teilnehmern gibt (Leiter eines Termins, einfache Teilnehmer) ist ein Peer-to-Peer System unpraktikabel, auch wenn Echtzeitkommunikation mit Peer-to-Peer umsetzbar wäre und es auch in solchen Szenarien Verwendung findet (Zum Beispiel in der Echtzeitkommunikation von Spielen).<sup>6</sup> Es ist also nicht möglich, diese Anwendung, einfach und effizient mit Peer-to-Peer umzusetzen, da, ohne einen prüfenden Server, alle gleichberechtigt sind und man den Eingaben der anderen Teilnehmern vertrauen müsste.

### **2.2.3 Umsetzung im Raum von Webtechnologien**

Wie eingangs erwähnt, befasst sich diese Arbeit mit der Umsetzung von Echtzeitkommunikation mit Hilfe von Webtechnologien. Um die Kriterien für eine Datenübertragung mit einer verachtend geringen Latenz zu erreichen, bieten sich „normale“ HTTP-Anfragen nicht an, da diese ein aktives Nachfragen des Clients (auch „Polling“ genannt) benötigen, um neue Datensätze zu empfangen und eine bidirektionale Kommunikation nicht ohne „an abuse of HTTP to poll the server for updates while sending upstream notifications as distinct HTTP calls“(Fette&Melnikov 2011, The WebSocket Protocol rfc6455).<sup>7</sup> Eine bidirektionale Kommunikation(Kommunikation ist in beide Richtungen möglich) ist gefordert, da wir Aktualisierungen vom Server erwarten und der Client den Server über neue Ereignisse informieren muss. Es wird also eine Kommunikationsmöglichkeit gesucht, welche Server und Client erlaubt eine Verbindung aufzubauen und Datensätze in beide Richtungen auszutauschen, ohne, dass der Webbrower die ganze Zeit Anfragen stellen muss, ob es mittlerweile eine Änderung gibt. Eine geeignete Lösung, die auch den bidirektionalen Datentransfer ermöglicht, stellen die relativ neuen Websocket Verbindung dar. Weitere Erläuterungen und eine Definition von Websockets, sowie eine Erklärung ihrer Funktion und die Umsetzung in der Beispieldokumentation, finden sich in Abschnitt 3: Websocket dieser Ausarbeitung.

---

5 Vgl. [https://www.sachsen.schule/~gdb/daten\\_verarbeiten/nw/Netzwerktypen.html](https://www.sachsen.schule/~gdb/daten_verarbeiten/nw/Netzwerktypen.html) zuletzt abgerufen am 09.01.2021

6 Vgl. <https://docs.microsoft.com/de-de/dotnet/framework/network-programming/peer-to-peer-networking-scenarios> zuletzt abgerufen am 09.01.2021

7 Vgl. <https://tools.ietf.org/html/rfc6455> zuletzt abgerufen am 16.01.2021

### *Wie ist die Netzwerkstruktur im Implementationsbeispiel aufgebaut?*

Wenn ein neues Ereignis erstellt wird, geht der Client eine Verbindung mit dem Server ein und überträgt die Informationen des angelegten Ereignisses. Sollte nun ein weiterer Nutzer nun beitreten, wird von diesem eine Verbindung zum Server aufgebaut und der Client erhält vom Server alle relevanten Informationen. Dies geschieht mit jedem Client, der sich in Zukunft verbindenden wird. Sollte der Host-Client, der Client, der das Ereignis erstellt hat und nun intern als Host angesehen wird, eine Änderung am Ereignis vornehmen, wird diese durch die vorhandene Verbindung zum Server geleitet und dieser, welcher eine Liste aller Clients führt, die das Ereignis abonniert haben, überträgt dann die Änderungen an alle Clients. Man kann sich dieses Verhalten leicht an einem Online-Gruppen-Chat erläutern, bei welchem eine abgesendete Nachricht zu einem Server geschickt wird, welcher dann die Nachricht an alle anderen Teilnehmer der Gruppe überträgt.

## **3. Websockets**

### **3.1 Erläuterung**

*WebSocket ist ein bidirektionaler und (voll-)duplexer Kommunikationsstandard und besteht aus einer Client-API und einem Netzwerkprotokoll.<sup>8</sup>*

Heißt konkret, dass es dem Server und dem Client möglich ist, über eine einzelne Verbindung Daten in beide Richtungen zu übertragen. Solche Websocket Verbindungen basieren auf einer Client-API, welche in jedem modernen Browser verfügbar ist<sup>9</sup>. Der Aufbau, dieser API, wurde so entwickelt, dass sie den Vorgaben des WebSocket-Protokolls, welches durch die Internet Engineering Taskforce entwickelt wurde, entspricht.<sup>10</sup>

### **3.2 Ablauf einer Websocket-Verbindung im Implementationskontext**

Die Websocket-Verbindung wird vom Client erstellt. Dieser verbindet sich mit dem Endpunkt des Servers, indem er eine Anfrage für das Wechseln zum

---

<sup>8</sup> Vgl. <https://www.heise.de/developer/artikel/WebSocket-Annäherung-an-Echtzeit-im-Web-1260189.html?seite=2> zuletzt abgerufen am 17.01.2021

<sup>9</sup> Vgl. Abbildung 3

<sup>10</sup> Vgl. <https://www.heise.de/developer/artikel/WebSocket-Annäherung-an-Echtzeit-im-Web-1260189.html?seite=2> zuletzt abgerufen am 17.01.2021

Websocket-Protokoll stellt. Dies ist in der Benutzeroberflächen-Implementation durch die zuvor angesprochene Client-API des Browsers gelöst. In der Implementation wird ein neues Websocket-Objekt erzeugt und es wird die URL angegeben, zu welcher eine Verbindung hergestellt werden soll(vgl. Abbildung 4). Anschließend wird eine HTTP-Anfrage gestellt, um die Verbindung herzustellen. Diese Anfrage fordert den Server auf, eine Websocket-Verbindung mit dem Browser/Client einzugehen. Bei einer erfolgreichen Verbindung antwortet der Server, indem er angibt, dass das Protokoll gewechselt werden soll (HTTP-Code 101 Switching Protocols) und die Anfrage der Websocket-Verbindung akzeptiert wird. Die Verbindung ist nun aktiv.<sup>11</sup>

Es ist nun möglich mit der *send*-Methode des nun erstellten Websocket-Objekts Daten an den Server zu senden (vgl. Abbildung 5) und mit dem *message*-Event Nachrichten des Servers zu erhalten und zu verarbeiten (vgl. Abbildung 6).<sup>12</sup> In der Implementation wird die Nachricht nun in Kommando und Daten zerteilt. Diese Kommandos sind zuständig für die Manipulation des Erscheinungsbilds der Benutzeroberfläche, durch in etwa das Ändern der Konferenzreihenfolge, oder weitere interne Methodenaufrufe, wie beispielsweise das Senden von Benachrichtigungen.

Die vom Server gesendeten Datensätze bestehen aus zwei Teilen:

1. Dem Kommando, welches wie oben genannt zur internen Zuordnung dient
2. Einem, mit einem Leerzeichen separierten, Datensatz im JSON-Format. Dieser Datensatz ist eine Java-Objektinstanz, welche in einen JSON-String konvertiert wurde. Konkret heißt dies, dass es sich um ein Java-Objekt handelt, welches in eine Zeichenkette im JSON-Format umgewandelt wurde, sodass es von einem Programm in einer anderen Programmiersprache verwendbar ist.

*Exkurs: Was ist JSON beziehungsweise das JSON-Format*

*„Bei JSON handelt es sich um ein Textformat, das komplett unabhängig von Programmiersprachen ist, aber vielen Konventionen folgt, die Programmieren aus der Familie der C-basierten Sprachen (inklusive C, C++, C#, Java,*

---

<sup>11</sup> Vgl. <https://www.heise.de/developer/artikel/WebSocket-Annäherung-an-Echtzeit-im-Web-1260189.html?seite=2> zuletzt abgerufen am 17.01.2021

<sup>12</sup> Vgl. <https://html.spec.whatwg.org/multipage/web-sockets.html#dom-websocket> zuletzt abgerufen am 18.01.2021

*JavaScript, Perl, Python und vielen anderen) bekannt sind. Diese Eigenschaften machen JSON zum idealen Format für Datenaustausch.“<sup>13</sup>*

*JSON ist also einfach eine Art der Datenformatierung, die es einfach macht Daten zwischen verschiedenen Programmiersprachen auszutauschen.*

Die Benutzeroberfläche, genauer gesagt die Web-Applikation, hat nun eine bidirektionale Verbindung mit dem Backend-Server und kann Nachrichten nach dem eigens definierten Muster parsen und dementsprechend verarbeiten. Außerdem ist es möglich Nachrichten, nach demselben Muster, an den Server zu senden, um weiter verarbeitet und verteilt zu werden.

### ***3.3 Realisierung innerhalb der Implementation***

Nachdem die Verbindung zum Server steht und nun das Senden und Empfangen von Nachrichten möglich ist, gehe ich nun weiter auf den Aufbau des Datenaustausches ein. Die erste Anfrage, beziehungsweise die erste Nachricht, geht vom Client aus. Er meldet dem Server seinen Login-Versuch und gibt an, ob er ein einfacher Client oder ob er Konferenzleiter ist (vgl. Abbildung 7). Im User-Objekt wird nun serverseitig vermerkt, ob es sich um einen Konferenzleiter handelt und welcher Konferenz er angehört. Sollte nun bereits eine Konferenz hinterlegt sein, antwortet der Server mit der Anweisung „*updateConference*“ und einem Datensatz, welcher den kompletten Plan der Konferenz enthält (Verzug, Liste einzelner Konferenzpunkte mit Dauer, Name etc.) und einer einzigartigen Kennung (identifier), unter welcher der Server den Client adressiert.

Sollte sich der Client, beim Login, als Konferenzleiter identifiziert haben, akzeptiert der Server zwei weitere Befehle von ihm: „*setupConference*“ und „*controlConference*“. Der erste Befehl setzt, wie der Name schon sagt, die Konferenz auf und der Datensatz muss die komplette Konferenz, als Objekt repräsentiert, enthalten, und Letzterer ermöglicht das Wechseln des aktiven Konferenzthemas und sorgt für eine Neuberechnung des Verzugs sowie für eine Aktualisieren der Konferenz bei allen verbundenen Teilnehmern.

Sollte ein Client nun Benachrichtigungen für ein Thema der Konferenz erhalten wollen, so teilt er dies über den „*setNotification*“ Befehl dem Server

---

13 Vgl. <https://www.json.org/json-de.html> abgerufen am 31.01.2021

mit. Der Datensatz dieses Befehls enthält den Index des Elements, zu welchem er Benachrichtigungen erhalten möchte sowie alle erforderlichen Daten für die Push-API, welche genutzt wird, um Nachrichten zu versenden, ohne dass der Client die Applikation geöffnet haben muss.<sup>14</sup>

### **3.4 Erzielung der Kriterien für eine Echtzeitkommunikation**

Die Hauptanforderung dieser Applikation besteht darin, die Kriterien der Echtzeitkommunikation mit Webtechnologien zu erfüllen. Im Zuge dieser Erarbeitung hat sich das WebSocket-Protokoll als vielversprechende Lösung präsentiert und Websockets werden als Bestandteil von dem angesehen, was dem Web die Möglichkeit zur Echtzeitkommunikation und Datenverarbeitung gibt.<sup>15</sup>

Betrachten wir die Implementation nun also nach den in [Abschnitt 2.1](#) definierten Kriterien, also „[ein Datenaustausch] bei dem Anwender Informationen sofort oder mit zu vernachlässigiger Latenz austauschen können.“ so stellen wir fest, dass eine sofortige Kommunikation zwar realistisch nicht möglich ist, die Implementation jedoch beim Anwender das Gefühl vermittelt, dass sich Änderungen in Echtzeit wiederfinden. Dies bleibt der Funktion, dass Änderungen der Konferenz vom Server an alle Clients verteilt werden und es keine Interaktion, wie etwa eine manuelle Synchronisation, oder eine Ladezeit benötigt, geschuldet.

## **4. Implementation**

### **4.1 Aufbau der Implementierung**

Die Implementation ist in zwei Teile aufgebaut, einem Frontend, geschrieben in Javascript und dem Frontendframework Nuxt.js(Vue.js), und einem Backend, geschrieben in Java mit einer WebSocket-Bibliothek.

„Als Frontend wird die so genannte Präsentationsebene bezeichnet – also der Teil einer Software-Anwendung oder anderen Applikation wie beispielsweise einer Webseite, der für den Betrachter sichtbar ist (grafische

---

14 Vgl. <https://www.w3.org/TR/push-api/> zuletzt abgerufen am 21.01.2021

15 Vgl. <https://stackoverflow.blog/2019/12/18/websockets-for-fun-and-profit/> zuletzt abgerufen am 21.02.2021

Benutzeroberfläche).“<sup>16</sup> In dieser Implementation befindet sich das Frontend im „frontend“-Ordner und basiert auf dem Javascript-Framework Nuxt.js(<https://nuxtjs.org/>), welches selbst auf Vue.js(<https://vuejs.org/>) aufgebaut ist. Im Hauptordner des Frontends finden sich einige Dateien, welche zuständig für die Versions- und Paketverwaltung sind, sowie als Konfiguration für Nuxt.js dienen. Die Funktionen der einzelnen Ordner sind wie folgt: Der „assets“ Ordner beinhaltet unkomplizierte Dateien, wie Schriftarten und Bilder. Im „components“ Ordner finden sich wiederverwendbare Komponente der Applikation, in diesem Fall zum Beispiel sind es die einzelnen Elemente der Konferenzliste, welche in einer Schleife erzeugt werden.<sup>17</sup> Der „layouts“ Order beinhaltet eine „default.vue“, welche ein Seitenlayout definiert, welches um alle Seiten eingefügt wird, die aufgerufen werden. In diesem Fall ist es die Navigation.<sup>18</sup> Der „pages“ Ordner enthält die einzelnen Seiten der Applikation, heißt also einmal die Index-Seite, erreichbar sobald man die App öffnet, und die Einstellungsseite, bei welcher man auch eine Konferenz erstellt.<sup>19</sup> Der „plugin“ Ordner enthält Java-/Typescript Dateien, welche vor dem eigentlichen Ausführen der Applikation geladen werden. In diesem Fall wird die Websocket-Verbindung erstellt und dem Anwendungskontext angehängt.<sup>20</sup> Alle Dateien des „static“ Ordners werden direkt an den Webroot, also wenn man es als URL betrachtet hinter dem Schrägstrich(/), gebunden. Hier finden sich Dateien, wie das „favicon“ (Icon welches neben einem Tab im Browser angezeigt wird).<sup>21</sup> Der letzte Ordner, der „store“, beinhaltet die Typescript Datei, welche zuständig dafür ist den momentanen Stand der Applikation zu verwalten und überall Verfügbar zu machen. Der Stand kann über einige Setter verändert werden.<sup>22</sup>

Die Websocket-API der Webapplikation verbindet sich mit dem Websocket-Server des Backend-Service. Beim Backend handelt es sich um serverseitige Programmlogik und „*Im Allgemeinen wird der Begriff verwendet, um die Anwendung und Verwaltung einer bestimmten Software oder dem „internen“*

---

16 Vgl. <https://it-service.network/it-lexikon/frontend> zuletzt abgerufen am 27.01.2021

17 Vgl. <https://nuxtjs.org/docs/2.x/get-started/directory-structure> zuletzt abgerufen am 27.01.2021

18 Vgl. <https://nuxtjs.org/docs/2.x/directory-structure/layouts> zuletzt abgerufen am 27.01.2021

19 Vgl. <https://nuxtjs.org/docs/2.x/directory-structure/pages> zuletzt abgerufen am 27.01.2021

20 Vgl. <https://nuxtjs.org/docs/2.x/directory-structure/plugins> zuletzt abgerufen am 27.01.2021

21 Vgl. <https://nuxtjs.org/docs/2.x/directory-structure/static> zuletzt abgerufen am 27.01.2021

22 Vgl. <https://nuxtjs.org/docs/2.x/directory-structure/store> zuletzt abgerufen am 27.01.2021

*Bereich einer Seite zu definieren.“<sup>23</sup>* Im Zuge der Implementation werden also die Verbindungen der Nutzer verwaltet und eine kommunikative Brücke zur Verteilung von aktualisierten Datensätzen gestellt. Die „*WebSocketManager*“-Klasse dient als Einstiegspunkt der Applikation und verwaltet die einzelnen User-Objekte und hält weitere Instanzen diverse Klassen, welche Zuständigkeiten wie beispielsweise das Parsen von Befehlen übernehmen. Im „*entity*“-Paket finden sich verschiedenste Klassen, welche einzelne Komponente wie einen User oder eine Konferenz beschreiben zum Großteil zu Serialisierung über das Websocket genutzt werden. Im „*.net*“-Paket findet sich die Implementierung der eigentlichen Websocket-Server Logik, Verbindungen akzeptiert und eingehende Nachrichten anhand ihres Absenders identifiziert und dem Kommando-Parser im „*command*“-Paket übergibt, welcher die Nachrichten nach dem bereits erwähnten Schema zerlegt und das zuständige Kommando aus dem „*commands*“-Paket aufruft und ausführt. Das Backend nutzt 2 Bibliotheken, einmal die „*org.json*“-Bibliothek für das Verarbeiten von JSON-Datensätzen, welche wir über das Websocket erhalten und versenden, sowie für das Serialisieren von Java-Objektinstanzen in ein JSON-Format, welches von einem Javascript Programm einfach lesbar und verarbeitbar ist. Zudem wird die „*org.java\_websocket*“-Bibliothek genutzt um eine Websocket Schnittstelle in einem Java-Programm bereitzustellen.

#### **4.4 Gründe für die Umsetzung mit Webtechnologien**

Wie eingangs erwähnt haben Websockets die Möglichkeit für Echtzeitkommunikation in Webapplikation ermöglicht und mit der Einführung von „Progressiv Web Apps“ in fast allen populären Webbrowsern ist die Erreichbarkeit einer solchen App stark gestiegen. Webapps bieten die Vorteile, dass sie plattformunabhängig sind und eine günstigere und einfachere Entwicklung mit sich ziehen, da man nicht für jede Plattform eine native App entwickeln, warten und betreuen muss. Die dafür nötigen Technologien sind zwar noch sehr neu, haben sich jedoch schon bei einigen Unternehmen als gute Alternative oder Ergänzung zu nativen Apps entwickelt<sup>24</sup>. Zudem sollen solche Webapplikationen eine Nutzererfahrung mit sich bringen, welche vergleichbar mit einer App aus einem App-Store ist, gleichzeitig aber durch die

---

23 Vgl. <https://www.seo-kueche.de/lexikon/backend/> abgerufen am 28.01.2021

24 Vgl. <https://web.dev/what-are-pwas/> abgerufen am 27.01.2021

Verfügbarkeit im Web offener und schneller ist. Dieses Ziel der Schnelligkeit und die kleinere Dateigröße einer solchen App ist einer der Hauptgründe wieso ich diese Technologie bei dieser Implementation gewählt, neben dem weiterem Grund der Plattformunabhängigkeit, sodass keine 2 Apps erforderlich sind. Twitter war es zum Beispiel möglich durch ihre Webapp 40% weniger an Mobilen Daten zu verbrauchen und eine App auszuliefern die nur 1-3% der Größe ihrer nativen Apps auf dem Gerät benötigt.<sup>25</sup> AliExpress zeigt zudem ein weiteres Argument für Webapps, nämlich, dass es sich gezeigt hat, dass Nutzer mehr Zeit in Webapps verbringen und es sich mehr bleibende Nutzer entwickelt haben. Sie hatten zudem das Problem, dass sehr wenig Nutzer ihrer Mobile App installiert haben und dort nur sehr wenig Zeit verbracht haben, dieses Problem hat eine leicht installierbare Webapp gelöst<sup>26</sup>.

#### **4.5 Ausblicke**

Zum Zeitpunkt der Implementierung ist es noch nicht möglich Push-Benachrichtigungen Serverseitig, bei einer Webapp, auszulösen. Die meisten großen Webbrower unterstützen dieses Feature bereits, außer Safari auf IOS und da Browser wie Chrome auf einem IOS Gerät auch auf Webkit, Apples mobile Implementierung, beschränkt sind ist es auf einem IOS Gerät schlicht unmöglich dies anhand der geltenden technischen Bedingungen zu implementieren. Es bleibt also abzuwarten wie lange es noch braucht, bis diese Funktion in Webkit implementiert wird, es wäre aber jedoch ein guter Punkt zur Weiterentwicklung der App, da dies den Funktionskomfort verbessern würde und der Grundstein für diese Funktion bereits steht (das Backend enthält bereits Programmcode für Benachrichtigungen und der Datenverkehr ist weiter oben in dieser Arbeit erklärt)<sup>27</sup>. Weitere Möglichkeiten der Weiterentwicklung beständen zudem in einer Erweiterung der Applikation um ein Menü, indem man eine Konferenz mit Nutzereingaben erstellen kann und nicht darauf angewiesen ist eine CSV-Tabelle hochzuladen. Auch wäre denkbar eine Möglichkeit zu schaffen Konferenzen bearbeiten zu können, während sie aktiv sind. Dies würde zum Beispiel ermöglichen ein spontan entfallendes

---

25 Vgl. [https://blog.twitter.com/engineering/en\\_us/topics/open-source/2017/how-we-built-twitter-lite.html](https://blog.twitter.com/engineering/en_us/topics/open-source/2017/how-we-built-twitter-lite.html) abgerufen am 28.01.2021

26 Vgl. <https://developers.google.com/web/showcase/2016/aliexpress> abgerufen am 28.01.2021

27 Vgl. <https://www.izooto.com/blog/ios-safari-push-notifications-in-2021> abgerufen am 28.01.2021

Konferenzthema zu entfernen, ohne eine neue Konferenz generieren zu müssen und dementsprechend die CSV-Tabelle abzuändern. Es bleiben also noch einige Möglichkeiten zu Erweiterung der Applikation, vor allem in Betrachtung von Technologischen Weiterentwicklungen im Raum von Webtechnologien.

## 5. Fazit

Abschließend lässt sich also sagen, dass Echtzeitkommunikation mit Webtechnologien eine gute Alternative zu aufwendigen Netzwerkimplementierungen mit eigens entwickelten Desktopprogrammen ist. Websockets eröffneten mit ihrer Implementierung in allen populären Internet-Browsern, auf dem Desktop oder Mobil, enorme Möglichkeiten von Plattform unabhängige Software auszuliefern und das Internet beziehungsweise das Web als Plattform für einfache Echtzeitkommunikation nutzbar zu machen. Mit weiterer Adaption von Webstandarts der „Progressive Web Apps“ bieten sich immer mehr Möglichkeiten statt einer nativen Applikation, einfach hochkompatible Webapplikation zu entwickeln und das Gefühl einer echten nativen App<sup>28</sup> zu erzeugen oder zu emulieren. Echtnachrichtendienste könnten auf eine reine Webanwendung beschränkt werden, was die Entwicklungskosten massiv reduziert und Budget für Innovation lässt. Allein mit der momentanen Verfügbarkeit und Unterstützung bewähren sich Webapps als sehr effektiv<sup>29</sup> und das wäre ohne die Möglichkeit der Echtzeitdatenverarbeitung durch Websockets nicht möglich.

---

28 Eine native App wurde nach den Vorgaben des Plattformanbieters entwickelt und läuft ohne weitere Hilfe auf Endgeräten dieser Plattform. Möchte man seine App auf, zum Beispiel Android-Geräten anbieten, so wäre eine native App in Java geschrieben und im Google Play-Store erhältlich.

29 Vgl. <https://web.dev/what-are-pwas/> abgerufen am 27.01.2021

## 6. Literaturverzeichnis

### Internetquellen:

Rouse, Margaret: RTC – Real-Time Communications: Echtzeitkommunikation, <https://whatis.techtarget.com/de/definition/RTC-Real-Time-Communications-Echtzeitkommunikation> (Zugriff am 07.01.2021)

Client-Server, <https://www.melaschuk-medien.de/begriffe-definition-publishing-crossmedia-marketing-it/client-server.html> (Zugriff am 08.01.2021)

Client-Server-Modell, <https://www.fonial.de/wissen/begriff/client-server-modell/>, (Zugriff am 09.01.2021)

Lackes, Richard; Siepermann, Markus; Kollmann, Tobias(19.02.2018):Peer-to-Peer (P2P), <https://wirtschaftslexikon.gabler.de/definition/peer-peer-p2p-42462/version-265810> (Zugriff am 09.01.2021)

Zwei Aufbauprinzipien von Computernetzwerken, [https://www.sachsen.schule/~gdb/daten\\_verarbeiten/nw/Netzwerktypen.html](https://www.sachsen.schule/~gdb/daten_verarbeiten/nw/Netzwerktypen.html) (Zugriff am 09.01.2021)

Peer-to-Peer-Netzwerkszenarien,  
<https://docs.microsoft.com/de-de/dotnet/framework/network-programming/peer-to-peer-networking-scenarios> (Zugriff am 09.01.2021)

Infografik WELT (21.07.2019): So schnell ist das Internet in Ihrem Bundesland,

<https://www.welt.de/wirtschaft/webwelt/article197193483/Internetgeschwindigkeit-Diese-Bundeslaender-sind-die-schnellsten.html> (Zugriff am 16.01.2021)

Fette & Melnikov(Dezember 2011): The WebSocket Protocol, <https://tools.ietf.org/html/rfc6455> (Zugriff am 16.01.2021)

Weßendorf, Matthias(15.06.2011): WebSocket: Annäherung an Echtzeit im Web, <https://www.heise.de/developer/artikel/WebSocket-Annaeherung-an-Echtzeit-im-Web-1260189.html> (Zugriff am 17.01.2021)

Can I use Websockets?, <https://caniuse.com/websockets> (Zugriff am 18.01.2021)

HTML Living Standard: Web sockets, <https://html.spec.whatwg.org/multipage/web-sockets.html> (Zugriff am 18.01.2021)

Einführung in JSON, <https://www.json.org/json-de.html> (Zugriff am 31.01.2021)

Push API, <https://www.w3.org/TR/push-api/> (Zugriff am 21.01.2021)

Pekarsky, Max(18.12.2019): WebSockets for fun and profit, <https://stackoverflow.blog/2019/12/18/websockets-for-fun-and-profit/> (Zugriff am 21.01.2021)

Frontend – Definition, <https://it-service.network/it-lexikon/frontend> (Zugriff am 27.01.2021)

Nuxt.js Dokumentation, <https://nuxtjs.org/docs/2.x/get-started/installation> (Zugriff am 27.01.2021)

Was ist das Backend?, <https://www.seo-kueche.de/lexikon/backend/> (Zugriff am 28.01.2021)

Gallagher, Nicolas(06.04.2017):How we built Twitter Lite, [https://blog.twitter.com/engineering/en\\_us/topics/open-source/2017/how-we-built-twitter-lite.html](https://blog.twitter.com/engineering/en_us/topics/open-source/2017/how-we-built-twitter-lite.html) (Zugriff am 28.01.2021)

AliExpress, <https://developers.google.com/web/showcase/2016/aliexpress> (Zugriff am 28.01.2021)

Pravin, Pravya: iOS Safari Push Notifications 2021: Are We There Yet?, <https://www.izooto.com/blog/ios-safari-push-notifications-in-2021> (Zugriff am 28.01.2021)

Richard, Sam; LePage, Pete(6. Januar 2020, aktualisiert am 24 Februar 2020):What are Progressive Web Apps?, <https://web.dev/what-are-pwas/> (Zugriff am 27.01.2021)

## 7. Anhang

I. Abbildungen

II. Eigenständigkeitserklärung

III. Protokollbogen

IV. Programmcode

### I. Abbildungen

#### Downloadgeschwindigkeiten 2019

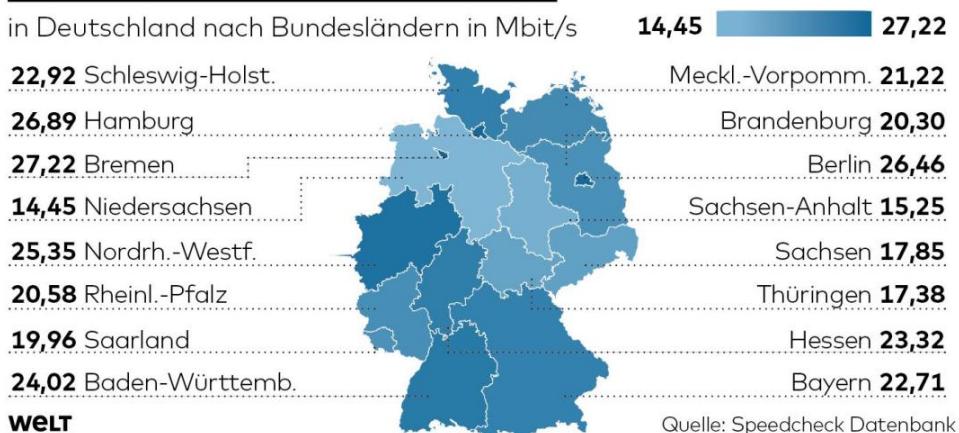


Abbildung 1: Download-Geschwindigkeiten in Deutschland (Welt, 2019)

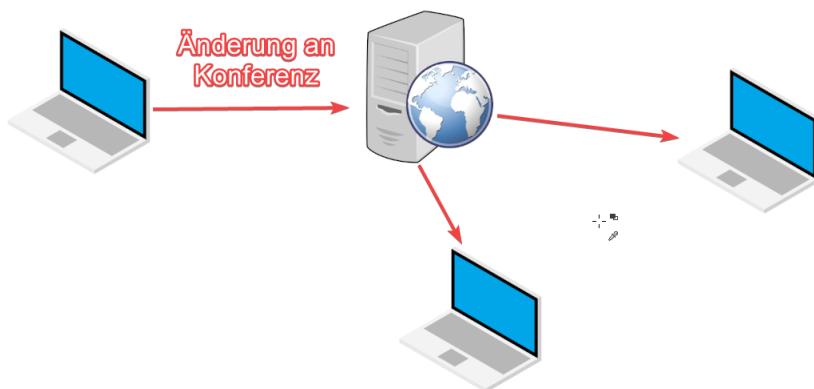


Abbildung 2: Datenaktualisierung im Client-Server-Modell

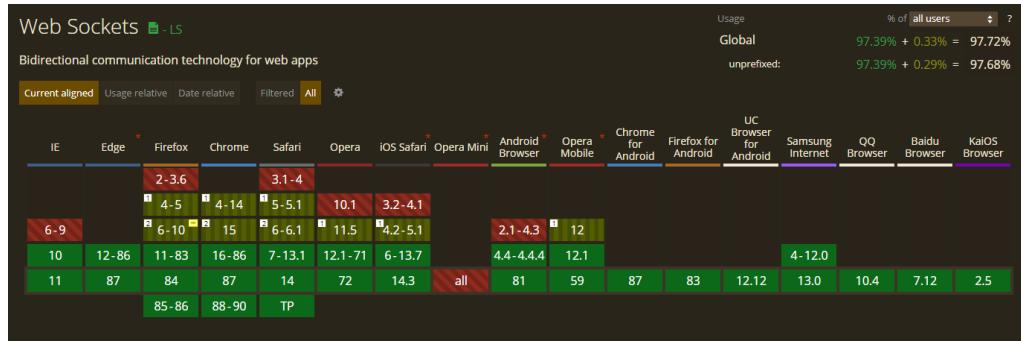


Abbildung 3: Browser-Kompatibilität von Websockets (caniuse.com)



Abbildung 4: Erzeugen eines Websocket-Objekts in der Implementation



Abbildung 5: Senden von Daten an den Server innerhalb der Implementation



Abbildung 6: Erwarten von Nachrichten des Servers anhand der Websocket-Client-API

```
{  
  "privileged": false, // Rolle des Clienten, Konferenzleiter oder nur Teilnehmer  
  "code": "abc123"     // Konferenz zu welcher am beitritt  
}
```

Abbildung 7: Beispiel eines Login-Pakets, welches vom Client zum Server gesendet wird

## II. Eigenständigkeitserklärung:

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Hilfsmittel verwendet habe.

Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht habe.

Neuss, den 06.02.2021 Franz Keppler  
(Ort) (Datum) (Unterschrift)

### III. Protokollbogen

#### II. Protokollbogen

Thema der Facharbeit

Webtechnologien: Echtzeitkonversationen mit Websockets

implementiert am Beispiel einer zeitplan - Webapplikation für Konferenzen

Datum, Unterschrift des Fachlehrers/ der Fachlehrerin

Erstes Gespräch

Datum: 29.10.2020

Thema des Gesprächs: Bewertungskriterien und mögliche Inhalte

Vereinbarungen:

Datum, Unterschrift des Fachlehrers/ der Fachlehrerin 29.10.20 Plu

Zweites Gespräch

Datum: 27.11.2020

Thema des Gesprächs: Themenfindung und Fragestellung

Vereinbarungen:

Formulierung eines Themas und Strukturierung einer Fragestellung

Datum, Unterschrift des Fachlehrers/ der Fachlehrerin 27.11.20 Plu

Drittes Gespräch

Datum:

Thema des Gesprächs:

Vereinbarungen:

Datum, Unterschrift des Fachlehrers/ der Fachlehrerin

Hier bestätige ich, dass ich über die fachspezifischen Arbeitsweisen, insbesondere die Zitierweise, informiert wurde.

Natascha, 29.01.2021, Frau Koppa

Ort, Datum, Unterschrift Schüler/in

\_\_\_\_\_  
Unterschrift Lehrer/in

## IV. Programmcode

```
1 package space.kappes;
2
3 import space.kappes.command.CommandManager;
4 import space.kappes.commands.ControlConference;
5 import space.kappes.commands.Login;
6 import space.kappes.commands.SetupConference;
7 import space.kappes.entity.Conference;
8 import space.kappes.entity.ConferenceItem;
9 import space.kappes.entity.User;
10 import space.kappes.net.IWebsocketServer;
11
12 import java.util.HashMap;
13 import java.util.Map;
14
15 /**
16  * Main Class of Project, responsible for managing the Socketserver and keeping
17  * track of all users/connected clients
18 */
19 public class WebsocketManager {
20
21     private static WebsocketManager instance;
22     private final CommandManager commandManager;
23     private final Map<String, User> userList;
24     private IWebsocketServer server;
25     private final Map<String, Conference> conferences;
26
27     /**
28      * Assigns the attributes their values, initializes the program
29      */
30     public WebsocketManager() {
31         // Check if Environment-variable is set, abort program if not
32         if (System.getenv("WEBSOCKET_PORT") == null) {
33             throw new RuntimeException("Serversocket port is not set, aborting
34             Program!");
35         }
36         // Create Instance of our Websocketserver and start it
37         this.server = new
38             IWebsocketServer(Integer.parseInt(System.getenv("WEBSOCKET_PORT")));
39         // Start websocketserver and handle incoming connections
40         this.server.start();
41         this.userList = new HashMap<>();
42         this.commandManager = new CommandManager(this);
43         this.conferences = new HashMap<>();
44         instance = this;
45         // register all commands
46         this.registerCommands();
47         System.out.println("Websocket started, waiting for connections");
48     }
49
50     /**
51      * @return instance of this class, used on multiple accounts through the program
52     */
53     public static WebsocketManager getInstance() {
54         return instance;
55     }
56
57     public static void main(String[] args) {
58         new WebsocketManager();
59     }
60
61     /**
62      * Create instances of all commands an register them for execution
63      */
64     private void registerCommands() {
65         this.commandManager.registerCommands(
66             new Login(),
67             new SetupConference(),
68             new ControlConference()
69         );
70     }
71
72     /**
73
```

```
71  * Responsible for adding users to the User-Map, indexed by their Identifier
72  *
73  * @param user User to add to the User-Map
74  */
75  public void addUser(User user) {
76      this.userList.put(user.getIdentifier(), user);
77  }
78
79 /**
80  * Responsible for removing users from the User-Map, indexed by their Identifier
81  *
82  * @param user User to remove from the User-Map
83  */
84  public void removeUser(User user) {
85      this.userList.remove(user.getIdentifier());
86  }
87
88 /**
89  * @param identifier Identifier of the user to retrieve
90  * @return User instance with given identifier
91  */
92  public User getUser(String identifier) {
93      return this.userList.get(identifier);
94  }
95
96  public IWebsocketServer getServer() {
97      return server;
98  }
99
100 public Map<String, User> getUserList() {
101     return userList;
102 }
103
104 public CommandManager getCommandManager() {
105     return commandManager;
106 }
107
108 /**
109  * @return the Map with all conferences, key is the session identifier code and
110  * value is the conference object
111  */
112 public Map<String, Conference> getConferences() {
113     return conferences;
114 }
115
116
117 }
118 }
```

## WebsocketManager.java

```
1 package space.kappes.entity;
2
3 import org.java_websocket.WebSocket;
4 import org.json.JSONObject;
5
6 import java.util.ArrayList;
7 import java.util.UUID;
8
9 /**
10  * Responsible for managing a connection and the messages from one end-user
11 */
12 public class User {
13     private final WebSocket socket;
14     private final String identifier; // Random string used for differentiating clients
15     private final ArrayList<NotificationSettings> subscribedEvents;
16     private boolean privileged;
17     private String code;
18
19 /**
20  * User object representing a client/end-user in the frontend
21  * @param socket Socket object received by accepting a connection, used for
22  * further communication with the user/client this object represents
23 */
24     public User(WebSocket socket) {
25         this.socket = socket;
26         this.identifier = UUID.randomUUID().toString().replace("-", "").substring(0, 7);
27         this.subscribedEvents = new ArrayList<>();
28         this.privileged = false;
29         this.code = "";
30     }
31
32 /**
33  * Send instructions to a client
34  * @param command invoke for the command sent to the client
35  * @param json data which will be send to the client, requires json-format
36 */
37     public void send(String command, JSONObject json) {
38         // Send command-string to client, format: invoke + SPACE + Json-Data
39         this.socket.send(command + " " + json.toString());
40     }
41
42     public WebSocket getSocket() {
43         return socket;
44     }
45
46     public String getIdentifier() {
47         return identifier;
48     }
49
50     public ArrayList<NotificationSettings> getSubscribedEvents() {
51         return subscribedEvents;
52     }
53
54 /**
55  * @param notificationsettings Notificationsettings instance which a user
56  * subscribed to
57 */
58     public void addToSubscribedEvent(NotificationSettings notificationsettings) {
59         this.subscribedEvents.add(notificationsettings);
60     }
61
62 /**
63  * Removes the subscription of a event
64  * @param eventIndex index of the event, which should be removed
65 */
66     public void removeFromSubscribedEvent(int eventIndex) {
67         // Remove an instance if the has the same index as the parameter
68         this.subscribedEvents.removeIf(i -> i.getEventIndex() == eventIndex);
69     }
70
71     public boolean isPrivileged() {
72         return privileged;
73     }
74 }
```

```
72     public void setPrivileged(boolean privileged) {
73         this.privileged = privileged;
74     }
75
76     public String getCode() {
77         return code;
78     }
79
80     /**
81      * @param code Session identifier, identifying to which conference this user
82      * belongs
83      */
84     public void setCode(String code) {
85         this.code = code;
86     }
87 }
88
```

## User.java

```
1 package space.kappes.commands;
2
3 import org.json.JSONArray;
4 import org.json.JSONObject;
5 import space.kappes.WebsocketManager;
6 import space.kappes.command.Command;
7 import space.kappes.entity.Conference;
8 import space.kappes.entity.ConferenceItem;
9 import space.kappes.entity.User;
10
11 public class SetupConference extends Command {
12
13     /**
14      * Used for setting up a conference, need argument with conference object
15      * @see Command#Command(String, String[])
16      */
17     public SetupConference() {
18         super("setupConference", new String[]{"conference", "code"});
19     }
20
21     /**
22      * @see Command#run(WebsocketManager, User, JSONObject)
23      */
24     @Override
25     public void run(WebsocketManager manager, User user, JSONObject args) {
26         // The json-data contains an key named conference which holds an array of
27         // conference items as its values
28         JSONArray jsonArray = args.getJSONArray("conference");
29         // Create new instance of the conference class
30         Conference conference = new Conference();
31         // Run the follow for each conference-item in the array
32         jsonArray.forEach((item) -> {
33             // Cast the object to an jsonobject
34             JSONObject object = (JSONObject) item;
35             // Check if the description is empty, if not proceed. Frontend should
36             // never send an item without description so we can be sure that if it is
37             // available the rest works too
38             if(object.has("Beschreibung") && !object.isEmpty() &&
39                 !object.getString("Beschreibung").isEmpty())
40                 // Add new item to the list of conference-items, fill its values by
41                 // retrieving them from the jsonobject
42                 conference.getConferenceItems().add(new
43                     ConferenceItem(object.getString("Beschreibung"),
44                         object.getString("Start"), object.getInt("Dauer")));
45             });
46             // Set the first conference item as the currently active
47             conference.setCurrentItem(conference.getConferenceItems().get(0));
48             // Add the newly created conference to the conference-pool
49             manager.getConferences().put(args.getString("code"), conference);
50         }
51     }
52 }
```

## SetupConference.java

```
1 package space.kappes.commands;
2
3 import org.json.JSONObject;
4 import space.kappes.WebsocketManager;
5 import space.kappes.command.Command;
6 import space.kappes.entity.Conference;
7 import space.kappes.entity.User;
8
9 public class Login extends Command {
10
11     /**
12      * Used for authenticating on first connect, argument tells if user is conference
13      * leader or not
14      * @see Command#Command(String, String[])
15      */
16     public Login() {
17         super("login", new String[]{"privileged", "code"});
18     }
19
20     /**
21      * @see Command#run(WebsocketManager, User, JSONObject)
22      */
23     @Override
24     public void run(WebsocketManager manager, User user, JSONObject args) {
25         boolean privileged = args.getBoolean("privileged");
26         String code = args.getString("code");
27         user.setPrivileged(privileged);
28         user.setCode(code);
29         Conference conference = manager.getConferences().get(code);
30         // Add identifier and the conference Object to the json-data
31         JSONObject conferenceObject = new JSONObject().put("identifier",
32             user.getIdentifier()).put("conference", new JSONObject(conference));
33         // Send update command with json-object
34         user.send("updateConference", conferenceObject);
35     }
36 }
```

## Login.java

```
1 package space.kappes.command;
2
3 import org.json.JSONObject;
4 import space.kappes.WebsocketManager;
5 import space.kappes.entity.User;
6
7 /**
8 * Class on which all commands extend
9 */
10 public abstract class Command {
11     private final String invoke;
12     private String[] requiredArgs = new String[]{};
13
14     /**
15      * Constructs a command with no required arguments
16      * @param invoke Command invoke/name on which this command should be run
17      */
18     public Command(String invoke) {
19         this.invoke = invoke;
20     }
21
22     /**
23      * Constructs a command with required arguments, if they are not met a error is
24      * returned
25      * @param invoke Command invoke/name on which this command should be run
26      * @param requiredArgs array of string which have to be in the json-data to meet
27      * this commands requirements
28      */
29     public Command(String invoke, String[] requiredArgs) {
30         this.invoke = invoke;
31         this.requiredArgs = requiredArgs;
32     }
33
34     /**
35      * Method which gets run by the Commandmanager, when a matching command is
36      * received
37      * @param manager Instance of the Managerclass
38      * @param user User which called this command
39      * @param args arguments provided by command call
40      */
41     public abstract void run(WebsocketManager manager, User user, JSONObject args);
42
43     public String getInvoke() {
44         return invoke;
45     }
46
47     public String[] getRequiredArgs() {
48         return requiredArgs;
49     }
50
51     public void setRequiredArgs(String[] requiredArgs) {
52         this.requiredArgs = requiredArgs;
53     }
54 }
```

## Command.java

```
1 package space.kappes.commands;
2
3 import org.joda.time.DateTime;
4 import org.joda.time.Duration;
5 import org.joda.time.format.DateTimeFormat;
6 import org.joda.time.format.DateTimeFormatter;
7 import org.json.JSONObject;
8 import space.kappes.WebsocketManager;
9 import space.kappes.command.Command;
10 import space.kappes.entity.Conference;
11 import space.kappes.entity.User;
12
13 import java.text.ParseException;
14 import java.text.SimpleDateFormat;
15 import java.util.Date;
16
17 public class ControlConference extends Command {
18
19
20     /**
21      * Used for navigating inside a conference, need argument of which item to
22      * switch to
23      * @see Command#Command(String, String[])
24      */
25     public ControlConference() {
26         super("controlConference", new String[]{"switchTo", "code"});
27     }
28
29     /**
30      * @see Command#run(WebsocketManager, User, JSONObject)
31      */
32     @Override
33     public void run(WebsocketManager manager, User user, JSONObject args) {
34         String code = args.getString("code");
35         // Abort if user has no permission
36         if(!user.isPrivileged())
37             return;
38         // Abort if no conference is set, yet
39         if(manager.getConferences().get(code) == null)
39             return;
40         Conference conference = manager.getConferences().get(code);
41         // Get the conference by its code, then set the provided index as current
42
43         conference.setCurrentItem(manager.getConferences().get(code).getConferenceItems()
44             .get(args.getInt("switchTo")));
45         // Formatter is set to the time-format we expect ( full hour:full minutes )
46         DateTimeFormatter formatter = DateTimeFormat.forPattern("HH:mm");
47         // Parse our starting time to the format
48         DateTime start =
49             formatter.parseDateTime(conference.getCurrentItem().getStart());
50         // Reformat the current time to the format
51         DateTime current = formatter.parseDateTime(new
52             DateTime().toString(formatter));
53         // calculate the duration between both times
54         Duration duration = new Duration(current, start);
55         try {
56             // Cast the minutes to a whole and set them as delay
57             conference.setDelay(Math.toIntExact(duration.getStandardMinutes() * -1));
58         } catch (ArithmaticException ignored) {
59             // Do not set delay if an error occurs
60         }
61
62         // Get all users, which subscribed to this conference, and then filter for
63         // the client which send the instruction, so he doesn't get the update
64         manager.getUserList().values().stream().filter(c ->
65             c.getCode().equals(code)).forEach((client) -> {
66             // Send user the updated conference object
67             client.send("updateConference", new JSONObject().put("identifier",
68                 user.getIdentifier()).put("conference", new JSONObject(conference)));
69         });
70     }
71 }
```

## ControlConference.java

```
1 package space.kappes.entity;
2
3 import java.util.ArrayList;
4
5 /**
6  * Class representing a conference
7  */
8 public class Conference {
9
10    private final ArrayList<ConferenceItem> conferenceItems; // Ordered list of all
11    items in the conference
12    private int delay; // total delay, negative or positive
13    private ConferenceItem currentItem;
14
15    public Conference() {
16        this.conferenceItems = new ArrayList<>();
17        this.delay = 0;
18        this.currentItem = null;
19    }
20
21    public ArrayList<ConferenceItem> getConferenceItems() {
22        return conferenceItems;
23    }
24
25    public int getDelay() {
26        return delay;
27    }
28
29    public void setDelay(int delay) {
30        this.delay = delay;
31    }
32
33    public ConferenceItem getCurrentItem() {
34        return currentItem;
35    }
36
37    public void setCurrentItem(ConferenceItem currentItem) {
38        this.currentItem = currentItem;
39    }
40}
```

## Conference.java

```
1 package space.kappes.entity;
2
3 /**
4  * Class representing a single item in a conference
5  */
6 public class ConferenceItem {
7
8    private final String description;
9    private final String start;
10   private final int duration;
11
12   /**
13    * @param description Description of the item, something like 9a,8c,Q1...
14    * @param start Starting time of the item, something like 8:00, 8:55...
15    * @param duration duration of that item, something like 30, 40. Duration is in
16    * minutes
17    */
18   public ConferenceItem(String description, String start, int duration) {
19       this.description = description;
20       this.start = start;
21       this.duration = duration;
22   }
23
24   public String getDescription() {
25       return description;
26   }
27
28   public String getStart() {
29       return start;
30   }
31
32   public int getDuration() {
33       return duration;
34   }
35 }
```

## ConferenceItem.java

```
1 package space.kappes.entity;
2
3 /**
4  * Class representing the selected settings of a user
5  */
6 public class NotificationSettings {
7
8     private final int eventIndex;
9     private boolean nextUp;
10
11    /**
12     * @param eventIndex index of the event on which notifications should be sent
13     * @param nextUp send notification if this item is the next up
14     */
15    public NotificationSettings(int eventIndex, boolean nextUp) {
16        this.eventIndex = eventIndex;
17        this.nextUp = nextUp;
18    }
19
20    public int getEventIndex() {
21        return eventIndex;
22    }
23
24    public boolean isNextUp() {
25        return nextUp;
26    }
27
28    public void setNextUp(boolean nextUp) {
29        this.nextUp = nextUp;
30    }
31
32 }
33 }
```

### NotificationSettings.java

```
1 package space.kappes.net;
2
3 import org.java_websocket.WebSocket;
4 import org.java_websocket.handshake.ClientHandshake;
5 import org.java_websocket.server.WebSocketServer;
6 import space.kappes.WebsocketManager;
7 import space.kappes.entity.User;
8
9 import java.io.IOException;
10 import java.net.InetSocketAddress;
11
12 /**
13 * Responsible for handling the Websocketserver, e.g. all incoming connections
14 */
15 public class IWebsocketServer extends WebSocketServer {
16     private final int port;
17
18     /**
19      * Initializes the Socketserver
20      */
21     public IWebsocketServer(int port) {
22         super(new InetSocketAddress(port));
23         this.port = port;
24     }
25
26
27     /**
28      * Called when the Socketserver shuts down, stops the server
29      * @throws InterruptedException thrown when there was an error closing the
30      * Socketserver
31      * @throws IOException thrown when there was an error closing the Socketserver
32      */
33     public void destroy() throws InterruptedException, IOException {
34         this.stop();
35     }
36
37     /**
38      * Called when a new Client opens a connection and the handshake is done
39      * @param conn New connection, which has just done the required handshake
40      * @param handshake Handshake itself, containing infos over the http connection
41      */
42     @Override
43     public void onOpen(WebSocket conn, ClientHandshake handshake) {
44         // Create new User object
45         User user = new User(conn);
46         // Add user to main register, used for calling all users
47         WebsocketManager.getInstance().addUser(user);
48         // Attach the identifier to the client so we can later differentiate the
49         // clients
50         conn.setAttachment(user.getIdentifier());
51     }
52
53     /**
54      * @param conn Connection object of the client, many methods not longer working
55      * because of disconnected connection
56      * @param code disconnect code
57      * @param reason Reason for the disconnect
58      * @param remote true when the client disconnected himself, false if the server
59      * disconnected him
60      */
61     @Override
62     public void onClose(WebSocket conn, int code, String reason, boolean remote) {
63         // Remove the user by its attached identifier
64         WebsocketManager.getInstance().removeUser(WebsocketManager.getInstance().getUser(conn.getAttachment()));
65     }
66
67     /**
68      * Called whenever a message is received by a client
69      * @param conn Connection instance of the client
70      * @param message Message received from client
71      */
72 }
```

```
68     @Override
69     public void onMessage(WebSocket conn, String message) {
70         User user = WebsocketManager.getInstance().getUser(conn.getAttachment());
71         WebsocketManager.getInstance().getCommandManager().parseCommand(message,
72             user);
73     }
74     /**
75      * @param conn Connection instance of the client, which ran into an error
76      * @param ex Exception which occurred
77      */
78     @Override
79     public void onError(WebSocket conn, Exception ex) {
80         // -
81     }
82     /**
83      * Run when the start method is called on this object
84      */
85     @Override
86     public void onStart() {
87         // -
88     }
89     /**
90      * @return the port on which the server listens
91      */
92     @Override
93     public int getPort() {
94         return port;
95     }
96 }
97 }
98 }
99 }
```

### IWebsocketServer.java

```
1 package space.kappes.command;
2
3 import org.json.JSONObject;
4 import space.kappes.WebsocketManager;
5 import space.kappes.entity.User;
6
7 import java.util.HashMap;
8 import java.util.Map;
9
10 /**
11  * Class responsible for parsing and executing all incoming commands
12 */
13 public class CommandManager {
14
15     private final WebsocketManager manager;
16     private final Map<String, Command> commandMap;
17
18     /**
19      * Creates an instance of the commandmanager, responsible for parsing and
20      * executing all incoming commands
21      * @param manager Main instance of this project, used for accessing various
22      * other objects
23      */
24     public CommandManager(WebsocketManager manager) {
25         this.manager = manager;
26         this.commandMap = new HashMap<>();
27     }
28
29     /**
30      * Registers an command to be parsed if received
31      * @param command command to register
32      */
33     public void registerCommand(Command command) {
34         commandMap.put(command.getInvoke(), command);
35     }
36
37     /**
38      * Iteratively calls {@link #registerCommand(Command)} for every passed command
39      * @param commands Multiple commands to register
40      */
41     public void registerCommands(Command... commands) {
42         for (Command command : commands)
43             registerCommand(command);
44     }
45
46     /**
47      * Parses an incoming message into an command and executes it when existing
48      * @param message Message which is received from websocket
49      * @param user user from whom the message origins
50      */
51     public void parseCommand(String message, User user) {
52         // Invoke is determent by the first word before an whitespace, here the
53         // string is split at whitespaces and the returning array are all words
54         String invoke = message.split("\\s") [0];
55         // Abort if this command isn't registered
56         if(!commandMap.containsKey(invoke))
57             return;
58         Command command = commandMap.get(invoke);
59         // JSON-Data is everything after the invoke plus the trailing whitespace
60         JSONObject jsonObject = new JSONObject(message.replace(invoke+" ",""));
61         // Loop through all required arguments and check if the exist, return an
62         // error if not so
63         for (String required: command.getRequiredArgs()) {
64             if (!jsonObject.has(required)) {
65                 user.send("error", new JSONObject().put("error", "Invalid
66                 Command").put("message", String.format("Missing argument: %s",
67                 required)));
68             }
69         }
70         // Run the logic of the command, gets passed the json-data, the user and
71         // manager
72         command.run(manager, user, jsonObject);
73     }
74 }
```

## CommandManager.java

```
1 import colors from 'vuetify/es5/util/colors'
2
3 export default {
4   // Global page headers: https://go.nuxtjs.dev/config-head
5   head: {
6     title: 'Konferenztool - Gymnasium Norf',
7     meta: [
8       { charset: 'utf-8' },
9       { name: 'viewport', content: 'width=device-width, initial-scale=1' },
10      {
11        hid: 'description',
12        name: 'description',
13        content: 'Konferenztool des Gymnasium Norf',
14      },
15      {
16        hid: 'og:description',
17        name: 'og:description',
18        content: 'Konferenztool des Gymnasium Norf',
19      },
20    ],
21    link: [{ rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' }],
22  },
23
24 // Global CSS: https://go.nuxtjs.dev/config-css
25 css: [],
26
27 // Plugins to run before rendering page: https://go.nuxtjs.dev/config-plugins
28 plugins: [{ src: '~/plugins/websocket.ts', mode: 'client' }],
29
30 // Auto import components: https://go.nuxtjs.dev/config-components
31 components: true,
32
33 // Modules for dev and build (recommended): https://go.nuxtjs.dev/config-modules
34 buildModules: [
35   // https://go.nuxtjs.dev/typescript
36   '@nuxt/typescript-build',
37   // https://go.nuxtjs.dev/vuetify
38   '@nuxtjs/vuetify',
39 ],
40
41 // Modules: https://go.nuxtjs.dev/config-modules
42 modules: [
43   // https://go.nuxtjs.dev/axios
44   '@nuxtjs/axios',
45   // https://go.nuxtjs.dev/pwa
46   '@nuxtjs/pwa',
47 ],
48
49 // Axios module configuration: https://go.nuxtjs.dev/config-axios
50 axios: {},
51
52 // PWA module configuration: https://go.nuxtjs.dev/pwa
53 pwa: {
54   manifest: {
55     lang: 'de',
56     display: 'standalone',
57     name: 'Konferenztool Gymnasium Norf',
58     short_name: 'Konferenztool',
59   },
60 },
```

```
61 // Vuetify module configuration: https://go.nuxtjs.dev/config-vuetify
62 vuetify: {
63   customVariables: ['~/assets/variables.scss'],
64   theme: {
65     dark: true,
66     themes: {
67       dark: {
68         primary: colors.blue.darken2,
69         accent: colors.grey.darken3,
70         secondary: colors.amber.darken3,
71         info: colors.teal.lighten1,
72         warning: colors.amber.base,
73         error: colors.deepOrange.accent4,
74         success: colors.green.accent3,
75       },
76     },
77   },
78 },
79 },
80
81 // Build Configuration: https://go.nuxtjs.dev/config-build
82 build: {},
83 }
84 }
```

nuxt.config.js

```
1 {
2   "wsURL": "ws://localhost:3333"
3 }
4 }
```

assets/config.json

```
1 <template>
2   <v-list>
3     <!-- Delay Header, only show if we have an active conference -->
4     <v-subheader v-if="items.length > 0"
5       >Verzug: <span :class="textColor"> &nbsp;{{ delay }}m</span></v-subheader>
6   >
7   <!-- Create the create and join buttons, also only shown when no active
conference -->
8   <v-list-item v-if="items.length == 0" @click="$router.push('settings')">
9     <v-list-item-content>
10      <v-list-item-title>Konferenz einrichten</v-list-item-title>
11      <v-list-item-subtitle>
12        >Erstelle eine neue Konferenz
13      </v-list-item-subtitle>
14      </v-list-item-content>
15    </v-list-item>
16    <v-list-item v-if="items.length == 0" @click="dialog2 = true">
17      <v-list-item-content>
18        <v-list-item-title>Konferenz beitreten</v-list-item-title>
19        <v-list-item-subtitle>
20          >Einer vorhanden Konferenz beitreten
21        </v-list-item-subtitle>
22      </v-list-item-content>
23    </v-list-item>
24  <!-- Actual Conference -->
25  <v-list-item-group v-model="current" active-class="" multiple>
26    <!-- Create the conference-items in an loop -->
27    <template v-for="(item, index) in items">
28      <conference-item
29        :key="item.description"
30        :title="item.description"
31        :time="item.start"
32        :index="index"
33      ></conference-item>
34      <v-divider v-if="index < items.length - 1" :key="index"></v-divider>
35    </template>
36  </v-list-item-group>
37  <!-- Dialogs -->
38  <!-- Dialog show when your about to switch the conference item, show consequences
-->
39  <v-dialog v-model="dialog" max-width="290">
40    <v-card>
41      <v-card-title class="headline">
42        Konferenz wirklich wechseln?
43      </v-card-title>
44
45      <v-card-text>
46        Ein Wechsel wird sofort bei allen verbundenen Teilnehmern angezeigt
47        und löst zudem, bei Teilnehmern die dies aktiviert haben, eine
48        Benachrichtigung aus.
49      </v-card-text>
50
51      <v-card-actions>
52        <v-spacer></v-spacer>
53
54        <v-btn color="green darken-1" text @click="dialog = false">
55          Abbrechen
56        </v-btn>
57
58        <v-btn color="green darken-1" text @click="accept"> Wechseln </v-btn>
```

```
59      </v-card-actions>
60    </v-card>
61  </v-dialog>
62  <!-- Dialog asking you for the conference-id when your joining an active
conference --&gt;
63  &lt;v-dialog v-model="dialog2" max-width="290"&gt;
64    &lt;v-card&gt;
65      &lt;v-card-title class="headline"&gt; Konferenz beitreten &lt;/v-card-title&gt;
66
67      &lt;v-card-text&gt;
68        Bitte den Beitrtscode angeben. Diesen erhalten sie vom
69        Konferenzleiter
70        &lt;v-text-field
71          v-model="input"
72          label="Konferenzcode"
73          single-line
74        &gt;&lt;/v-text-field&gt;
75      &lt;/v-card-text&gt;
76
77      &lt;v-card-actions&gt;
78        &lt;v-spacer&gt;&lt;/v-spacer&gt;
79
80        &lt;v-btn color="green darken-1" text @click="dialog2 = false"&gt;
81          Abbrechen
82        &lt;/v-btn&gt;
83
84        &lt;v-btn color="green darken-1" text @click="join"&gt; Beitreten &lt;/v-btn&gt;
85      &lt;/v-card-actions&gt;
86    &lt;/v-card&gt;
87  &lt;/v-dialog&gt;
88 &lt;/v-list&gt;
89 &lt;/template&gt;
90
91 &lt;script&gt;
92 import { mapState } from 'vuex'
93 import ConferenceItem from '~/components/ConferenceItem.vue'
94
95 export default {
96   components: {
97     ConferenceItem,
98   },
99   data() {
100     return {
101       current: [0],
102       dialog: false,
103       dialog2: false,
104       input: '',
105       dialogIndex: 0,
106     }
107   },
108   computed: {
109     textColor() {
110       // This methods decides in which color the delay is shown, when delay is
111       // greater then 0 then it is red else it is green
112       return this.delay === 0
113         ? ''
114         : this.delay &lt; 0
115         ? 'green--text'
116         : 'red--text'
117     },
118   }
119 }</pre>
```

```
117     ...mapState(['items', 'delay']),
118 },
119 mounted() {
120     // Add a listener to a function, which saves everything when you close a tab
121     window.addEventListener('beforeunload', this.handler)
122     // Event which gets triggered by a conferenceitem when you pressed the set as
123     // active button, initiates the confirmation dialog
124     this.$root.$on('askConfirmation', (index) => {
125         this.dialog = true
126         this.dialogIndex = index
127     })
128     // Websocket listener, triggered when the conference gets updated
129     this.$websocket.on('updateConference', (data) => {
130         // data is a string, parse it to an object
131         const json = JSON.parse(data)
132         // Set the items and delay in storage, triggers an automatic update in the ui
133         this.$store.commit('setItems', json.conference.conferenceItems)
134         this.$store.commit('setDelay', json.conference.delay)
135         // Loop through the items and set the active one in the ui
136         for (let index = 0; index < this.items.length; index++) {
137             const element = this.items[index]
138             if (element.description === json.conference.currentItem.description) {
139                 this.$store.commit('setCurrent', index)
140             }
141         }
142         // Save everything to localstorage in the event the application gets closed
143         localStorage.setItem('code', this.$store.state.code)
144         localStorage.setItem('items', JSON.stringify(this.$store.state.items))
145     })
146     // Code which runs when the app gets opened
147     // Load the conference out of storage, if available
148     if (
149         this.items.length === 0 &&
150         localStorage.getItem('code') !== null &&
151         localStorage.getItem('code') !== ''
152     ) {
153         this.$store.commit('setCode', localStorage.getItem('code'))
154         this.$store.commit(
155             'setPrivileged',
156             localStorage.getItem('privileged') === 'true'
157         )
158         this.$store.commit(
159             'setCurrent',
160             parseInt(localStorage.getItem('current'))
161         )
162         this.$store.commit('setDelay', parseInt(localStorage.getItem('delay')))
163         this.$store.commit('setItems', JSON.parse(localStorage.getItem('items')))
164         // Login again, so the server knows we want future update
165         this.$websocket.send(
166             'login',
167             `{"privileged": ${this.$store.state.privileged}, "code": ${this.$store.state.code}}`
168         )
169     },
170     beforeDestroy() {
171         // before unloading this component, unregister the listener, so we dont save
172         // incorrect data
173         window.removeEventListener('beforeunload', this.handler)
174     }
175 }
```

```
174 | methods: {
175 |   // Method responsible for saving everything in case of app close
176 |   handler(event) {
177 |     event.preventDefault()
178 |     localStorage.setItem('code', this.$store.state.code)
179 |     localStorage.setItem('privileged', this.$store.state.privileged)
180 |     localStorage.setItem('current', this.$store.state.current)
181 |     localStorage.setItem('delay', this.$store.state.delay)
182 |     localStorage.setItem('items', JSON.stringify(this.$store.state.items))
183 |   },
184 |   // Method triggered if the switch dialog was confirmed
185 |   accept() {
186 |     // Set new current item
187 |     this.$store.commit('setCurrent', this.dialogIndex)
188 |     // Send switch to server so all clients get the update
189 |     this.$websocket.send(
190 |       'controlConference',
191 |       `{"switchTo": ${this.dialogIndex}, "code": "${this.$store.state.code}"}`)
192 |   },
193 |   this.dialog = false
194 | },
195 | // Method triggered if you join a conference
196 | join() {
197 |   // Set the conference code your now connected to
198 |   this.$store.commit('setCode', this.input)
199 |   this.input = ''
200 |   // Login with the server to receive the conference information
201 |   this.$websocket.send(
202 |     'login',
203 |     `{"privileged": false, "code": "${this.$store.state.code}"}`)
204 |   ),
205 |   this.dialog2 = false
206 | },
207 | },
208 | }
209 | </script>
210 |
211 | <style></style>
212 |
```

pages/index.vue

```
1 <template>
2   <v-list>
3     <!-- Button for setting up a conference -->
4     <v-list-item v-if="!!!code" @click="toggleUpload">
5       <v-list-item-content>
6         <v-list-item-title>Konferenzplan setzen</v-list-item-title>
7         <v-list-item-subtitle>
8           >Erfordert eine CSV-Datei, nach folgenden Format:<br />
9             Beschreibung (z.B. „9e“ oder „Pause“), Start (z.B. „9:15 Uhr“)
10            <br />
11            und Dauer in Minuten (z.B. „25“)
12          </v-list-item-subtitle>
13        </v-list-item-content>
14      </v-list-item>
15      <!-- Button displaying the conference-code, copy to clipboard on press -->
16      <v-list-item v-if="!!code" @click="copy">
17        <v-list-item-content>
18          <v-list-item-title>
19            >Konferenzcode:
20            <span class="font-weight-black">{{ code }}</span></v-list-item-title>
21          </v-list-item-content>
22        </v-list-item>
23      </v-list-item>
24      <!-- Dialog for uploading a .csv sheet -->
25      <v-dialog v-model="dialog" max-width="290">
26        <v-card>
27          <v-card-title class="headline"> Datei hochladen </v-card-title>
28
29          <v-card-text>
30            <v-container>
31              <v-file-input
32                accept=".csv"
33                label="CSV-Tabelle"
34                @change="selectFile"
35              ></v-file-input>
36            </v-container>
37          </v-card-text>
38
39          <v-card-actions>
40            <v-spacer></v-spacer>
41            <v-btn color="blue darken-1" text @click="dialog = false">
42              Abbrechen
43            </v-btn>
44            <v-btn color="blue darken-1" text @click="sendUpload">
45              Speichern
46            </v-btn>
47          </v-card-actions>
48        </v-card>
49      </v-dialog>
50      <!-- Dialog showing the code after creation -->
51      <v-dialog v-model="dialog2" max-width="290">
52        <v-card>
53          <v-card-title class="headline"> Konferenz erstellt </v-card-title>
54
55          <v-card-text>
56            <v-container>
57              Der Konferenz code ist:
58              <span class="font-weight-black">{{ code }}</span>
59            </v-container>
60          </v-card-text>
```

```
61      <v-card-actions>
62          <v-spacer></v-spacer>
63          <v-btn color="blue darken-1" text @click="dialog2 = false">
64              Schließen
65          </v-btn>
66      </v-card-actions>
67  </v-card>
68 </v-dialog>
69 <!-- Notification when copying the code -->
70 <v-snackbar v-model="snackbar" timeout="2000">
71     Code kopiert!
72     <template #action="{ attrs }">
73         <v-btn color="blue" text v-bind="attrs" @click="snackbar = false">
74             Schließen
75         </v-btn>
76     </template>
77 </v-snackbar>
78 <!-- Notification when wrong format -->
79 <v-snackbar v-model="snackbar2" timeout="2000">
80     Das Format der CSV-Datei ist falsch!
81     <template #action="{ attrs }">
82         <v-btn color="blue" text v-bind="attrs" @click="snackbar2 = false">
83             Schließen
84         </v-btn>
85     </template>
86 </v-snackbar>
87 </v-list>
88 </template>
89 </script>
90 import { mapState } from 'vuex'
91 import * as Papa from 'papaparse'
92
93 export default {
94     data() {
95         return {
96             dialog: false,
97             dialog2: false,
98             file: undefined,
99             snackbar: false,
100            snackbar2: false,
101        }
102    },
103    computed: mapState(['code']),
104    methods: {
105        // Shows the upload dialog
106        toggleUpload() {
107            this.dialog = true
108        },
109        // Sets the uploaded file as an attribute to this object
110        selectFile(file) {
111            this.file = file
112        },
113        // Copies the code to the clipboard, displays notification if successfull
114        copy() {
115            navigator.clipboard.writeText(this.code).then(() => {
116                this.snackbar = true
117            })
118        }
119    }
120}
```

```
121 // Method responsible for parsing the sheet and sending the conference to the
122 server
123 sendUpload() {
124     // Read the file contents
125     const reader = new FileReader()
126     reader.readAsText(this.file)
127     reader.onloadend = () => {
128         // Parse the csv to a js-object
129         Papa.parse(reader.result, {
130             complete: (result) => {
131                 // Check if we have a result
132                 if (result.data[0]) {
133                     // Only run code if we have all required headers
134                     if (
135                         result.data[0].Beschreibung &&
136                         result.data[0].Dauer &&
137                         result.data[0].Start
138                     ) {
139                         // Generate a new random code based on time and a random number
140                         const code = (+new Date() + Math.random())
141                             .toString(36)
142                             .slice(-7)
143                             .replace('.', '')
144                         // Send the newly parsed conference and the generated code to the
145                         // server
146                         this.$websocket.send(
147                             'setupConference',
148                             `{"conference": ${JSON.stringify(
149                                 result.data
150                             )}, "code": "${code}" }`
151                         )
152                         // Login to the server as the conference host
153                         this.$websocket.send(
154                             'login',
155                             `{"privileged": true, "code": "${code}" }`
156                         )
157                         // Set everything in the app context
158                         this.$store.commit('setCode', code)
159                         this.$store.commit('setPrivileged', true)
160                         // Show code to user
161                         this.dialog2 = true
162                         } else {
163                             this.snackbar2 = true
164                         }
165                         } else {
166                             this.snackbar2 = true
167                         }
168                     },
169                     header: true,
170                 )
171             }
172             // reset all attributes
173             this.dialog = false
174             this.file = undefined
175         },
176     </script>
177     <style></style>
```

pages/settings.vue

```
1 <template>
2   <v-app dark>
3     <!-- navigation -->
4     <v-navigation-drawer v-model="drawer" temporary absolute>
5       <v-list dense>
6         <!-- Main page with current conference and join button | see pages/index.vue
-->
7         <v-list-item key="Zeitplan" link nuxt to="/">
8           <v-list-item-icon>
9             <v-icon>mdi-view-dashboard</v-icon>
10            </v-list-item-icon>
11
12           <v-list-item-content>
13             <v-list-item-title>Zeitplan</v-list-item-title>
14           </v-list-item-content>
15         </v-list-item>
16         <!-- settings page, used for settings up a conference | see
pages/settings.vue -->
17         <v-list-item
18           v-if="privileged"
19           key="Konferenz verwalten"
20           link
21           nuxt
22           to="settings"
23         >
24           <v-list-item-icon>
25             <v-icon>mdi-forum</v-icon>
26           </v-list-item-icon>
27
28           <v-list-item-content>
29             <v-list-item-title>Konferenz verwalten</v-list-item-title>
30           </v-list-item-content>
31         </v-list-item>
32         <!-- leave button, see method leave below -->
33         <v-list-item v-if="!!code" key="Konferenz verlassen" @click="leave">
34           <v-list-item-icon>
35             <v-icon>mdi-close</v-icon>
36           </v-list-item-icon>
37
38           <v-list-item-content>
39             <v-list-item-title>Konferenz verlassen</v-list-item-title>
40           </v-list-item-content>
41         </v-list-item>
42       </v-list>
43     </v-navigation-drawer>
44   <v-main>
45     <!-- Main header on all pages, has navigation button -->
46     <v-container>
47       <v-card class="mx-auto">
48         <v-toolbar color="blue" dark>
49           <v-app-bar-nav-icon
50             @click.stop="drawer = !drawer"
51           ></v-app-bar-nav-icon>
52           <v-toolbar-title>Konferenztool</v-toolbar-title>
53           <v-spacer></v-spacer>
54         </v-toolbar>
55         <!-- in runtime this element will be replaced with the content of the
current page | see pages directory -->
56         <nuxt />
57       </v-card>
```

```
58      </v-container>
59    </v-main>
60  </v-app>
61 </template>
62
63<script>
64 import { mapState } from 'vuex'
65
66 export default {
67   data() {
68     return {
69       drawer: null,
70     }
71   },
72   computed: mapState(['privileged', 'code']),
73   methods: {
74     // Method which gets triggered when you leave a conference, deletes everything
75     // out of storage and out of the context
76     leave() {
77       this.drawer = false
78       this.$store.commit('setCode', '')
79       this.$store.commit('setPrivileged', false)
80       this.$store.commit('setCurrent', 0)
81       this.$store.commit('setItems', [])
82       this.$store.commit('setDelay', 0)
83       localStorage.removeItem('code')
84       localStorage.removeItem('privileged')
85       localStorage.removeItem('items')
86       localStorage.removeItem('current')
87       localStorage.removeItem('delay')
88       this.$router.push('/')
89     },
90   }
91 </script>
92
93<style></style>
94
```

layout/default.vue

```
1 import { EventEmitter } from 'events'
2 import * as config from '~/assets/config.json'
3
4 /**
5  * Class responsible for the websocket connection
6 */
7 class WebsocketWrapper extends EventEmitter {
8     private webSocket: WebSocket
9     private open: boolean
10    private url: string
11    private reconnectTimeout: number
12
13 /**
14  * creates new instance and sets default values
15  * @param url url to which the websocket-client should connect
16 */
17 constructor(url: string) {
18     super()
19     this.webSocket = new WebSocket(url)
20     this.registerListener()
21     this.open = false
22     this.url = url
23     this.reconnectTimeout = 250
24 }
25
26 private registerListener() {
27     // Run when there is an incoming message
28     this.webSocket.onmessage = (event) => {
29         // Parse message after custom scheme
30         const msg: string = event.data
31         const command: string = msg.substr(0, msg.indexOf(' '))
32         const data: string = msg.replace(command + ' ', '')
33         // This class is a child of the eventemitter, so it can emit events, to which
34         other parts in the code can subscribe to
35         this.emit(command, data)
36     }
37     // Run when the connection is sucessfully established
38     this.webSocket.onopen = () => {
39         // Set websocket open and reset reconnecttimer
40         this.open = true
41         this.reconnectTimeout = 250
42     }
43     // Run when the connection gets closed
44     this.webSocket.onclose = () => {
45         // Set open to false so there are now trys to send something while closed
46         // connection and start trying to reconnect to the server
47         this.open = false
48         console.log('Websocket closed, retrying connection')
49         setTimeout(() => {
50             this.webSocket = new WebSocket(this.url)
51             console.log('Reconnecting to websocket')
52             this.registerListener()
53             this.reconnectTimeout = this.reconnectTimeout + 250
54         }, Math.min(10000, this.reconnectTimeout)) // Interval gets greater every try,
55         // max out at 10 seconds
56     }
57 }
58
59 // Public method used for sending data to server
60 public send(invocation: string, data: string) {
```

```
58 // If the connection is not open try again in 1 second, else send
59 if (!this.open) {
60   setTimeout(() => this.trySend(invocation, data), 1000)
61 } else {
62   this.trySend(invocation, data)
63 }
64 }
65
66 // Internal method checking if sending is possible
67 private trySend(invocation: string, data: string) {
68   // If websocket is closed try the send method again and retry in 1 sec, else
69   // finally send to server
70   if (!this.open) {
71     this.send(invocation, data)
72   } else {
73     this.websocket.send(invocation + ' ' + data)
74   }
75 }
76
77 export default (_: any, inject: any) => {
78   // Create new instance with the url in /assets/config.json
79   const wrapper = new WebsocketWrapper(config.wsURL)
80   // Inject the instance to the app, now everywhere available with this.$websocket
81   inject('websocket', wrapper)
82 }
83 }
```

plugins/websocket.ts

```
1 <template>
2   <v-list-item
3     :key="title"
4     :class="{ 'blue--text': current === index }"
5     style="pointer-events: none"
6     :style="getVars"
7   >
8     <v-list-item-content>
9       <v-list-item-title
10        style="color: white"
11        v-text="title"
12      ></v-list-item-title>
13    </v-list-item-content>
14    <div class="action">
15      <v-list-item-action-text v-text="time"></v-list-item-action-text>
16
17      <div>
18        <div v-if="this.$store.state.privileged" style="display: unset">
19          <v-icon
20            v-if="current !== index"
21            color="grey lighten-1"
22            style="pointer-events: auto"
23            @click="askConfirmation"
24            >mdi-arrow-right-drop-circle-outline</v-icon>
25        >
26        <v-icon v-else color="red darken-3"
27          >mdi-arrow-right-drop-circle</v-icon>
28      >
29    </div>
30    <!-- IOS/Safari still doesnt support notifications after 5 years!!!! Roadmap
says will maybe be available in 2021 -->
31    <!-- <v-icon
32      v-if="!notification"
33      color="grey lighten-1"
34      style="pointer-events: auto"
35      @click="toggleNotification"
36      >
37        mdi-alarm-off
38      </v-icon>
39
40    <v-icon
41      v-else
42      color="yellow darken-3"
43      style="pointer-events: auto"
44      @click="toggleNotification"
45      >
46        mdi-alarm
47      </v-icon> -->
48    </div>
49  </div>
50 </v-list-item>
51 </template>
52
53 <script>
54 import { mapState } from 'vuex'
55
56 export default {
57   props: {
58     title: {
59       type: String,
```

```
60      required: true,
61    },
62    time: {
63      type: String,
64      required: true,
65    },
66    index: {
67      type: Number,
68      required: true,
69    },
70  },
71  data() {
72    return {
73      notification: false,
74    }
75  },
76  computed: {
77    getVars() {
78      return {
79        '--conf-opacity': this.current === this.index ? '0.24' : '0',
80        '--bgColor':
81          this.current === this.index ? 'currentColor' : 'transparent',
82      },
83    },
84    ...mapState(['current']),
85  },
86  mounted() {
87    this.unsubscribe = this.$store.subscribe((mutation, state) => {
88      if (mutation.type === 'setCurrent') {
89        if (state.current === this.index) {
90          setTimeout(() => {
91            this.$el.classList.add('v-list-item--active')
92          }, 300)
93        }
94      }
95    })
96  },
97  beforeDestroy() {
98    this.unsubscribe()
99  },
100 methods: {
101   /* toggleNotification() {
102     this.notification = !this.notification
103     setTimeout(() => {
104       this.$el.classList.add('v-list-item--active')
105     }, 300)
106   }*/
107   // Method triggered when you want to change the current active element
108   askConfirmation() {
109     if (this.current !== this.index) {
110       // Send an event to the index page telling it to show the confirmation dialog
111       this.$root.$emit('askConfirmation', this.index)
112     }
113     // Readd active class which sometimes gets removed when clicking
114     buttons(feature?!? of the ui-framework)
115     setTimeout(() => {
116       this.$el.classList.add('v-list-item--active')
117     }, 300)
118   },
119 }
```

```
119 }
120 </script>
121
122 <style scoped>
123 .v-list-item--active::before {
124   opacity: var(--conf-opacity) !important;
125   background-color: var(--bgColor) !important;
126 }
127 .action {
128   display: inline-flex;
129   min-width: 24px;
130   margin: 12px 0;
131   align-items: flex-end;
132   align-self: stretch;
133   justify-content: space-between;
134   white-space: nowrap;
135   flex-direction: column;
136 }
137 </style>
138
```

## components/ConferenceItem.vue

```
1 // Current state/context of the application
2 export const state = () => ({
3   privileged: false,
4   // notificationOn: [],
5   code: '',
6   items: [],
7   delay: 0,
8   current: 0,
9 })
10
11 // Setters for the state/context. Called with this.$store.commit("setterName",
12 // params...)
12 export const mutations = {
13   setPrivileged(state: { privileged: boolean }, newValue: boolean) {
14     state.privileged = newValue
15   },
16   /* addNotification(state: { notificationOn: number[] }, index: number) {
17     state.notificationOn.push(index)
18   }, */
19   setCode(state: { code: string }, code: string) {
20     state.code = code
21   },
22   setItems(state: { items: string[] }, items: string[]) {
23     state.items = items
24   },
25   setDelay(state: { delay: number }, delay: number) {
26     state.delay = delay
27   },
28   setCurrent(state: { current: number }, current: number) {
29     state.current = current
30   },
31 }
32
```

## store/index.ts