**DIPLOMARBEIT**

Gesamtprojekt

**Webbasiertes Managementsystem für Klassenkonferenzen**

Betreuer/Betreuerin:

**Spezifische Themenstellung Christian Höller**

Christian Höller 5AHWII Prof. MSc Johannes Egger

**Spezifische Themenstellung Elias Werth**

Elias Werth 5AHWII Prof. MSc Johannes Egger

Ausgeführt im Schuljahr 2019/20 von: Christian Höller und Elias Werth

Abgabevermerk:

Datum: übernommen von:

**DIPLOMARBEIT**

**DOKUMENTATION**

|  |  |
| --- | --- |
| Namen der  Verfasser/innen | Christian Höller  Elias Werth |
| Jahrgang  Schuljahr | 5. Jahrgang  2019/20 |
| Thema der Diplomarbeit | Webbasiertes Managementsystem für Klassenkonferenzen |
| Kooperationspartner | HTL Vöcklabruck |

|  |  |
| --- | --- |
| Aufgabenstellung | Die Schule verfügt über eine Anwendung zur Steuerung von Klassenkonferenzen. Dieses System ist jedoch veraltet und daher soll ein neues erstellt werden. Gefordert sind neue Features und ein entsprechend modernes Design. |

|  |  |
| --- | --- |
| Realisierung | Um den Anforderungen der Schule umzusetzen, wurde die Seite neu aufgebaut und mit einem entsprechend modernen Design versehen.  Die Seite wurde in C# mit HTML und Bootstrap realisiert. Dies ermöglicht eine Übersichtliche Darstellung und einen reibungslosen Ablauf der Konferenz. |

|  |  |
| --- | --- |
| Ergebnisse | Durch den übersichtlichen Aufbau ist es den Lehrern möglich, die Seite leichter zu bedienen. Mithilfe der Nutzerauthentifizierung kann es zu keinem Missbrauch der Rechte kommen. |

|  |  |
| --- | --- |
| Typische Grafik, Foto etc.  (mit Erläuterung) | Moderatoransicht  Nutzeransicht  Microsoft Weiterleitung |

|  |  |
| --- | --- |
| Teilnahme an Wettbewerben, Auszeichnungen | nicht |

|  |  |
| --- | --- |
| Möglichkeiten der Einsichtnahme in die Arbeit | Schulbibliothek |

|  |  |  |
| --- | --- | --- |
| Approbation  (Datum / Unterschrift) | Prüfer/Prüferin | Direktor/Direktorin  Abteilungsvorstand/Abteilungsvorständin |

**DIPLOMA THESIS**

**Documentation**

|  |  |
| --- | --- |
| Author(s) | Christian Höller  Elias Werth |
| Form  Academic year | 5th Form  2019/2020 |
| Topic | Web-based management system for class conferences |
| Co-operation partners | HTL Vöcklabruck |

|  |  |
| --- | --- |
| Assignment of tasks | The school has an application for controlling class conferences. However, this system is out of date and a new one should be created. New features and a correspondingly modern design are required. |

|  |  |
| --- | --- |
| Realisation | In order to implement the requirements of the school, the page was rebuilt and given a modern design.  The page was realized in C # with HTML and Bootstrap. This enables a clear presentation and a smooth running of the conference. |

|  |  |
| --- | --- |
| Results | The clear structure enables teachers to use the site more easily. With user authentication, rights cannot be misused. |

|  |  |
| --- | --- |
| Illustrative graph, photo  (incl. explanation) | Siehe oben |

|  |  |
| --- | --- |
| Participation in competitions  Awards |  |

|  |  |
| --- | --- |
| Accessibility of  Diploma Thesis | School library |

|  |  |  |
| --- | --- | --- |
| Approval  (Date / Signature) | Examiner | Head of College / Department |

**EIDESSTATTLICHE ERKLÄRUNG**

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Vöcklabruck, am TT.MM.JJJJ Verfasser / innen:

Christian Höller  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Elias Werth  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Inhaltsverzeichnis

1 Allgemeines 1

1.1 Vorstellung des Projektteams 1

1.1.1 Diplomanden 1

1.1.2 Vorstellung der Betreuer 4

1.2 Aufgabenstellung der Diplomarbeit 5

1.3 Ausgangssituation 5

1.4 Individuelle Zielsetzungen 6

1.4.1 Zielsetzung Höller 6

1.4.2 Zielsetzung Werth 9

2 Verwendete Programme und Tools 11

2.1 Visual Studio 11

2.2 DB Browser for SQLite 11

2.3 Git 11

2.4 GitHub 11

2.5 JSON-Formatierung 12

2.6 NuGet-Pakete 13

2.6.1 SQLite 13

2.6.2 ASP.net SignalR 13

2.6.3 Newtonsoft.JSON 13

2.6.4 AzureAD 14

2.6.5 Bootstrap 14

3 Sprachen 15

3.1 C# 15

3.2 HTML 15

3.3 CSS 16

3.4 jQuery 16

3.5 JavaScript 16

3.6 MySQL 17

4 Realisierung 18

4.1 Datenhandhabung 18

4.1.1 JSON 18

4.1.2 SQLite 20

4.2 SignalR 22

4.3 Modellklassen 25

4.4 Klassen 26

4.4.1 Allgemeine Informationen 26

4.4.2 Datenbankverbindung 29

4.5 Ansicht-Moderator 31

4.5.1 Raumauswahl 31

4.5.2 Steuerung der Konferenz 34

4.6 Ansicht-Nutzer 45

4.6.1 Laden der Räume 45

4.6.2 Laden der Informationen 48

4.7 Authentifizierung mit Microsoft Azure AD 51

4.7.1 Registrierung im Azure-Portal 51

4.7.2 Konfiguration der Applikation 52

4.7.3 Implementierung des Logins 53

5 Dokumentation der individuellen Themenstellung 1 57

6 Dokumentation der individuellen Themenstellung 2 58

7 Zusammenfassung 60

8 Literatur- und Quellverzeichnis 61

9 Abbildungsverzeichnis 61

ANHANG A - Terminplan 62

ANHANG B - Begleitprotokolle 63

ANHANG C - Tätigkeitsprotokolle 64

ANHANG D - Dokumentationen 65

ANHANG E - Layouts und Verzeichnisse in Microsoft WORD 66

# Allgemeines

## Vorstellung des Projektteams

### Diplomanden

#### Ein Bild, das Person, Schlips, Mann, Wand enthält. Automatisch generierte BeschreibungChristian Höller

Klasse: 5AHWII

Geburtsdatum: 03.10.2000

Wohnort: Attnang-Puchheim

E-Mail: christian.hoeller@htlvb.at

Ausbildungszweig: Betriebsinformatik

**Beruflicher Werdegang**

**August 2019 - Verfahrenstechnik**

Lenzing AG

Arbeit im Schichtbetrieb zum eigenständigen Messen von Lauge- und Viskosewerten. Diese Werte wurden anschließend analysiert und auf Richtigkeit überprüft.

**August 2018 – Schlosserei**

Lenzing AG

Hilfskraft in der Schlosserei für Reparaturarbeiten, Assistieren bei Schweißarbeiten und Transport.

**August 2017 – Büroarbeit**

MIBA Gleitlager Austria GmbH

Erstellen von Arbeitsmappen über Produktionsmaschinen; Überarbeiten von bereits vorhandenen Dokumenten.

**Ausbildung**

**Seit 2016 – HTL Vöcklabruck – Wirtschaftsingenieure: Betriebsinformatik**

Maturatermin Juni 2020

**2014 – 2016 – HTL Wels – Mechatronik**

**2010-2014 – Bundesrealgymnasium Schloss Wagrain**

Deutsch, Mathematik, Englisch, Religion

#### Elias Werth

Klasse: 5AHWII

Geburtsdatum: 05.05.2001

Wohnort: Edt bei Lambach

E-Mail: elias.werth@htlvb.at

Ausbildungszweig: Betriebsinformatik

**Beruflicher Werdegang**

**August 2019 – Netzwerktechnik**

BRP-Rotax GmbH & Co KG

Als Hilfskraft für die Netzwerktechniker der Firma über ein Monitoringsystem IP-Adressen vergeben und Gruppen der Device verwalten; Aufbauen von Workstations in der Produktion sowie in anderen Abteilungen.

**August 2019 – Informationstechnologie**

eww Gruppe

Instandhaltung von alten Dokumenten durch Scannen und Abspeichern in dem jeweiligen Ordner; Unterstützung der Außendienstmitarbeiter durch das Festhalten der Maße auf dem jeweiligen Plan.

**Ausbildung**

**Seit 2015 – HTL Vöcklabruck – Wirtschaftsingenieure: Betriebsinformatik**

Maturatermin Juni 2020

**2011-2015 – Sporthauptschule Lambach**

Deutsch, Mathematik, Englisch, Religion

### 1.1.2 Vorstellung der Betreuer

**Prof. MSc BSc Johannes Egger**

Wir möchten uns sehr herzlich bei Herrn Prof. Egger für die gute Zusammenarbeit und Unterstützung bedanken.  
Er hat uns stets sehr gut unterstützt, und bei schwierigen Arbeitsschritten geholfen, wofür wir uns zusätzlich herzlich bedanken möchten!

## Aufgabenstellung der Diplomarbeit

Aufgabe der Diplomarbeit war es, eine neue Website zu erstellen, welche eine Nutzerauthentifizierung über Microsoft, eine einfache Steuerung der Konferenz und ein modernes Design aufweist.

## Ausgangssituation

Bei der alten Anwendung existiert nur eine Ansicht, wodurch es jedem Nutzer ermöglicht wird, die Konferenz zu steuern. Dieses Problem sollte durch Authentifizierung von Nutzern und dazugehöriger Rechteverteilung gelöst werden. Folglich ergeben sich eine Nutzer-Ansicht und eine Administrator-Ansicht.

Durch ein modernes und ansprechendes Design sollte eine übersichtliche Website erstellt werden.

## Individuelle Zielsetzungen

### Zielsetzung Höller

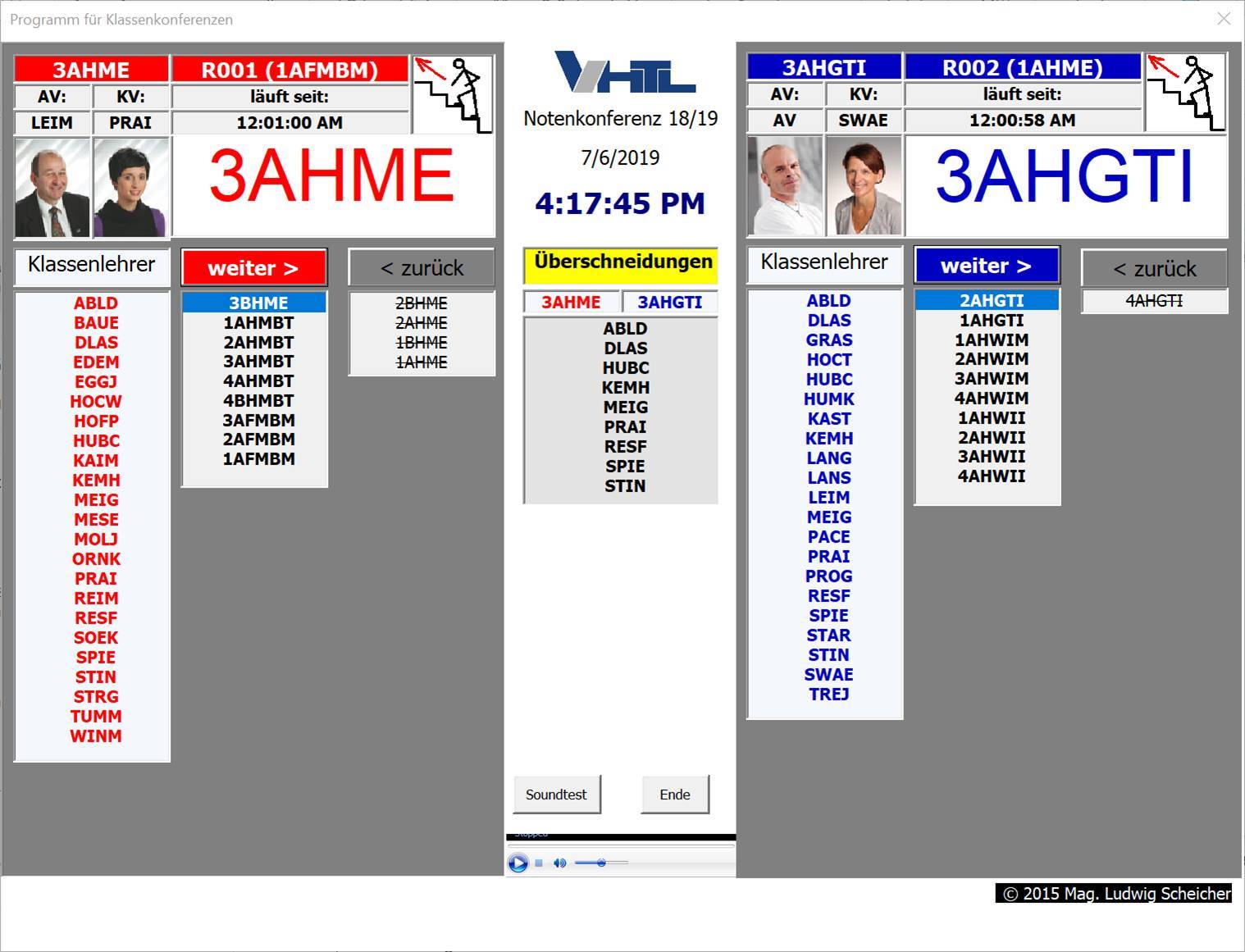
#### Datenbankstruktur

Bevor die Arbeit an den Ansichten beginnen konnte, musste eine geeignete Datenbankstruktur erstellt werden. Diese war essenziell für einen reibungslosen und übersichtlichen Ablauf.

In der Datenbank werden folgende Werte gespeichert werden:

* Status der Konferenz
* Raum, in welchem die Klasse besprochen wird
* Besprechungsreihenfolge der Klassen
* Besprechungsstatus der Klassen
* Start- und Endzeit der Besprechung je Klasse

#### Moderator-Ansicht



Eines der wichtigsten Anliegen der Schule war es, die Darstellung der Daten für die Besprechung auf je eine eigene Seite zu verlagern. Somit gibt es für jede Besprechung in einem Raum auch eine eigene Seite. Wie auf der Abbildung zu sehen, werden beide Besprechungen der Konferenz auf einer Seite angezeigt.

Im Gegensatz zu weiter und zurück, soll die Konferenz nur mehr mit einem zentralen Button gesteuert werden. Sobald der Moderator die Konferenz startet, wird die erste Klasse besprochen. Anschließend wird je Knopfdruck automatisch die nächste Klasse und deren zugehörige Daten geladen.

Durch die Verlagerung der Besprechungen auf je eine eigene Seite war es eine große Hürde den Datenaustausch zwischen den Clients zu gewährleisten. Da sich die Überschneidungen der Lehrer ändern sobald eine neue Klasse geladen wird, müssen diese Daten an den anderen Client in der zweiten Klasse übermittelt werden.

#### Nutzer-Ansicht

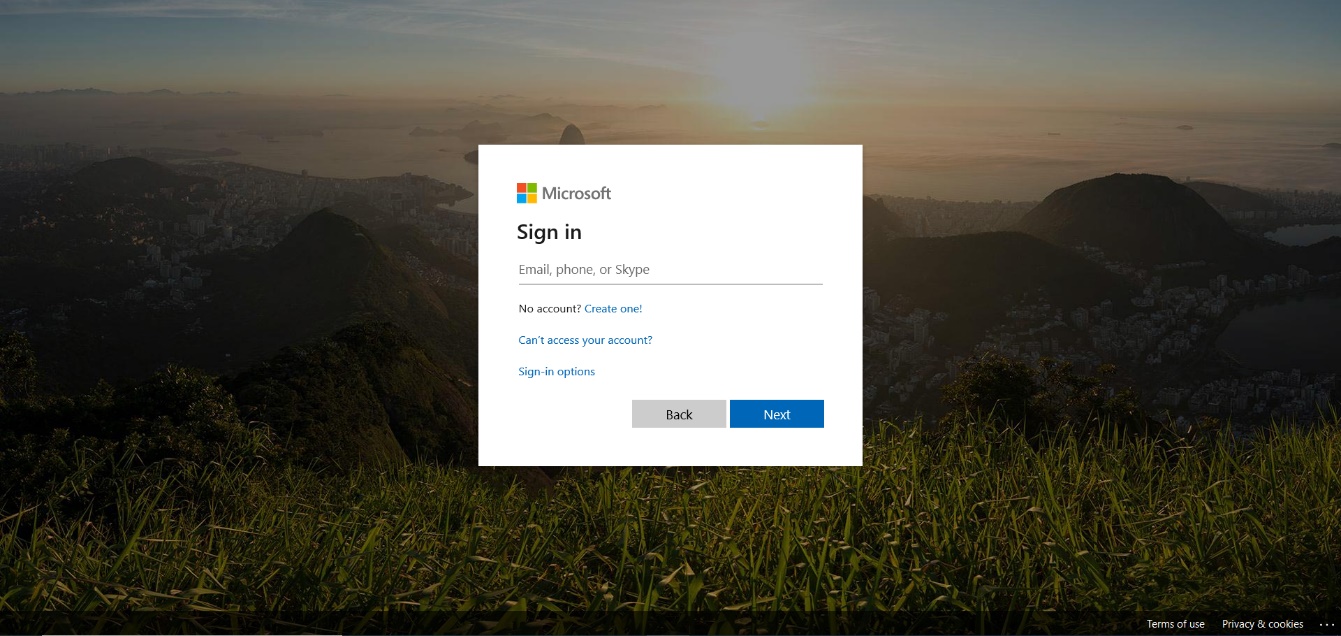
Diese Ansicht existierte in der alten Anwendung noch nicht und sollte die Einsicht in die Konferenz für Lehrer ermöglichen. Der Nutzer sollte eine Ansicht beider Klassen auf einer Seite haben. Die abgeschlossenen und nächsten Klassen je Raum werden ebenfalls angezeigt.

Grundvoraussetzung für die Ansicht war das automatische Aktualisieren der Daten. Somit werden die Daten für die besprochene Klasse stets aktuell auf der Seite dargestellt.

### Zielsetzung Werth

#### Authentifizierung mittels Azure AD

Im Gegensatz zum alten Managementsystem soll das neue Programm die Möglichkeit besitzen, sich anzumelden. Mithilfe von Microsoft soll dies realisiert werden. Der Grund, warum hierfür ein externer Login implementiert wird, ist, dass bereits jede einzelne Lehrkraft über einen Office-365-Account (htlvb-Email) verfügt.

Das Prinzip hinter der Authentifizierung ist also folgendes: Ohne Anmeldung im neuen Managementsystem darf es den Lehrerinnen und Lehrern nicht möglich sein, eine Einsicht auf den Status der Konferenz zu bekommen bzw. den Ablauf steuern zu können. Die Lehrkraft wird also mehr oder weniger gezwungen sich mit ihrem MS-Konto anzumelden. Für die Anmeldung selbst wird man auf die typische Microsoft-Anmeldeseite weitergeleitet und nachdem man sich erfolgreich eingeloggt hat, wird man wieder auf unsere Seite zurück auf das Managementsystem gelenkt.

<https://login.microsoftonline.com/>

Der wohl größte Vorteil, der sich durch die Verwendung des Office-365-Konto ergibt, ist der, dass man nun Rollen vergeben kann. Durch eine Überprüfung der Rechte bzw. des Titels nach dem Login, lässt sich leicht feststellen, ob der Benutzer nun zur Moderator-Ansicht oder zur Nutzer-Ansicht weitergeleitet werden soll. Somit wird es dem User nochmals um einiges erleichtert, die Konferenz problemlos zu steuern.

#### Progressive Web App

Der Grund warum sich eine PWA hier sehr gut anbietet ist einfach: Für Lehrkräfte ist es um einiges leichter und angenehmer, wenn sie den Status der Konferenz mühelos über ihr Mobiltelefon abrufen können. Eine PWA bietet zu dem noch viel mehr Vorteile. Grundsätzlich vereint eine progressive Web App die Eigenschaften mobiler Websites und nativen Apps. Somit passt sich die Website an das Gerät an, über das sie aufgerufen wird. Bugs der Webpage können somit vorgebeugt werden.

Das bedeutet für das Managementsystem der Klassenkonferenzen, dass das Design entsprechend angepasst werden muss. Dem Benutzer soll es auch möglich sein, die Webseite wie eine native App auf dem Desktop des Smartphones zu speichern.

#### Ausrufen von Lehrern

Ein Moderator (Abteilungsvorstand) soll die Möglichkeit haben, nach Bedarf fehlende Lehrkräfte ausrufen zu können. Dieser hat in der Moderator-Ansicht eine Übersicht über alle Lehrer, die bei der aktuellen Klasse anwesend sein sollten. Wenn der Moderator nach der Anwesenheitskontrolle der Lehrkräfte feststellt, dass ein oder mehrere Personen fehlen, können diese mit einem einfachen Knopfdruck ausgerufen werden.

Bei der Lehrkraft, die ausgerufen wird, ist der Aufruf an sich eine einfache Meldung auf der Website. Dabei wird die Person höflich darauf hingewiesen, dass sie in dem Raum erwartet wird, in dem sie sein solle. Zudem kann jeder normale Benutzer für sich selbst entscheiden, ob er Benachrichtigungen (Ausrufe) erhalten möchte oder nicht.

#### Text-to-Speech

Text-to-Speech wird im Managementsystem als Feature implementiert, welches beim Ausrufen von Lehrkräften eine Rolle spielen soll. Sobald die ausgewählte Person ausgerufen wird, soll mittels JavaScript, die Benachrichtigung automatisch vorgelesen werden. Ziel ist dabei die Sicherstellung, dass die Lehrkraft den Ausruf auch tatsächlich wahrnimmt.

#### Design

Das Design soll nicht nur ansprechend und modern sein. Die Lehrkraft soll sich vor allem in der Umgebung wohl fühlen. Die Steuerung der Konferenzen muss sich leicht anfühlen und die Daten in Nutzer-Ansicht sollen angemessen dargestellt werden. Als Hauptfarbe der Website wird das Blau der HTL gewählt.

# Verwendete Programme und Tools

## Bildergebnis für visual studio logoVisual Studio

Visual Studio ist eine von dem Unternehmen Microsoft angebotene integrierte Entwicklungsumgebung für verschiedene Hochsprachen.

<https://de.wikipedia.org/wiki/Visual_Studio>

Bild: <https://de.m.wikipedia.org/wiki/Datei:Visual_Studio_2013_Logo.svg>

## DB Browser for SQLite

DB Browser ist ein visuelles Programm, welches das Erstellen und Bearbeiten von SQLite-Datenbankdateien ermöglicht.

<https://sqlitebrowser.org/>

Bild: <https://twitter.com/sqlitebrowser/status/967968416236785665>

## Git

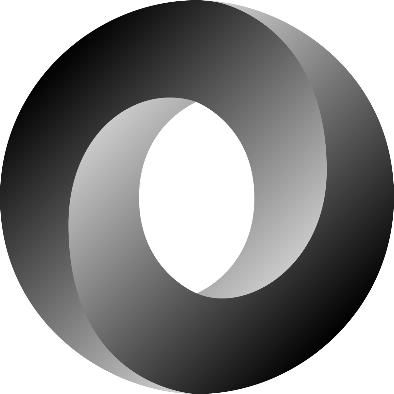
Git ist ein kostenloses Open-Source Versionskontrollsystem, welches entwickelt wurde, um das gleichzeitige Arbeiten an einem Projekt, unabhängig vom Standort, zu ermöglichen.

Mithilfe von Git können Versionen eines Projekts gespeichert werden und bei Bedarf auch ältere Versionen abgerufen werden. Der Zugriff auf Git erfolgt durch die

## GitHub

GitHub beinhaltet das Projekt und wurde als Online-Plattform für unsere Arbeit gewählt.

## JSON-Formatierung

JSON (JavaScript Object Notation) ist ein sprachenübergreifendes Textformat, welches den Austausch von strukturierten Daten ermöglicht. JSON basiert auf der JavaScript-Sprache und ist sehr einfach zu lesen.

<https://www.cloudcomputing-insider.de/was-ist-json-a-704909/>

Bild: <https://de.wikipedia.org/wiki/JavaScript_Object_Notation>

## NuGet-Pakete

### SQLite

SQLite ist ein Datenbanksystem, welches keinen eigenen Server benötigt, sondern lokal gespeichert wird. Das System wird vorwiegend für geringe Datenmengen verwendet und unterstützt einen Großteil der im SQL-92-Standard festgelegten SQL-Sprachbefehle.

<https://de.wikipedia.org/wiki/SQLite>

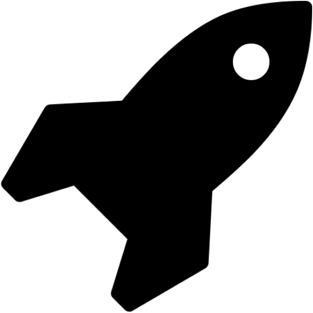
### ASP.net SignalR

Ist eine Bibliothek für ASP.NET-Entwickler, die das Hinzufügen von Echtzeit Webfunktionen zu Anwendungen vereinfacht. Servercodeinhalte werden direkt an verbundene Clients übertragen, ohne dass diese die Daten erst anfordern müssen. Sie ermöglicht also das asynchrone Senden von Nachrichten an Clients.

Bild: <https://blog.dotnetstudio.nl/connect-azure-ad-authenticated-signalr-hub-from-spfx/>

<https://docs.microsoft.com/de-de/aspnet/signalr/overview/getting-started/introduction-to-signalr>

### Newtonsoft.JSON

Ist eine Softwarebibliothek, welche das einfache und schnelle Serialiseren und Deserialiseren von Objekten ermöglicht. Mit über 100 Millionen Downloads ist diese Bibliothek die Nummer eins bei NuGet.

<https://www.newtonsoft.com/json>

### AzureAD

Die ASP.NET Core Azure Active Directory-Integration bietet Komponenten für die einfache Integration der Azure Active Directory-Authentifizierung in einer ASP.NET Core-Anwendung.

<https://www.nuget.org/packages/Microsoft.AspNetCore.Authentication.AzureAD.UI/>

Bild: <https://www.ceteris.ag/innovative-bi/technologien/microsoft-azure/>

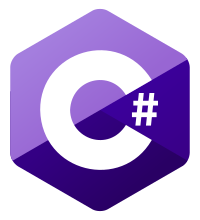
### Bootstrap

Bootstrap ist ein freies Frontend-CSS-Framework. Es enthält auf HTML und CSS basierende Gestaltungsvorlagen für Typografie, Formulare, Buttons, Tabellen, Grid-Systeme, Navigations- und andere Oberflächengestaltungselemente sowie zusätzliche, optionale JavaScript-Erweiterungen.

<https://de.wikipedia.org/wiki/Bootstrap_(Framework)>

# Sprachen

## C#

C# (ausgesprochen "See Sharp") ist eine einfache, moderne, objektorientierte und typsichere Programmiersprache. C# hat seine Wurzeln in der C-Sprachfamilie und wird C-, C++ -, Java- und JavaScript-Programmierern sofort bekannt sein.

<https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/>

Bild: <https://de.wikipedia.org/wiki/C-Sharp>s

## HTML

HTML (Hypertext Markup Language) ist eine textbasierte Auszeichnungssprache und sorgt für die Struktur und den eigentlichen Inhalt von nahezu allen Webseiten.

Für diese Arbeit wird die zurzeit aktuellste Version, HTML5, verwendet. Diese wird aktuell von allen modernen Browsern unterstützt.

<https://de.wikipedia.org/wiki/HTML5>

<https://de.wikipedia.org/wiki/Hypertext_Markup_Language>

<https://www.w3schools.com/html/html5_browsers.asp>

## CSS

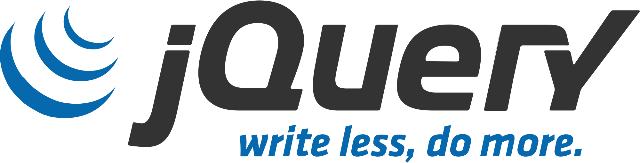
CSS (Cascading Style Sheets) ist eine Stylesheet-Sprache, die zusammen mit HTML in fast jeder Website zum Einsatz kommt. CSS kümmert sich um die Formatierung der Seite.

So können beispielsweise Schriftarten, Hintergrundfarben, Größen und sogar Animationen festgelegt werden.

<https://de.wikipedia.org/wiki/Cascading_Style_Sheets>

<https://www.w3schools.com/cssref/css3_browsersupport.asp>

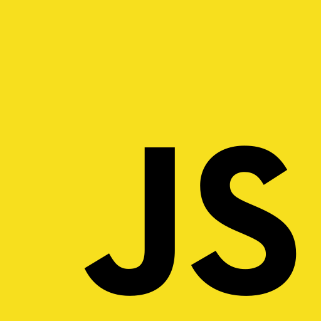
## jQuery

jQuery ist eine schnelle, kleine und funktionsreiche JavaScript-Bibliothek. Mit einer benutzerfreundlichen API, die in einer Vielzahl von Browsern funktioniert, werden Dinge wie das Durchlaufen und Bearbeiten von HTML-Dokumenten, die Ereignisbehandlung, die Animation und Ajax erheblich vereinfacht. Mit einer Kombination aus Vielseitigkeit und Erweiterbarkeit hat jQuery die Art und Weise verändert, wie Millionen von Menschen JavaScript schreiben.

<https://jquery.com/>

Bild: <https://de.wikipedia.org/wiki/Datei:JQuery-Logo.svg>

## JavaScript

JavaScript (kurz JS) ist eine Skriptsprache, die für dynamisches HTML in Webbrowsern entwickelt wurde, um Benutzerinteraktionen auszuwerten, Inhalte zu verändern, nachzuladen oder zu generieren und somit die Möglichkeiten von CSS und HTML zu erweitern.

<https://de.wikipedia.org/wiki/JavaScript>

Bild: <https://de.wikipedia.org/wiki/Datei:Unofficial_JavaScript_logo_2.svg>

## MySQL

MySQL ist ein frei verfügbares Open Source-Relational-Database-Management-System (RDBMS), das Structured Query Language (SQL) verwendet. MySQL ist eines der weltweit verbreitetsten relationalen Datenbankverwaltungssysteme.

<https://www.siteground.com/tutorials/php-mysql/mysql/>

<https://de.wikipedia.org/wiki/MySQL>

# Realisierung

## Datenhandhabung

Zum Speichern der Konferenzdaten wurden zwei Technologien herangezogen:

* JSON- zum Speichern der allgemeinen Daten (Lehrer, Klassen, Räume, …)
* SQLite- zum Speichern des Konferenzstatus

### JSON

Die Daten für den Ablauf der Konferenz werden in einer JSON-Datei gespeichert. Dazu wurde in dem Ordner „wwwroot“ ein neuer Ordner mit dem Namen „json“ erstellt. Darin befindet sich die JSON-Datei „conference-info.json“.

Die Datei besteht aus drei Objekten

* Lehrer (teachers)
* Klassen (classes)
* Reihenfolge (order)

Lehrer

Ein Lehrer besteht aus einer ID und einem Namen. Als ID wird die E-Mail gespeichert und der Name besteht aus Nach- und Vornamen.

{

"id": "grug@htlvb.at",

"name": "Gruber Gerald"

},

Klassen

Es werden Klassenname, Abteilungsvorstand, Klassenvorstand und die Lehrer, welche diese Klasse unterrichten, gespeichert.

Die Lehrer werden immer mit der ID abgespeichert, sodass es zu keinen Verwechslungen kommen kann.

{

"className": "5AHWII",

"headOfDepartment": "baue@htlvb.at",

"formTeacher": "fram@htlvb.at",

"teachers": [

{ "id": "hoec@htlvb.at" },

{ "id": "were@htlvb.at" },

{ "id": "molj@htlvb.at" },

{ "id": "grug@htlvb.at" },

{ "id": "edem@htlvb.at" }

]

},

Reihenfolge

Die Reihenfolge besteht aus einem Raum und den Klassen. Das heißt, dass die Klassen, welche angeführt sind, in dem angegebenen Raum besprochen werden.

Die Reihenfolge wie die Klassen besprochen werden ist so, wie sie in der Datei stehen.

{

"room": "R001 (2AFMBM)",

"classes": ["2AHWII", "3AHWII", "4AHWII"]

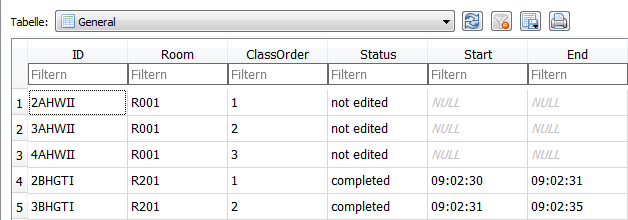
},

### SQLite

Die Daten für den Konferenzstatus und der besprochenen Klassen werden in der Datenbankdatei „database\_conference.db“ gespeichert. Diese befindet sich im Projektordner „wwwroot/sqlite“.

**Besprochene Klassen**

Zum Abspeichern des Status, sowie der Start- und Endzeit der einzelnen Klassen wurde die Tabelle „General“ erstellt.

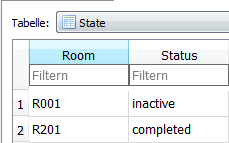


**Status der Konferenz**

In der Tabelle „State“ wird der aktuelle Konferenzstatus für die einzelnen Besprechungsräume gespeichert.

Es wird zwischen drei Werten für den Status der Konferenz unterschieden:

* Inactive- Die Konferenz wurde noch nicht gestartet
* Running- Die Konferenz läuft gerade
* Completed- Die Konferenz ist abgeschlossen



## SignalR

Da zwei Klassen gleichzeitig besprochen werden, sind auch zwei Clients gleichzeitig auf der Moderator-Seite angemeldet. Um den Austausch von Informationen zwischen den Clients zu ermöglichen, kommt hier erstmals die Softwarebibliothek „SignalR“ zum Einsatz.

Die Bibliothek ermöglicht es nun Informationen, wie die Überschneidungen, stets aktuell an den anderen Client zu übergeben. Die Moderator-Seite ist so aufgebaut, dass alle Daten mithilfe von SignalR erneuert werden.

#### Hinzufügen von SignalR

Solution Explorer 🡪 Rechtsklick auf Projekt 🡪 Add 🡪 Client-Side Library 🡪 Install



Nachdem die Bibliothek installiert wurde, musste eine neue Klasse erstellt werden.

Diese Klasse wurde in dem Ordner „Hubs“ erstellt und trägt den Namen „MainHub“. Hier werden alle asynchronen Methoden, welche mit JavaScript kommunizieren, implementiert.

namespace Managementsystem\_Classconferences.Hubs

{

public class MainHub : Hub

{

Einstellungen in der „Startup.cs“ Datei

using Managementsystem\_Classconferences.Hubs;

In der Methode „ConfigureServices“ muss SignalR noch hinzugefügt warden. Dies geschieht mit „services.AddSignalR();“ am Ender der Methode.

Zum Schluss muss noch die Route zur Hub-Klasse festgelegt werden. Dazu wird in der Methode „Configure“ die Route folgend definiert:

app.UseSignalR(routes =>

{

routes.MapHub<MainHub>("/mainHub");

});

Nachdem die Hub-Klasse konfiguriert wurde, muss eine JavaScript-Datei erstellt werden.

Diese JavaScript-Datei verwaltet die Kommunikation zum Hub. Die Methoden, welche zur Verbindung notwendig sind, finden sich unter:

„<https://docs.microsoft.com/en-us/aspnet/core/tutorials/signalr?view=aspnetcore-3.1&tabs=visual-studio>“

Es wurden nun zwei JavaScript -Dateien erstellt:

* Moderator\_view.js
* User\_view.js

Damit die HTML-Seite mit JavaScript kommunizieren und so SignalR verwenden kann, müssen die beiden Dateien jeweils in HTML verlinkt werden.

<script src="~/lib/signalr/dist/browser/signalr.js"></script>

<script src="~/js/moderator\_view.js"></script>

<script src="~/lib/signalr/dist/browser/signalr.js"></script>

<script src="~/js/user\_view.js"></script>

## Modellklassen

Um die erhaltenen JSON-Dateien in ein Objekt umzuwandeln, wurden sogenannte Modellklassen erstellt.

Wenn JSON Dateien erhalten werden, können diese sofort mithilfe von Newtonsoft in Objekte umgewandelt werden. Um dies zu ermöglichen müssen Klassen erstellt werden, in welchen die Namen der Eigenschaften mit jenen aus der JSON-Datei übereinstimmen.

**Lehrer**

public class Teacher

{

public string ID { get; set; }

public string Name { get; set; }

}

**Klasse**

public class MyClasses

{

public string ClassName { get; set; }

public string HeadOfDepartment { get; set;}

public string FormTeacher { get; set; }

public List<Teacher> Teachers { get; set; }

}

**Reihenfolge**

public class Order

{

public string Room { get; set; }

public List<string> Classes { get; set; }

public string RoomOnly { get; set; }

}

Da der Raum in der JSON-Datei aus Raum und Klassenname besteht (z.B. R001 2AHWII) wurde zusätzlich die Eigenschaft „RoomOnly“ hinzugefügt. Für diese Eigenschaft wird anschließend nur der Raum gespeichert und nicht zusätzlich auch die Klasse.

## Klassen

### Allgemeine Informationen

Zu Beginn wurde die Klasse „General“ in dem Ordner „Classes“ erstellt. Hier werden alle allgemeinen Informationen gespeichert.

Dazu zählen:

* Tabellennamen der Datenbank
* Pfadangaben innerhalb des Projekts
* JSON-Datei in Form eines Strings

**Pfadangaben**

Um den Zugriff auf die Datenbank und die JSON-Datei zu ermöglichen, wurden die Pfade in Eigenschaften integriert.

private string path\_json;

private string path\_DB;

private string Path\_DB

{

get

{

if (path\_DB == null)

{

string root = "wwwroot";

string location = "sqlite";

string fileName = "database\_conference.db";

path\_DB = Path.Combine(

Directory.GetCurrentDirectory(),

root,

location,

fileName);

}

return path\_DB;

}

}

public string Path\_Json

{

get

{

if (path\_json == null)

{

string root = "wwwroot";

string location = "json";

string fileName = "conference-info.json";

path\_json = Path.Combine(

Directory.GetCurrentDirectory(),

root,

location,

fileName);

}

return path\_json;

}

}

**Tabellennamen**

Die Tabellennamen wurden als statische Variablen deklariert und können mittels Eigenschaften aufgerufen werden.

private static string tableNameGeneral = "General";

private static string tablenNameStateOfConference = "State";

public string Table\_General

{

get

{

return tableNameGeneral;

}

}

public string TableStateOfConference

{

get

{

return tablenNameStateOfConference;

}

}

**JSON-Datei in Form eines Strings**

Zum Lesen der einzelnen Objekte aus JSON wird die Datei in einen String umgewandelt. Hierbei ist es wichtig als Kodierung UTF-8 zu wählen, um Umlaute problemlos lesen zu können.

private string jsonstring;

public string JsonString

{

get

{

if (jsonstring == null)

{

FileStream fileStream = new FileStream(Path\_Json, FileMode.Open);

StreamReader reader = new StreamReader(fileStream, Encoding.UTF8);

using (reader)

{

jsonstring += JsonConvert.DeserializeObject(reader.ReadToEnd());

}

}

return jsonstring;

}

}

### Datenbankverbindung

Da ständig Methoden zum Lesen und Schreiben von Datenbankinformationen verwendet werden, war es von Anfang an klar, dafür eine eigene Klasse zu implementieren.

In der Klasse wurden zwei Methoden implementiert:

* Query- zum ausführen von Befehlen wie Update, Insert oder Delete
* Reader- zum Lesen von Datensätzen

Bei beiden Methoden war es wichtig, diese gegen SQL-Injections zu schützen. Daher werden die Parameter erst hier in den SQL-Befehl eingesetzt.

*SQL-Injection (dt. SQL-Einschleusung) ist das Ausnutzen einer Sicherheitslücke in Zusammenhang mit SQL-Datenbanken, die durch mangelnde Maskierung oder Überprüfung von Metazeichen in Benutzereingaben entsteht.*

<https://de.wikipedia.org/wiki/SQL-Injection>

Die Methode „Query“ liefert als Rückgabewert die Anzahl der Zeilen, welche durch den Befehl betroffen sind.

General general = new General();

public int Query(string sqlstring, params object[] parametervalues)

{

using (var connection = new SQLiteConnection($"Data Source={general.PathDB}"))

{

connection.Open();

using (var command = new SQLiteCommand(sqlstring, connection))

{

command.CommandText = sqlstring;

foreach (var param in parametervalues)

{

command.Parameters.Add(new SQLiteParameter() { Value = param });

}

return command.ExecuteNonQuery();

}

}

}

Die Methode „Reader“ liefert als Rückgabewert einen DataTable, in welchem die

Daten der gelesenen Tabelle stehen.

public DataTable Reader(string sqlstring, params object[] parametervalues)

{

DataTable dt = new DataTable();

using (var connection = new SQLiteConnection($"Data Source={general.PathDB}"))

{

connection.Open();

using (var command = new SQLiteCommand(connection))

{

command.CommandText = sqlstring;

if (parametervalues != null)

{

foreach (var param in parametervalues)

{

command.Parameters.Add(new SQLiteParameter() { Value = param });

}

}

using (SQLiteDataReader reader = command.ExecuteReader())

{

dt.Load(reader);

}

}

}

return dt;

}

## Ansicht-Moderator

### Raumauswahl

Jeder Klassenraum in der Schule hat eine Raumbezeichnung und eine Klasse, welche dauerhaft in diesem Raum ist. Die Seitenansicht enthält nun zwei Knöpfe, welche zu der jeweiligen Ansicht weiterleiten.

Die Klassenräume werden aus der JSON-Datei gelesen und zu einer Liste hinzugefügt.

public List<Order> OrderList

{

get

{

if (orderlist == null)

{

JObject jobject = JObject.Parse(general.JsonString);

JArray jOrder = (JArray)jobject["order"];

orderlist = jOrder.ToObject<List<Order>>();

foreach (var item in orderlist)

{

item.Room\_only = item.Room.Split(' ')[0];

}

}

return orderlist;

}

}

Die Objekte dieser Liste werden anschließend in HTML ausgegeben

@foreach (var order in Model.OrderList)

{

<**form** name="form" id="form" method="post" **asp-route-room**="@order.Room\_only">

<input type="submit" class="btn btn-info" value="@order.Room">

</**form**>

}

Nach drücken eines beliebigen Knopfes wird man auf die jeweilige Seite weitergeleitet.

Der Benutzer wird auf die Seite „Moderator.cshtml“ weitergeleitet. Der Raum wird als Parameter an die Seite mitgeliefert.

public IActionResult OnPost(string room)

{

HttpContext.Items["classroom"] = room;

return new RedirectToPageResult("Moderator", room);

}

### Steuerung der Konferenz

Die Seite beinhaltet folgende Informationen:

* Raum
* Klassenname
* Überschneidungen
* Lehrer
* Abgeschlossene Klassen
* Nächste Klassen

<body>

<h1 id="room"></h1>

<h1 id="classname"></h1>

<h1>Überschneidungen</h1>

<table id="intersections"></table>

<h1>Lehrer</h1>

<table id="teachers"></table>

<h1>Abgeschlossene Klassen</h1>

<table id="classes\_completed"></table>

<h1>Nächste Klassen</h1>

<table id="classes\_notedited"></table>

<input type="button" class="btn btn-info" id="sendButton" value="Besprechung" />

<script src="~/lib/signalr/dist/browser/signalr.js"></script>

<script src="~/js/moderator\_view.js"></script>

</body>

#### Raum

Wenn die Seite geladen, oder neu gestartet wird, wird in der JavaScript-Datei „moderator\_view.js“ eine Verbindung zum Hub hergestellt. Wenn diese gelingt (gekennzeichnet durch „connection.start().then…“) wird das HTML-Element mit dem Namen „room“ gesucht. In dieses Element wird anschließend der übergebene Parameter für den Raum geschrieben.

connection.start().then(function () {

document.getElementById("sendButton").disabled = false;

document.getElementById("room").innerHTML = GetCurrentRoom();

FirstStart();

}).catch(function (err) {

return console.error(err.toString());

});

Die Methode „GetCurrentRoom“ liest den Wert für den Raum aus der URL, da dieser ja von der Raumauswahl mitgegeben wurde.

function GetCurrentRoom() {

$.urlParam = function (name) {

var results = new RegExp('[\?&]' + name + '=([^&#]\*)').exec(window.location.href);

if (results == null) {

return null;

}

else {

return decodeURI(results[1]) || 0;

}

}

return $.urlParam('handler');

}

#### Laden der Informationen

Die Methode FirstStart() wird ebenfalls bei Laden der Seite aufgerufen. Der aktuelle Raum wird von der Methode „GetCurrentRoom()“ geliefert. Anschließend wird durch das „connection.invoke“ die asynchrone Hub-Methode „LoadModeratorViewInfo()“ aufgerufen und der Wert für den aktuellen Raum mitgegeben.

function FirstStart() {

var currentroom = GetCurrentRoom();

connection.invoke("LoadModeratorViewInfo", currentroom).catch(function (err) {

return console.error(err.toString());

});

event.preventDefault();

}

Diese Methode kümmert sich um das Senden der Daten an den Client. Es wird ein neues JSON Objekt erstellt, welchem je nach Status der Konferenz Werte zugewiesen werden.

Wenn die Konferenz bereits abgeschlossen ist, signalisieren dies die Rückgabedaten. Ansonsten werden alle benötigten Daten für die Ansicht übergeben.

Die Methode sendet anschließend das JSON-Objekt als string an den Client.

public async Task LoadModeratorViewInfo(string \_currentroom)

{

Currentroom = \_currentroom;

JObject obj = new JObject();

if (GetCurrentStateOfConference() != "completed")

{

obj.Add("classname", GetCurrentClassName());

obj.Add("buttontext", GetButtonText());

obj.Add(new JProperty("classes\_completed", GetClassesFromJSON("previous")));

obj.Add(new JProperty("classes\_not\_edited", GetClassesFromJSON("next")));

}

else

{

obj.Add("classname", "Alle Klassen abgeschlossen");

obj.Add("buttontext", "Konferenz abgeschlossen");

obj.Add(new JProperty("classes\_completed", GetClassesFromJSON("previous")));

obj.Add(new JProperty("classes\_not\_edited", "abgeschlossen"));

}

await Clients.Caller.SendAsync("ReveiveLoadInformation", obj.ToString());

await SendIntersections();

await SendTeachers();

}

Die abgeschlossenen Klassen („previous“) und die nächsten Klassen („next“) werden von der Methode GetClassesFromJSON geliefert.

Zuerst werden alle Klassen geladen. Falls die Konferenz bereits abgeschlossen sein sollte, werden alle Klassen zurückgegeben. Andernfalls wird der Index der aktuellen Klasse gesucht und je nach Anforderung die Klassen vorher bzw. nachher zurückgegeben. Diese Klassen in Form eines JSON-Arrays zurückgegeben

public string GetClassesFromJSON(string type)

{

var orderlist = GetOrderList();

List<string> classesInOrder = orderlist.Find(order => order.Room.Split(' ')[0] == Currentroom).Classes;

List<string> classes = new List<string>();

if (GetCurrentStateOfConference() == "completed")

{

classes = classesInOrder;

}

else

{

int index = classesInOrder.IndexOf(GetCurrentClassName());

if (type == "next")

{

classes = classesInOrder.Skip(index + 1).Take(classesInOrder.Count - index).ToList();

}

else

{

classes = classesInOrder.Take(index).ToList();

}

}

return new JArray(classes).ToString();

}

Durch die Methode GetButtonText() erhält man den Text für den Button zur Steuerung der Konferenz. Da sich der Anfangstext von „Konferenz Starten“ auf einen anderen Wert ändern kann, muss dieser den entsprechend zutreffenden Text erhalten.

private string GetButtonText()

{

switch (GetCurrentStateOfConference())

{

case "inactive":

text\_Conference\_State = "Konferenz starten";

break;

case "running":

text\_Conference\_State = "Nächste Klasse";

break;

case "completed":

text\_Conference\_State = "Konferenz abgeschlossen";

break;

}

return text\_Conference\_State;

}

Wenn JavaScript letztendlich die Daten erhält wird der JSON-String gelesen und die Daten in die HTML-Elemente geschrieben.

connection.on("ReveiveLoadInformation", function (obj) {

var obj\_parsed = JSON.parse(obj);

document.getElementById("classname").innerHTML = obj\_parsed.classname;

document.getElementById("sendButton").value = obj\_parsed.buttontext;

WriteDataInTable("classes\_completed", obj\_parsed.classes\_completed);

WriteDataInTable("classes\_notedited", obj\_parsed.classes\_not\_edited);

});

Da die Klassen in Form eines JSON-Arrays übergeben werden, wird das Array zuerst geparst und jeder wert in die richtige HTML-Tabelle geschrieben.

function WriteDataInTable(tablename, jsonArray) {

$("#" + tablename).empty();

var parsedArray = JSON.parse(jsonArray);

var table = document.getElementById(tablename);

for (var i = 0; i < parsedArray.length; i++) {

var row = table.insertRow(i);

var cell = row.insertCell(0);

cell.innerHTML = parsedArray[i];

}

}

#### Steuern der Konferenz

Die Konferenz kann mit dem Button „sendButton“ gesteuert werden. Sobald die Konferenz einmal gestartet wurde, wird durch Drücken des Knopfes gleich die Besprechung für die nächste Klasse gestartet.

Die Hub-Methode „ConferenceAction“ aus der Klasse „mainHub“ wird aufgerufen, sobald der Knopf gedrückt wird. Hier wird der aktuelle Raum als Parameter übergeben.

document.getElementById("sendButton").addEventListener("click", function (event) {

connection.invoke("ConferenceAction", GetCurrentRoom()).catch(function (err) {

return console.error(err.toString());

});

event.preventDefault();

});

Wenn die Konferenz noch nicht gestartet wurde, wird die Methode „StartConference“ aufgerufen, ansonsten „NextClass“.

Anschließend werden die Informationen für den Moderator und die Informationen für den normalen Nutzer geladen.

public async Task ConferenceAction(string \_currentroom)

{

Currentroom = \_currentroom;

switch (GetCurrentStateOfConference())

{

case "inactive":

StartConference();

break;

case "running":

NextClass();

break;

}

await LoadModeratorViewInfo(\_currentroom);

await LoadUserViewInfo(\_currentroom);

}

StartConference

public void StartConference()

{

WriteTimeInDatabase("start");

SetStateOfConference("running");

}

Zuerst wird die aktuelle Zeit in die Datenbank geschrieben. Dies geschieht über die Methode „WriteTimeInDatabase“.

Der übergebene Parameter „time“ wird in den SQL-Befehl eingesetzt. Zusätzlich wird noch der aktuelle Klassenname ermittelt, wodurch der Befehl vervollständigt wird.

private void WriteTimeInDatabase(string time)

{

DateTime date = DateTime.Now;

string timeonly = date.ToLongTimeString();

DB.Query($"UPDATE {General.Table\_General} set {time} = ? WHERE ID = ?", timeonly, GetCurrentClassName()) ;

}

Anschließend wird der Status der Konferenz auf „running“ gesetzt. Damit wird signalisiert, dass die Konferenz läuft.

Dies geschieht über die Methode „SetStateOfConference“, welche in die Tabelle „General“ den Status schreibt.

private void SetStateOfConference(string status)

{

DB.Query($"Update {General.TableStateOfConference} set Status = ? where Room = ?", status, Currentroom);

}

NextClass

Zuerst wird die Endzeit der aktuellen Klasse in die Datenbank geschrieben. Danach wird der Status der Klasse von „not edited“ auf „completed“, also auf abgeschlossen, gesetzt.

Falls keine neue Klasse mehr geladen wird, ist die Konferenz abgeschlossen. Wenn dies der Fall ist, wird der Status der Konferenz durch die Methode „SetStateOfConference“ auf „completed“ gesetzt und signalisiert somit den Abschluss der Konferenz in jenem Raum.

Wenn noch nicht alle Klassen abgeschlossen wurden, wird gleich die Startzeit für die nächste Klasse mit „WriteTimeInDataBase“ in die Datenbank geschrieben.

private void NextClass()

{

WriteTimeInDatabase("end");

DB.Query($"UPDATE {General.Table\_General} set Status='completed' WHERE ID = ?", GetCurrentClassName());

if (GetCurrentClassName() == null)

{

SetStateOfConference("completed");

}

else

{

WriteTimeInDatabase("start");

}

}

## Ansicht-Nutzer

Auf der Nutzeransicht werden Informationen über die Konferenz in beiden Räumen angezeigt.

Die Ansicht je Raum besteht aus:

* Raum
* Klassenname
* Klassenvorstand
* Abteilungsvorstand
* Startzeitpunkt
* Abgeschlossene Klassen
* Nächste Klassen

### Laden der Räume

Sobald in der JavaScript-Datei „user\_view.js“ eine Verbindung mit dem Hub steht, wird die Methode „LoadRooms“ aus der Klasse „mainHub“ aufgerufen.

connection.start().then(function () {

connection.invoke("LoadRooms").catch(function (err) {

return console.error(err.toString());

});

}).catch(function (err) {

return console.error(err.toString());

});

Aus der Order-Liste werden die Räume gelesen und getrennt mit Semikolon an JavaScript zurückegegben.

Zuerst werden die Beiden Order-Objekte durch die Methode „GetOrderList“ geladen. Von diesen beiden Objekten, werden dann nur die Raumnamen (z.B. R001) ausgewählt und zu einer Liste konvertiert.

Die Liste wird in Form eines JSON-Arrays an den Client zurückgegeben.

public async Task LoadRooms()

{

var order = GetOrderList();

JArray jArrayRooms = new JArray(order.Select(room => room.Room\_only).ToList());

await Clients.All.SendAsync("ReceiveRooms", jArrayRooms.ToString());

}

In JavaScript wird die Liste geparst und die Räume in das entsprechende Element in HTML geschrieben. Dafür wurde eine entsprechende Methode mit dem Namen „WriteInElement“ implementiert, da diese noch öfter zum Einsatz kommt.

Die Methode erhält zwei Parameter. Der „elementname“ ist der Name des Elements in HTML. Der Wert „value“ ist der Text, welcher in das Element geschrieben werden soll.

connection.on("ReceiveRooms", function (order) {

var order\_parsed = JSON.parse(order);

WriteInElement("c1\_room", order\_parsed[0]);

WriteInElement("c2\_room", order\_parsed[1]);

for (var i = 0; i < order\_parsed.length; i++) {

connection.invoke("LoadUserViewInfo", order\_parsed[i]).catch(function (err) {

return console.error(err.toString());

});

}

});

Die Methode sucht das Element mit der entsprechenden ID in HTML und setzt den Wert des Elements.

function WriteInElement(elementname, value) {

var element = document.getElementById(elementname);

element.innerHTML = value;

}

Anschließend werden mittels einer Schleife für jeden Raum die Informationen geladen. Dies geschieht über die Methode „LoadUserViewInfo“, welche den Raum als Parameter übergeben bekommt.

### Laden der Informationen

Zuerst wird in der Methode der aktuelle Raum gesetzt, sodass darauf zugegriffen werden kann.

Anschließend wird ein neues JSON-Objekt erstellt. Durch einen Switch werden die entsprechenden Daten in das Objekt geschrieben und letztendlich als JSON-String zurückgegeben.

public async Task LoadUserViewInfo(string \_currentroom)

{

Currentroom = \_currentroom;

JObject information = new JObject();

var currentClass = GetClass(GetCurrentClassName());

switch (GetCurrentStateOfConference())

{

case "inactive":

information.Add("room", Currentroom);

information.Add("classname", GetCurrentClassName());

information.Add("formteacher", currentClass.FormTeacher);

information.Add("head\_of\_department", currentClass.HeadOfDepartment);

information.Add("time", "Besprechung noch nicht gestartet");

information.Add(new JProperty("classes\_not\_edited", GetClassesFromJSON("next")));

break;

case "running":

DataTable dt = DB.Reader($"SELECT room, start FROM {General.Table\_General} WHERE ID = ? limit 1", GetCurrentClassName());

information.Add("room", dt.Rows[0]["room"].ToString());

information.Add("time", dt.Rows[0]["start"].ToString());

information.Add("classname", GetCurrentClassName());

information.Add("formteacher", currentClass.FormTeacher);

information.Add("head\_of\_department", currentClass.FormTeacher);

information.Add(new JProperty("classes\_not\_edited", GetClassesFromJSON("next")));

break;

case "completed":

information.Add("room", Currentroom);

information.Add("classname", "Alle Klassen abgeschlossen");

information.Add("formteacher", "-");

information.Add("head\_of\_department", "-");

information.Add("time", "-");

information.Add("classes\_not\_edited", "-");

break;

}

information.Add("classes\_completed", GetClassesFromJSON("previous"));

await Clients.All.SendAsync("ReceiveUserViewInfo", information.ToString());

}

Sobald das Objekt erhalten wurde, wird dieses getrennt und kann nun verwendet werden.

Um zu verhindern, dass die Informationen für den falschen Raum eingetragen werden, wird dieser überprüft.

Durch das Laden der Räume stehen die beiden Räume bereits auf der HTML-Ansicht. Nun wird einer der beiden Räume mit dem Raumnamen des Objekts verglichen. Wenn diese nun gleich sind, werden die Informationen für diesen Raum eingetragen. Ansonsten werden die Informationen für den Anderen Raum geschrieben.

connection.on("ReceiveUserViewInfo", function (myobject) {

var obj\_parsed = JSON.parse(myobject);

if (document.getElementById("c1\_room").innerHTML == obj\_parsed.room) {

WriteUserViewInformation("c1\_", obj\_parsed);

}

else {

WriteUserViewInformation("c2\_", obj\_parsed);

}

});

Zu guter Letzt wird die Startsequenz und das Objekt an die Methode „WriteUserViewInformation“ übergeben. Die Startsequenz kennzeichnet, um welchen Raum es sich handelt. Wenn das Kürzel „r1\_“ ist, werden die Informationen für alle Objekte mit der Startsequenz „r1\_“ geschrieben.

Hier kommt wider die Methode „WriteInElement“ zum Einsatz. Es werden die Werte für alle HTML-Elemente mit der entsprechenden Startsequenz gesetzt.

function WriteUserViewInformation(element, obj\_parsed) {

WriteInElement(element + "room", obj\_parsed.room)

WriteInElement(element + "classname", obj\_parsed.classname);

WriteInElement(element + "formteacher", obj\_parsed.formteacher);

WriteInElement(element + "head\_of\_department", obj\_parsed.head\_of\_department);

WriteInElement(element + "time", obj\_parsed.time);

WriteDataInTable(element + "classes\_not\_edited", obj\_parsed.classes\_not\_edited);

WriteDataInTable(element + "classes\_completed", obj\_parsed.classes\_completed);

}

Da die abgeschlossenen und nächsten Klassen mehrere Werte besitzen, werden diese in einen HTML-Tabelle geschrieben. Die Methode „WriteDataInTable“ bekommt den Elementnamen und die Klassen übergeben.

Zuerst wird das Element mit dem entsprechenden Namen gesucht und geleert. Anschließend wird das JSON-Array geparst und die Werte in die HTML-Tabelle geschrieben.

function WriteDataInTable(tablename, jsonArray) {

$("#" + tablename).empty();

var parsedArray = JSON.parse(jsonArray);

var table = document.getElementById(tablename);

for (var i = 0; i < parsedArray.length; i++) {

var row = table.insertRow(i);

var cell = row.insertCell(0);

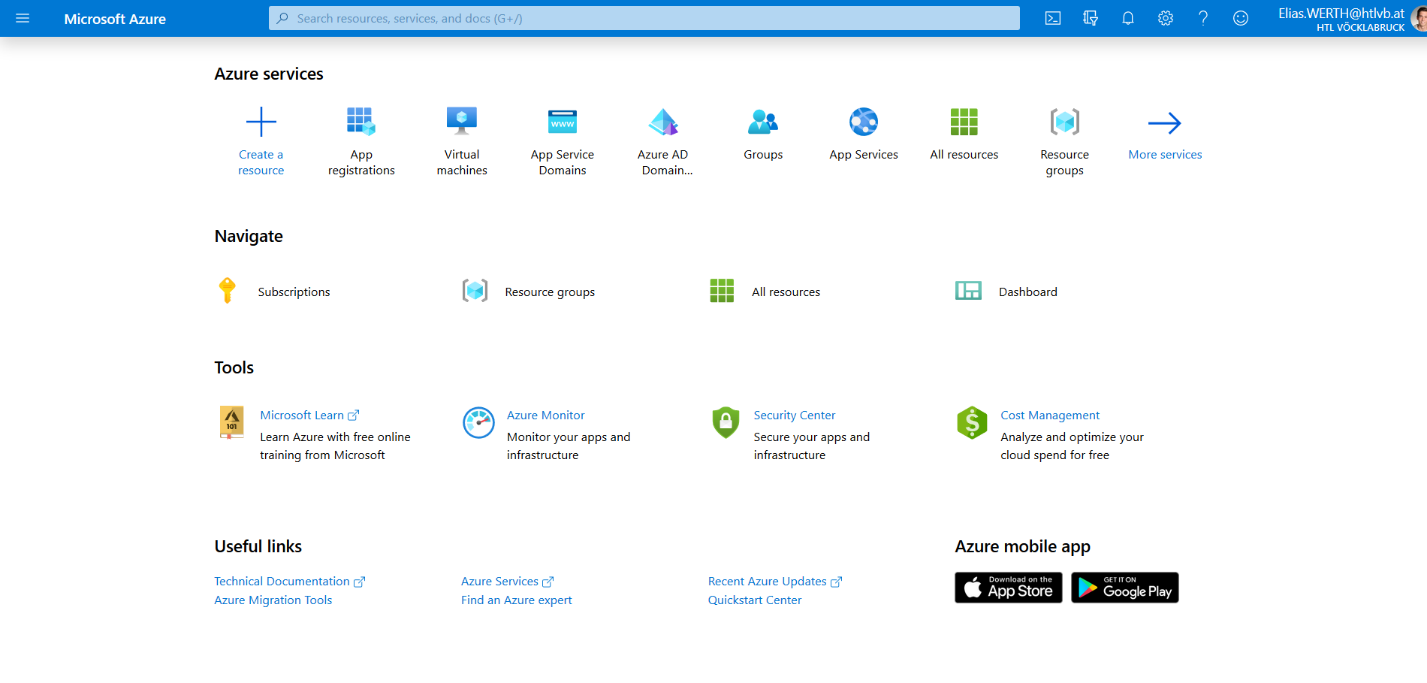
cell.innerHTML = parsedArray[i];

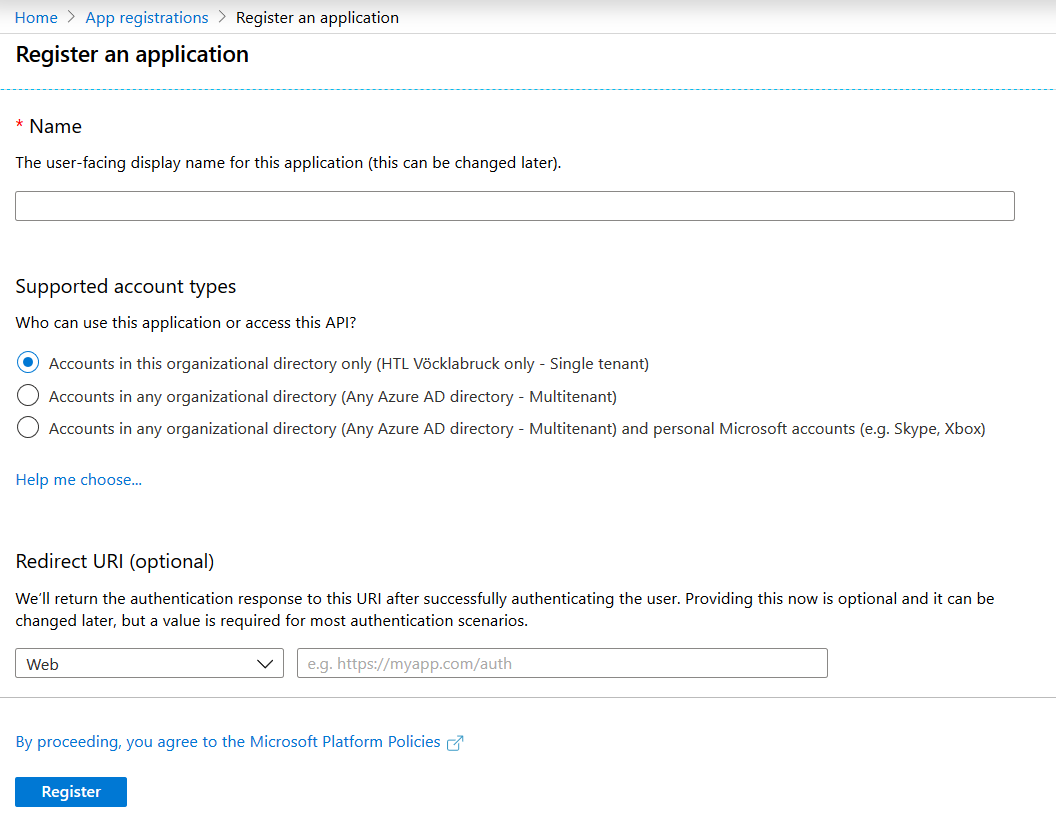
}

}

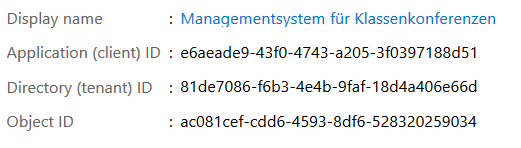
## Authentifizierung mit Microsoft Azure AD

### Registrierung im Azure-Portal

Bevor man fähig ist, den Office-Login zu implementieren, muss man seine Anwendung Azure-Portal registrieren. Im ersten Schritt navigiert man zu *portal.azure.com*.

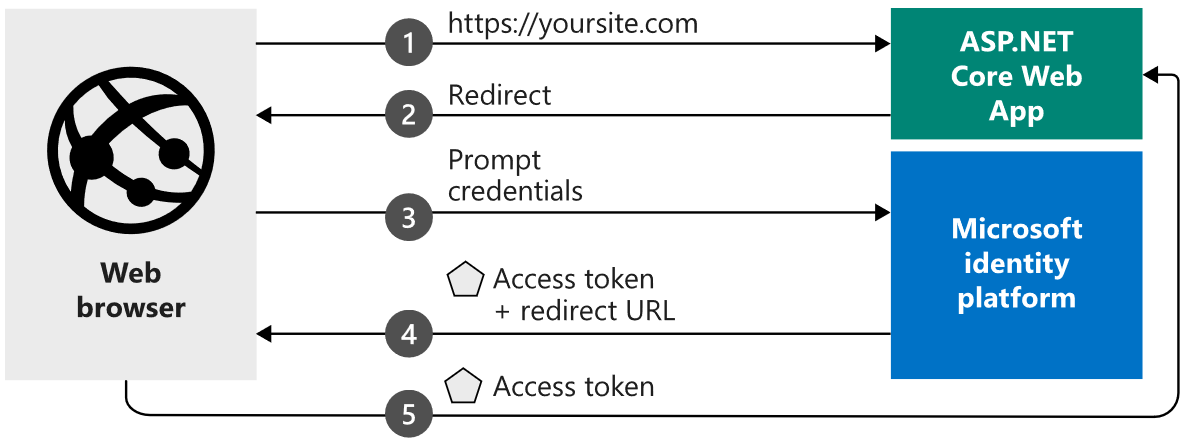
Unter dem Reiter „App registrations“ gelangt man auf eine Seite, auf der man vor allem seine bereits registrierten Applikationen verwalten, aber auch mit dem Button „New registration“ eine neue Anwendung registrieren kann. Hier muss man nur den Namen der Applikation eingeben und schon ist die Registrierung fertig.

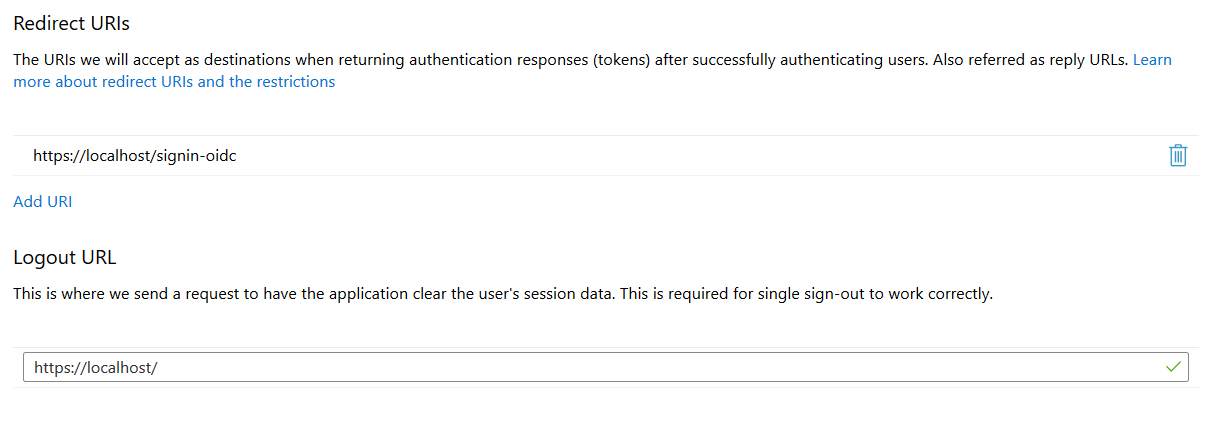
### Konfiguration der Applikation

Nach der Registrierung ist man sofort in der Lage seine Anwendung zu konfigurieren, denn man bekommt darauf all die wichtigen Informationen, die man dazu braucht.

Besonders die angegebenen IDs sind essenziell für eine korrekt Verbindung zwischen Anwendung und Azure-Portal:

* Application (client) ID
  + Die Client ID ist eine öffentliche Kennung für Anwendungen. Obwohl sie öffentlich ist, sollte sie nicht an Dritte weitergegeben werden. Ebenfalls muss die Client ID einzigartig sein.
  + <https://www.oauth.com/oauth2-servers/client-registration/client-id-secret/>
* Directory (tenant) ID
  + Die Tenant ID ist ein globally unique identifier (GUID). Diese ID gibt an, in welcher Azure-AD-Instanz sich die Anwendung befindet.
  + <https://microsoft.github.io/AzureTipsAndTricks/blog/tip153.html>

Die Quickstart-Guides dieser Seite bieten eine perfekte Schritt-für-Schritt-Anleitung an, die für das Managementsystem für Klassenkonferenzen hervorragend funktionieren. Zusätzlich wird hier auch erklärt wie die Authentifizierung funktioniert.

Im ersten Schritt muss im Azure-Portal der Redirect-URL konfiguriert werden. Ohne den richtigen Redirect-URL ist es nicht möglich, die Anwendung nach dem Login bzw. Logout zurückzuleiten, vorausgesetzt die Authentifizierungsantwort (Token) ist erfolgreich.

Der Port ist in diesem Fall nicht weiter wichtig. Essenziell ist nur der *localhost* und vor allem das *http****s****://*. Dasselbe gilt auch für den Logout-URL. Die Endung *signin-oidc* ist der Standardwert der OIDC-Client-Middleware und ist wesentlich für die korrekte Umleitung.

Im nächsten großen Schritt wird die appsettings.json-Datei konfiguriert. Hierbei werden alle zentralen AzureAD-Informationen eingetragen. Dazu gehören die Instanz (hier: Microsoft), die Domäne, auf der die Anwendung läuft und die bereits kennengelernten IDs.

"AzureAD": {

"Instance": "https://login.microsoftonline.com",

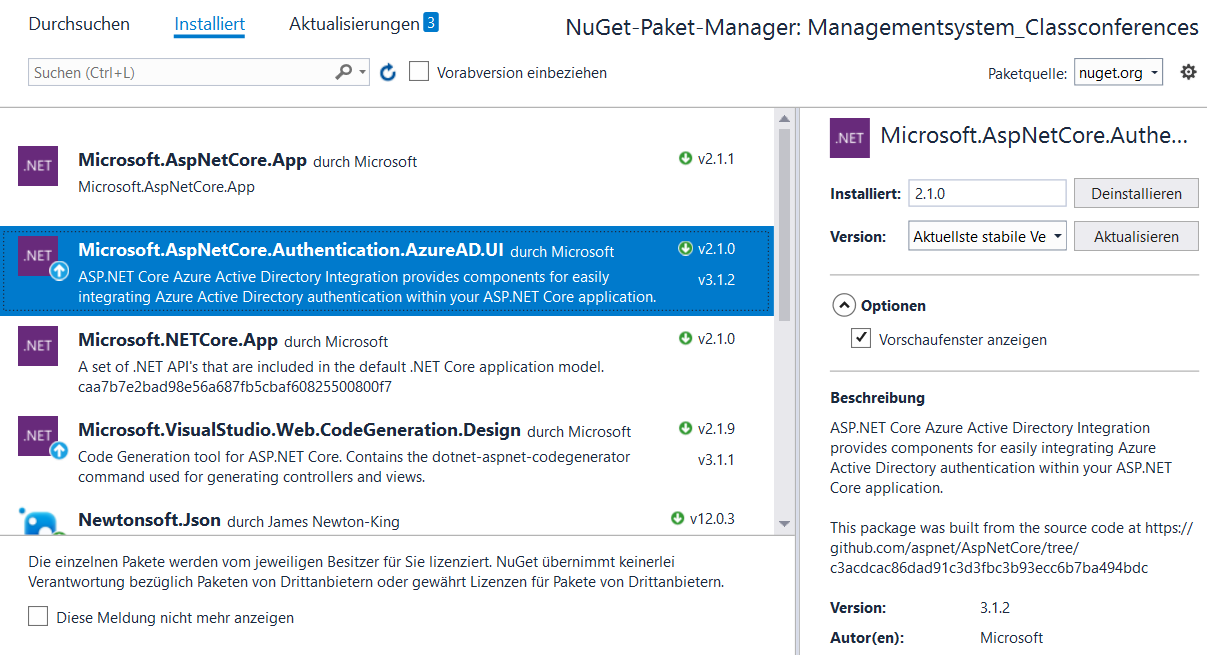
"Domain": "htlvb.at",

"ClientId": "e6aeade9-43f0-4743-a205-3f0397188d51",

"TenantId": "81de7086-f6b3-4e4b-9faf-18d4a406e66d"

},

### Implementierung des Logins

Bei der Implementierung von Microsoft Azure AD wird der Login korrekt in die Anwendung eingebaut. Der erste und wichtigste Schritt findet in der Startup.cs-Klasse statt, doch bevor man dazu fähig ist, braucht man das entsprechende NuGet-Paket.

Nach der Installation ist man nun im Stande, mit dem entsprechenden Code, AzureAD zu implementieren.

using Microsoft.AspNetCore.Authentication.AzureAD.UI;

using Microsoft.AspNetCore.Authentication;

[...]

public void ConfigureServices(IServiceCollection services)

{

[...]

services.AddAuthentication(AzureADDefaults.AuthenticationScheme

.AddAzureAD(options => Configuration.Bind("AzureAD", options));

services.Configure<OpenIdConnectOptions>(AzureADDefaults.OpenIdScheme, options =>

{

options.Authority = options.Authority + "/v2.0/";

options.TokenValidationParameters.ValidateIssuer = false;

});

[...]

}

[...]

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

[...]

app.UseAuthentication();

[...]

}

Mit using Microsoft.AspNetCore.Authentication.AzureAD.UI; wird die Klasse AzureADDefaults aktiviert. Das Using kann man mit der Installation des NuGet-Pakets hinzugefügt werden.

Mit app.UseAuthentication(); wird die Authentifizierungsmiddleware zum spezifischen IApplicationBuilder hinzugefügt, was wiederum die benötigten Authentifizierungsfähigkeiten aktiviert.

Der nächste und letzte wichtige Schritt ist die Implementierung des eigentlichen Buttons in der \_Layout.cshtml-Page. Vorgesehen ist dafür ein Button in der rechtsbündigen Navbar mit entsprechendem Icon.

<div class="navbar-collapse collapse">

@using System.Security.Claims

@if (User.Identity.IsAuthenticated)

{

var identity = User.Identity as ClaimsIdentity;

string preferredUsername = identity.Claims.FirstOrDefault(c => c.Type == "name")?.Value;

<ul class="nav navbar-nav navbar-right">

<li><a href="https://www.office.com/"><span class="glyphicon glyphicon-user"></span> @preferredUsername</a></li>

<li><**a** **asp-area**="AzureAD" **asp-controller**="Account" **asp-action**="SignOut"><span class="glyphicon glyphicon-log-out"></span> Log Out</**a**></li>

</ul>

}

else

{

<ul class="nav navbar-nav navbar-right">

<li>

<**a** **asp-area**="AzureAD" **asp-controller**="Account" **asp-action**="SignIn">

<span class="glyphicon glyphicon-log-in"></span> Login

</**a**>

</li>

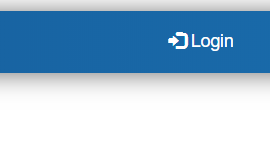
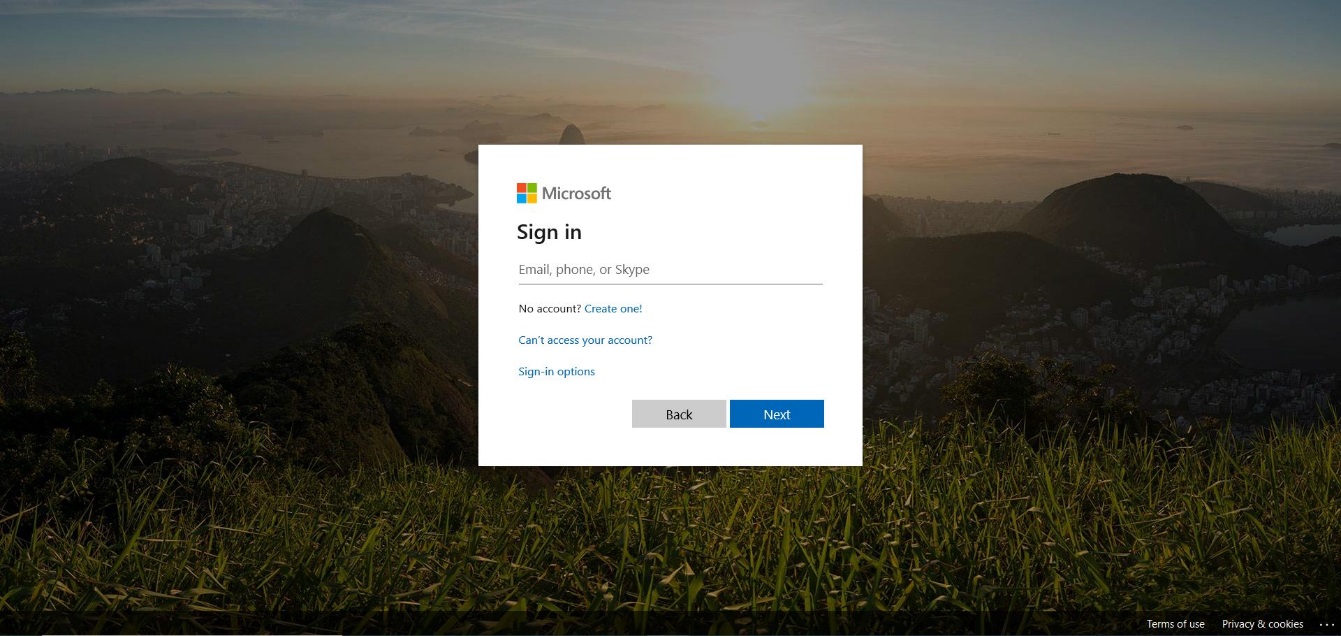
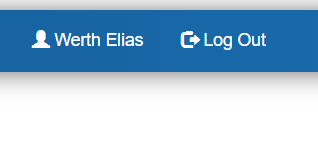
</ul>

}

</div>

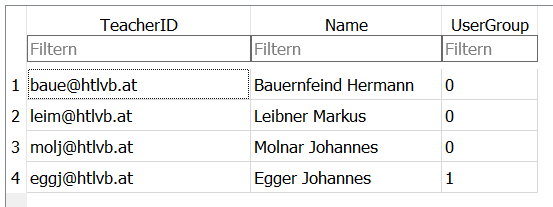
Zuerst wird überprüft, ob der Benutzer eingeloggt ist. Der Login-Button soll angezeigt werden, wenn er nicht angemeldet ist. Der Logout-Button, wenn er angemeldet ist.

Hinzu kommt, dass Vor- und Nachname links neben dem Button angezeigt werden soll, wenn er eingeloggt ist. Mit ClaimsIdentity werden einige essenzielle Daten des Accounts gesammelt. Danach wird nur noch der preferredUsername im Listitem ausgegeben. Mit einem Klick auf diesen Namen gelangt der Benutzer sofort zur Office-365-Startseite.

Nun ist die Applikation fertig konfiguriert und der Login vollständig implementiert. Der Authentifizierung funktioniert nun erfolgreich.

### Weiterleitung nach erfolgreicher Authentifizierung

Ein weiter wichtiger Teil des Logins, der auf keinen Fall vergessen werden darf, ist die Weiterleitung. Die Idee dabei ist folgende: Der Moderator soll auf die Seite für die Raumauswahl gelangen und der normale Benutzer soll zur Konferenzübersicht gelangen. Zusätzlich wird der Admin (Johannes Egger) zu der Seite für die Admineinstellungen weitergeleitet. Als Moderatoren zählen in diesem Fall die Abteilungsvorstände. Alle anderen Lehrkräfte haben kein Zugriffsrecht auf die Moderatorseite.

Der erste Schritt besteht daraus, eine neue Tabelle in der Datenbank anzulegen. Mit dem Namen „UserRights“ bestimmt diese, welche Rechte die jeweilige Lehrkraft hat. Die Spalten sind TeacherID, Name und UserGroup. Die TeacherID ist in diesem Fall die E-Mail-Adresse und die UserGroup bestimmt welches Recht die Lehrkraft besitzt. Die 0 repräsentiert hierbei die Abteilungsvorstände und die 1 steht für den Admin. Die Spalte „Name“ dient nur zur Vervollständigung.

Im nächsten Schritt wird die Abfrage nach den Rechten ins Programm implementiert. Das Wichtigste spielst dabei in der Index.cshtml.cs in der OnGet()-Methode ab, denn diese Methode wird jedes Mal abgerufen, wenn die Index-Seite neu lädt (also auch nach dem Login).

if (User.Identity.IsAuthenticated)

{

var identity = User.Identity as ClaimsIdentity;

string teacherId = identity.Claims.FirstOrDefault(c => c.Type ==

"preferred\_username")?.Value;

DataTable TeacherRight = DB.Reader($"SELECT UserGroup from

{general.TableUserRights} WHERE TeacherID LIKE ? LIMIT 1", teacherId);

if (TeacherRight.Rows.Count != 0)

{

if (Convert.ToInt64(TeacherRight.Rows[0][0]) == 0)

{

return new RedirectToPageResult("roomselection");

}

else return new RedirectToPageResult("admin\_settings");

}

else return new RedirectToPageResult("conference");

}

else return null;

Mit User.Identity.IsAuthenticated wird im ersten Schritt überprüft, ob der Benutzer überhaupt angemeldet ist. Im Falle eines Users, welcher nicht eingeloggt ist, wird lediglich ein null zurückgegeben. Andernfalls wird im nächsten Schritt, wie bei der Authentifizierung, mit ClaimsIdentity die E-Mail der angemeldeten Lehrkraft ermittelt. Im Anschluss setzt man eine SQL-Select-Anweisung ein, in der man die „UserGroup“ der Lehrkraft ermittelt. Die Tabelle „UserRights“ wurde schon im Vorfeld zu der Klasse General.cs hinzugefügt und kann so mit general.TableUserRights abgerufen werden.

Natürlich stehen nur Moderatoren und Admins in der Tabelle. Das bedeutet, dass wenn sich eine Lehrkraft ohne jegliche Rechte anmeldet, der DataTable leer bleibt. Genau dies wird in der ersten if-Abfrage überprüft. Wenn es wahr ist, dass der DataTable leer ist, wird die Lehrkraft mit RedirectToPageResult zur Konferenz-Page (conference) weitergeleitet.

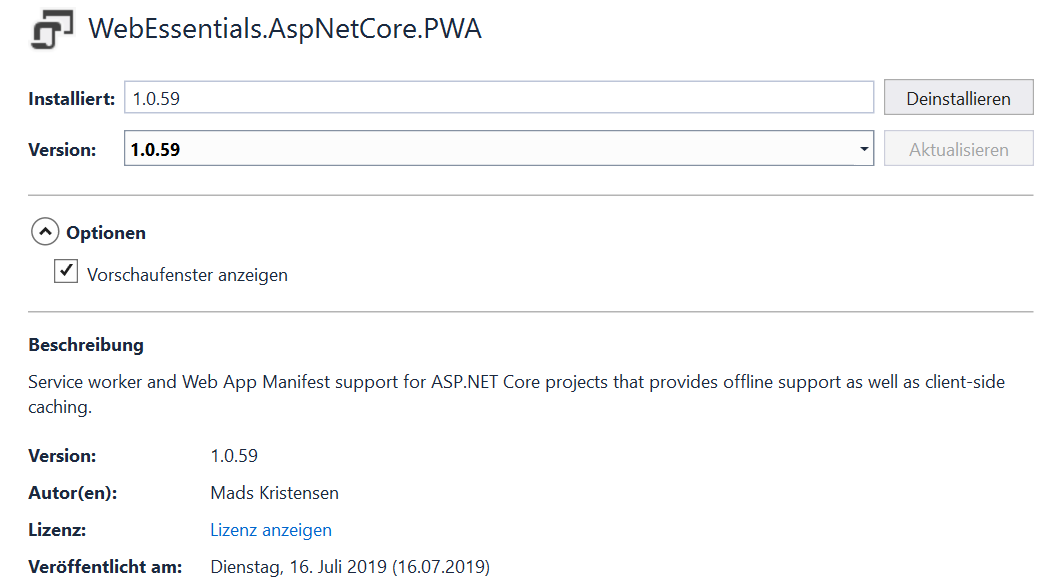
Falls in den DataTable etwas geschrieben wurde, bedeutet dies, dass sich entweder ein Moderator oder ein Admin angemeldet hat. Genau das wird in der nächsten if-Abfrage überprüft und auch hier wieder mit RedirectToPageResult auf die entsprechende Seite weitergeleitet (roomselection bzw. admin\_settings).

## Progressive Web-App (PWA)

Um die ASP.NET Core Website in eine PWA umzuwandeln sind einige Schritte notwendig. Der erste und wohl wichtigste Schritt besteht daraus, einen **Service Worker** und ein **Manifest** in die Anwendung zu implementieren.

### Service Worker

Der Service Worker ist eine Art von Web Worker. Im Wesentlichen ist es ein JavaScript-File, das im Hintergrund und unabhängig vom Browser-Thread arbeitet. Der Service Worker macht Caching einfach und definiert Regeln, wenn bestimmter Inhalt der Seite nicht vorhanden ist.

Um den Service Worker kümmert sich mehr oder weniger das NuGet-Paket „WebEssentials.AspNetCore.PWA“, entwickelt von Mads Kristensen.

Dieses NuGet erstellt nicht nur den Service Worker, es kummert sich auch um dessen Implementierung in die Anwendung. Erwähnenswert ist hierbei, dass der Service Worker im Programm nicht einsehbar ist, und somit nicht verändert werden kann.

Ein essenzieller Schritt, der gemacht werden muss, bevor der Service Worker funktionieren kann, ist die Einbindung der PWA in die Startup.cs-Klasse.

public void ConfigureServices(IServiceCollection services)

{

[...]

services.AddProgressiveWebApp();

[...]

}

Durch diesen Service wird die Referenz zum Service Worker erstellt und die Nutzung des Service Workers wird dadurch garantiert.

Weitere Einstellungen bezüglich der PWA werden in der appsettings.json-Datei definiert.

{

[...]

"pwa": {

"cacheId": "Worker 1.1",

"strategy": "cacheFirstSafe",

"routesToPreCache": "/Index, /Conference",

"registerServiceWorker": true,

"registerWebmanifest": true

}

}

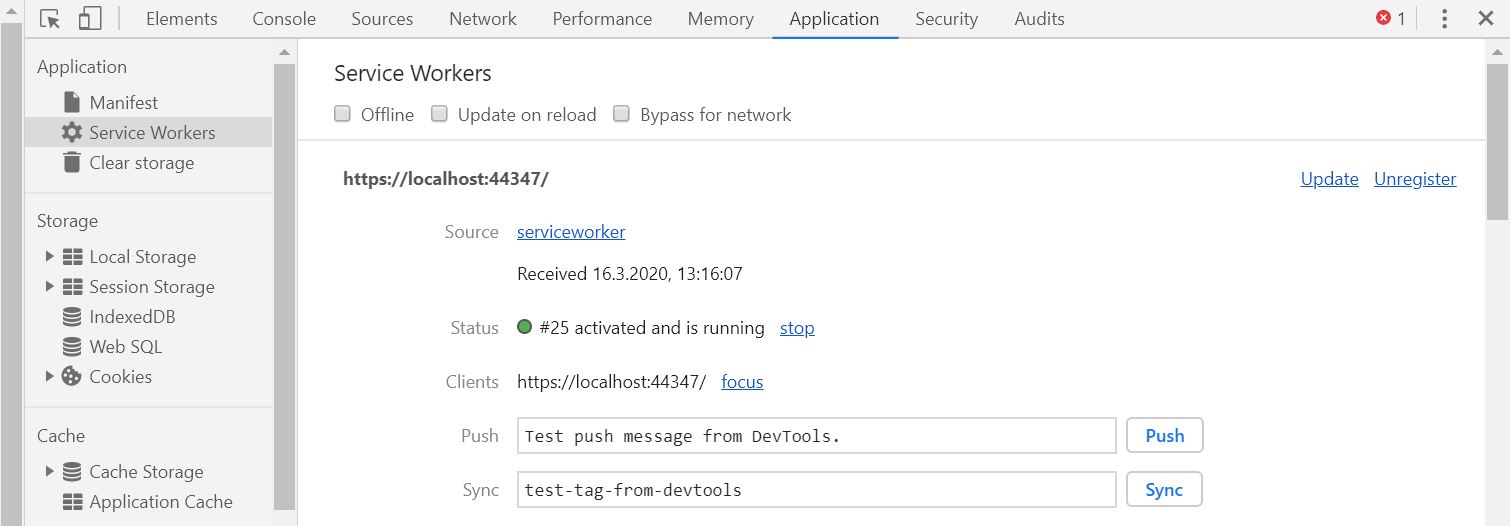
Das Merkmal cacheId definiert eine eindeutige ID für die aktuelle Version der Seite und wird im Service Worker verwendet, um zu überprüfen, ob der Benutzer derzeit über die neueste Version der Seite verfügt.

Mit strategy wählt man unter einigen vordefinierten Profilen eine Strategie für den Service Worker aus. Die Strategie cacheFirstSafe ist die Standardstrategie, mit der jede Ressource zwischengespeichert wird und sichergestellt wird, dass beim ersten Versuch über das Netzwerk auf Doctype text/html zugegriffen wird, um die neueste Version abzurufen. Wenn dies nicht verfügbar ist, wird auf den Cache zurückgegriffen.

routeToPreCache gibt an, welche Seiten vor dem cachen gespeichert werden sollen. Dies bedeutet, dass sie dem Cache hinzugefügt werden, sobald der Service Worker registriert wurde. Dies ist besonders nützlich, um alle Unterseiten auf der Webpage zwischenzuspeichern, damit der Benutzer diese besuchen kann, während er offline ist.

registerServiceWorker und registerWebmanifest dienen zum Deaktivieren einzelner Funktionen im Service.

Auf dem jetzigen Stand der PWA ist es bereits möglich zu sehen ob der Service Worker aktiv ist und läuft. Dies kann man in den Chrome DevTools unter Application > Service Worker einsehen.



### Manifest

Ausständig bleibt nun nur noch das Erstellen eines Web-App-Manifests. Das Manifest ist ein JSON-File, das dem Browser wichtige Information der Progressive Web-App übermittelt. Es sagt den Browser auch, wie es sich verhalten soll, wenn es am Mobiltelefon des Benutzers installiert wird.

Das Erstellen des Manifests ist keine große Herrausforderung, jedoch muss beachtet werden, dass sich die JSON-Datei im wwwroot-Ordner befindet. Der Name dieser Datei spielt für gewöhnlich keine entscheidende Rolle, jedoch ist eine sprechende Bezeichnung zu bevorzugen. Hier wurde der Name manifest.json gewählt.

Der Inhalt des Manifests ist folgendermaßen strukturiert:

{

"name": "Managementsystem für Klassenkonferenzen",

"short\_name": "Klassenkonferenzen",

"description": "Das Managementsystem der HTL VB für Klassenkonferenzen",

"display": "standalone",

"icons": [

{

"src": "/images/logox192.png",

"type": "image/png",

"sizes": "192x192"

},

{

"src": "/images/logox512.png",

"type": "image/png",

"sizes": "512x512"

}

],

"start\_url": "/",

"color": "#134883",

"background\_color": "#2b2b2b",

"theme\_color": "#134883"

}

Die Eigenschaft name ist der volle Name der Anwendung. Solange name zur Verfügung steht, ist es nicht unbedingt notwendig, die Eigenschaft short\_name auch in das Manifest einzubinden. Nichtsdestotrotz hat man sich hier dafür entschieden, denn short\_name wird verwendet, wenn der Benutzer die Web-App auf seinem Homescreen bzw. Desktop installiert.

Mit display wird definiert welches User Interface für den Browser gewählt wird. Das Item standalone öffnet die Web-App so, dass sie wie eine eigenständige native App aussieht und sich auch so anfühlt. Die App wird in einem eigenen Fenster ausgeführt, das vom Browser getrennt ist, und verbirgt Standardelemente der Browser-Benutzeroberfläche wie die URL-Leiste.

Die icons-Eigenschaft ist ein Array von Bildern. Jedes Objekt braucht eine src (Quelle), die sizes (Abmessungen) und den type (Art) des Objekts. Für Chrome braucht man mindestens ein 192x192pixel-Icon und ein 512x512pixel-Icon. Falls nur diese zwei Icons vorhanden sind, skaliert Chrome die Icons automatisch auf die Größe, die für das Gerät geeignet ist. Die benötigten Icons müssen sich in dem Ordner „images“ der Projektmappe befinden.

Die start\_url ist erforderlich und teilt dem Browser mit, wo Ihre Anwendung beim Ausführen gestartet werden soll, und verhindert, dass die App auf der Seite gestartet wird, auf der sich der Benutzer befand, als er Ihre App zu seinem Startbildschirm hinzufügte.

Die Eigenschaften color und background\_color hängen mit dem Splash-Screen zusammen. Der Splash-Screen wird angezeigt, wenn die App zum ersten Mal am Mobilgerät gestartet wird. Es ist also nichts weiter als ein grafischer Platzhalter, der solange zu sehen ist, wie die App zum Laden benötigt. Die color ist in diesem Fall die Schriftfarbe, in der der Titel der Anwendung angezeigt wird und die background\_color ist die Hintergrundfarbe des Splash-Screens. Ebenfalls am Splash-Screens zu sehen am Splashscreen ist das Icon.

Die theme\_color ist ein einfaches Attribut, dass bestimmt, welche Farbe der Toolbar bestimmt. Wie der Name schon sagt, ist es das Farbschema der Seite.

Das Manifest ist nun komplett und der letzte essenzielle Schritt ist es, das Manifest mit einem <link>-Tag zu verlinken. Dies geschieht im <head> der \_Layout.cshtml-Page.

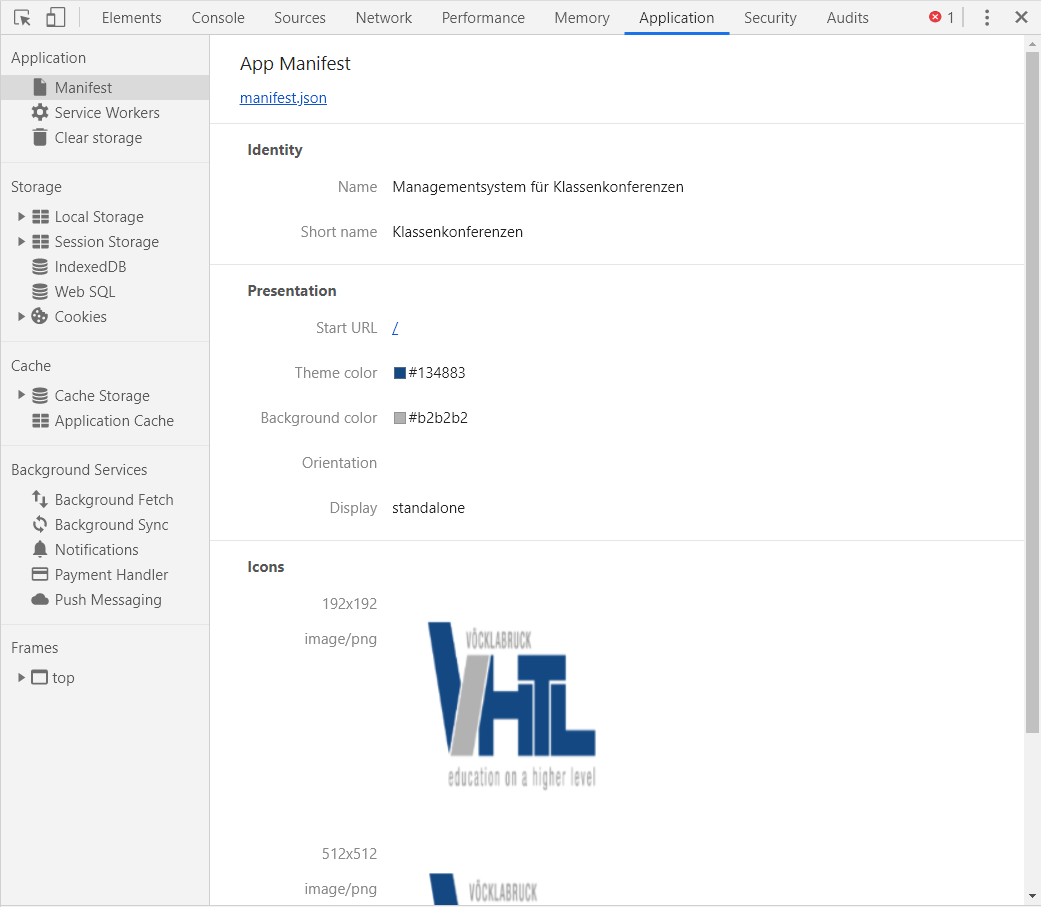
<**environment** **include**="Development">

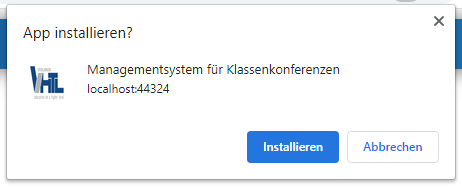
[...]

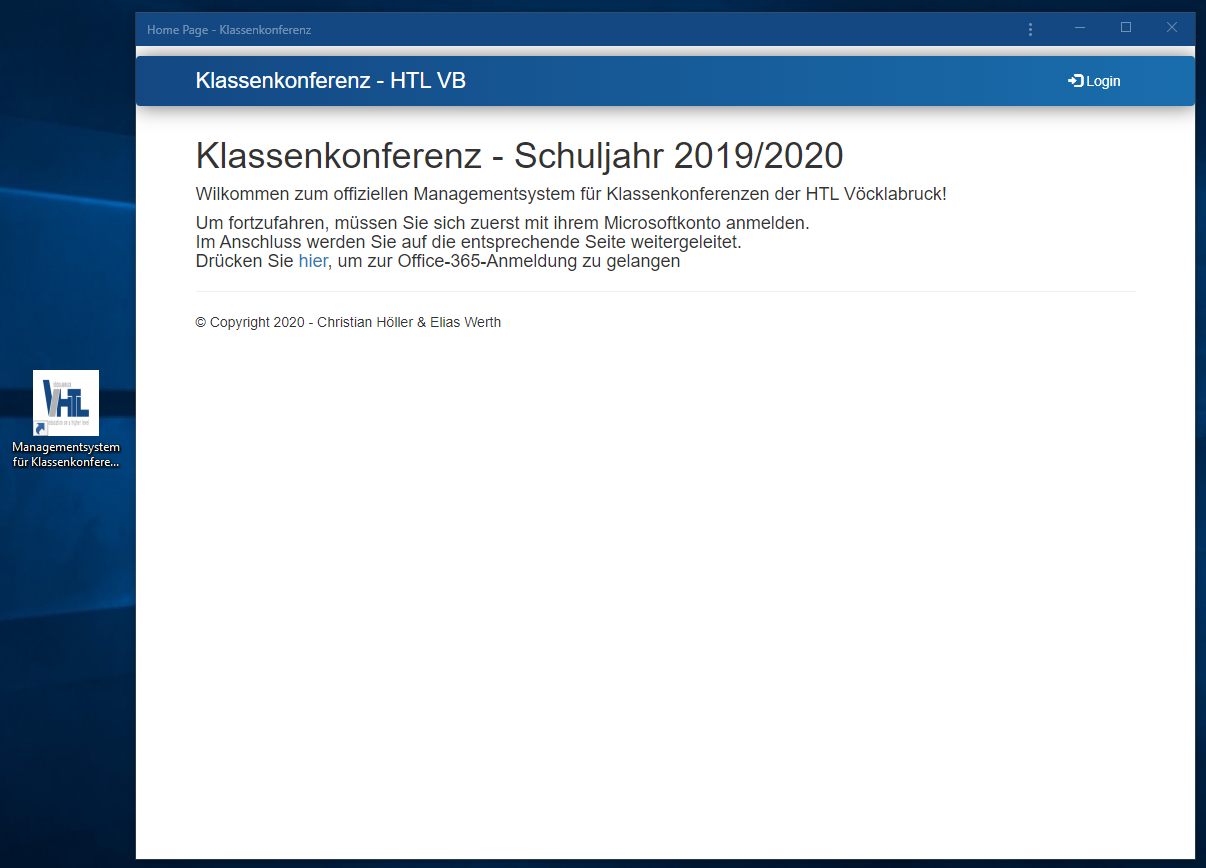
<link rel="manifest" href="~/manifest.json" />

[...]

</**environment**>

Danach ist die Progressive Web App startklar und die Anwendung ist bereit zum Testen. Mithilfe der Chrome DevTools hat man auch hier die Möglichkeit unter Application > Manifest alle wichtigen Details einzusehen.

Auch die Installation der Instanz ist nun bereit. Für Benutzer die über den PC auf das Managementsystem gelangen, wird die Installation direkt nach dem Abruf der Webseite (in Chrome) vorgeschlagen.

Und auch schon direkt nach dem Mausklick auf den Installations-Button öffnet sich die installierte Progressive Web App. Auch das Icon ist von jetzt an am Desktop sichtbar.

## Ausrufen der Lehrkräfte

## Design der Web-App

# Dokumentation der individuellen Themenstellung 1

Dieses Kapitel sollte entsprechend dem individuellen Arbeitspaket der Schülerin/des Schülers 1 doku­mentiert werden und folgendes umfassen:



Keine Angst vor unterschiedlichen Schreibstilen!

Die Dokumentation wird sowie die gesamte Diplomarbeit individuell beurteilt!

* Grundlagen, Methoden
* Ist-Situation
* Lösungsansätze
* Begründungen
* Ergebnisse

Es wird empfohlen, wie in diesem Dokument bereits vorbereitet, für diese individuellen Anteile eigene Abschnitte in Microsoft Word zu erzeugen. Für jeden Abschnitt kann dann in einer eigenen Kopfzeile der Namen des verantwortlichen Verfassers angegeben werden!

Es ist grundsätzlich darauf zu achten, dass nur für die Diplomarbeit relevante Fakten angeführt werden. Nicht die Seitenanzahl entscheidet über die Beurteilung, sondern Qualität vor Quantität sollte das Ziel sein!

Die Schüler/innen sind verpflichtet, ein Projekttagebuch zu führen. Dieses umfasst:

* Begleitprotokoll zur Dokumentation der Besprechungen
* Stundenaufzeichnung mit Angabe der Tätigkeit

Diese beiden Dokumente müssen von jedem Schüler/jeder Schülerin geführt werden, sollen aber nicht an dieser Stelle, sondern im Anhang beigelegt werden.

# Dokumentation der individuellen Themenstellung 2

Hier soll der individuelle Anteil der Schülerin/des Schülers 2 zusammengefasst werden.

Zur Erinnerung an dieser Stelle! Eine Diplomarbeit ist eine wissenschaftliche Arbeit, die einen entsprechenden Schreibstil verlangt. Dazu einige grundsätzliche Anmerkungen:

* Meiden Sie die „Ich“- bzw. „Wir“-Form.
* Sätze sollen inhaltlich nachvollziehbar aneinander gereiht sein, klare und präzise Satzanschlüsse erleichtern das Lesen.
* Verknüpfen Sie Satzteile korrekt miteinander.
* Finden Sie eine prägnante und leser/Innenfreundliche Wortwahl und Formulierung.
* Verwenden Sie aussagekräftige Zeitwörter, die den Text lebendig werden lassen.



Eine Arbeit ohne Quellangaben ist keine wissenschaftliche Arbeit und kann rechtliche und finanzielle Folgen haben!

Beim Verfassen dieser individuellen Passagen muss natürlich auf folgendes geachtet werden.

* Richtiges Zitieren   
  Wie bei Zitaten sind natürlich auch bei Bildern, Fotos, Prospekten, Tabellen, etc. die Quellen anzugeben. Diese sind zusätzlich im Literaturverzeichnis anzuführen.   
  **⇨ Zitierregeln.pdf**
* Erstellung eines Literaturverzeichnisses
* Fortlaufende Nummerierung von Abbildungen bzw. Tabellen und Führung eines Abbildungs- bzw. Tabellenverzeichnisses   
  **⇨ Tipps zur Erstellung dieser Verzeichnisse siehe ANHANG E**

An dieser Stelle ein kleiner Hinweis zu Recherchen im Internet!

Das World-Wide-Web hat die Recherche erleichtert und der Zugang zu Informationen wurde wesentlich vereinfacht. Diese Entwicklung hat jedoch nicht nur Vorteile mit sich gebracht, da allein die Fülle an Informationen es schwermacht, die Zuverlässigkeit der Quellen einzuschätzen. Die folgenden Fragen können die Bewertung von Internetseiten erleichtern:

* Wird die/der Autor/in der Seite genannt? […]
* Ist die Seite aktuell und einfach zugänglich?
* Wer betreibt den Server (Forschungseinrichtungen, öffentliche Einrichtungen etc)?
* An welche Zielgruppe richtet sich die Seite? Ist sie wissenschaftlich oder kommerziell angelegt?
* Werden Quellen richtig und vollständig angegeben?
* Ist die formale Qualität gegeben? Rechtschreib- oder Tippfehler?
* Sind weitere gültige Links angegeben?

# Zusammenfassung

In diesem Kapitel kann von den Schüler/innen gemeinsam eine Zusammenfassung eingefügt werden.

# Literatur- und Quellverzeichnis

Bundesministerium für Bildung. (Jänner 2016). *Leitfaden für die Durchführung von Diplomarbeiten.* (Bundesministerium für Bildung, Hrsg.) Wien: Eigendruck.

# Abbildungsverzeichnis

Abbildung 1 Screenshot Formatvorlage ändern 11

Abbildung 2 Screenshot Quelle erstellen 12

Abbildung 3 Screenshot Beschriftung einfügen 13

###### - Terminplan

**[](https://www.google.at/imgres?imgurl=https://previews.123rf.com/images/daksel/daksel1111/daksel111100005/11137191-gelben-Aufkleber-beigef-gten-Zeichnung-pin-Lizenzfreie-Bilder.jpg&imgrefurl=https://de.123rf.com/clipart-vektorgrafiken/stift_icon.html&docid=hNG3z7d6vhg7fM&tbnid=AFQGbeFzQxalXM:&vet=10ahUKEwjZhtLw7IPUAhXJIJoKHbiQCo4QMwhbKDEwMQ..i&w=1300&h=1300&bih=822&biw=1593&q=pinnadel%20clipart&ved=0ahUKEwjZhtLw7IPUAhXJIJoKHbiQCo4QMwhbKDEwMQ&iact=mrc&uact=8)****⇨ Kurzeinführung MS Project.pdf**

Microsoft Project (Bestandteil der Microsoft Schüler-CD!) ist nur eine Möglichkeit, um professionelle Terminpläne zu erstellen! Welches Programm auch immer, ein Terminplan ist Pflicht!

###### - Begleitprotokolle

**⇨ Begleitprotokoll.pdf**

###### - Tätigkeitsprotokolle



**⇨ Tätigkeitsprotokoll.xlsx**

###### - Dokumentationen

An dieser Stelle sollen gegebenenfalls weitere wichtige Dokumente, wie Berechnungen, Kalkulationen, technische Dokumentationen, Zeichnungen, etc. eingefügt werden.

###### - Layouts und Verzeichnisse in Microsoft WORD

Ändern der Layout-Vorlage

Möchte man die Schriftart in der Vorlage ändern, so setzt man einen Rechtsklick auf den gewünschten Bereich in der Menüleiste und wählt die Option „Ändern“. Hier können sowohl Schriftart als auch Schriftgröße verändert werden.



Abbildung 5 Screenshot Formatvorlage ändern

Es ist jedoch darauf zu achten, dass das Layout ein einheitliches Bild ergibt, daher nicht zu viele verschiedene Schriftarten und Schriftgrößen.

Erstellen eines Inhaltsverzeichnisses

Um in Microsoft Word automatisch ein Inhaltsverzeichnis erstellen zu lassen, ist es wichtig, dass die Überschriften und der Fließtext im Dokument richtig formatiert sind. Empfehlenswert ist es, gleich zu Beginn Wert auf die richtige Formatierung zu legen, nachträglich Änderungen sind sehr zeitaufwändig.

1. Verweise ⇒ Inhaltsverzeichnis ⇒ gewünschte Vorlage auswählen
2. Änderungen: Rechtsklick auf Inhaltsverzeichnis ⇒ Felder aktualisieren ⇒ Gesamtes Verzeichnis aktualisieren

Erstellen eines Literaturverzeichnisses

Sämtliche zitierte Quellen sind in einem Literaturverzeichnis aufzulisten, das sich mit Microsoft Word aber eher einfach erstellen lässt, wie die nachfolgende Anleitung zeigt.

1. Verweise ⇒ Quellen verwalten ⇒ Neu
2. Häkchen bei „Alle Literaturverzeichnisfelder anzeigen“ setzten
3. Um den Autor einzufügen auf das Feld „Bearbeiten“ klicken
4. Gewissenhaft die unten abgebildete Eingabemaske (Abbildung 2) ausfüllen
5. Alle Quellen anlegen
6. Literaturverzeichnis ⇒ Design auswählen ⇒ Literaturverzeichnis einfügen



Abbildung 6 Screenshot Quelle erstellen

Ist kein Autor bekannt, so wird beim Namen N.N. (no name) eingegeben. Felder mit einem  sind Pflichtfelder und müssen ausgefüllt werden.

Erstellen eines Abbildungs- und Tabellenverzeichnisses

Die in der Arbeit verwendeten Abbildungen und Tabellen sind fortlaufend zu num­merieren und am Ende der Arbeit in einem Abbildungsverzeichnis bzw. Tabellen­verzeichnis mit Angabe der Seitenzahlen anzuführen. Wie bei Zitaten sind auch bei Bildern oder Tabellen die Quellen anzugeben. Diese sind im Literaturverzeichnis ebenso anzuführen.

Mit Microsoft Word kann ein Abbildungsverzeichnis erstellt werden. Voraussetz­ung ist jedoch, dass einige Kleinigkeiten bei der Formatierung der Bilder und des Textes berücksichtig werden. Grundsätzlich macht es auch Sinn, das Bild und den Bildtext miteinander zu gruppieren, da dann beide Teile miteinander verschoben oder kopiert werden können. Hier finden Sie eine Schritt-für-Schritt-Anleitung zum Einfügen und Beschriften eines Bildes:

1. Bild einfügen
2. Mausklick rechts auf das Bild ⇒ Zeilenumbruch ⇒ eine Option außer „Mit Text in Zeile“ wählen
3. Bild markieren ⇒ Verweise ⇒ Beschriftung einfügen
4. Mit dem Button „Neue Bezeichnung“ kann man z.B. festlegen, dass anstatt „Abbildung“ nur „Abb.“ verwendet wird.
5. Mit gedrückter SHIFT-Taste Bildtext + Bild markieren ⇒ (Seiten)Layout ⇒ Gruppieren ⇒ Gruppieren



Abbildung 7 Screenshot Beschriftung einfügen

Nachdem Sie den Großteil der Bilder eingefügt haben, können Sie nun ein Ab­bildungsverzeichnis erstellen. Dabei gehen Sie wie folgt vor:

1. Verweise ⇒ Abbildungsverzeichnis einfügen
2. Layout an den Rest des Textes anpassen
3. Nachträgliche Änderungen ⇒ ganzes Verzeichnis markieren ⇒ Tabelle aktualisieren ⇒ Gesamtes Verzeichnis aktualisieren

Sollten Sie im Nachhinein noch Bilder einfügen und sich die Nummerierung der Bilder nicht ändern, einfach bei der falschen Nummer das Kontextmenü öffnen und Felder aktualisieren klicken.