

Assignment 2

TypeScript App

Projekt

Registrierungsformular

10.05.2020

Alischa Thomas,
Matrikelnummer: 3019616;
DHBW Mosbach,
Frontend-Entwicklung,
Herr Slezak,
HSS 2020

Inhaltsverzeichnis

PROJEKTVORHABEN.....	1
VORKENNTNISSE UND VORBEREITUNG.....	1
DOKUMENTE UND KLASSEN	4
DOKUMENTENSTRUKTUR UND -FUNKTIONALITÄT	4
KLASSENSTRUKTUR UND -FUNKTIONALITÄT	5
<i>Klasse MethodsUsernameGenerator</i>	<i>6</i>
<i>Klasse ApplyUsernameGenerator.....</i>	<i>7</i>
<i>Klasse MethodsPasswordCheck.....</i>	<i>8</i>
<i>Klasse ApplyPasswordCheck.....</i>	<i>8</i>
PERSÖNLICHE REFLEXION.....	9
LITERATURVERZEICHNIS.....	11

Projektvorhaben

Ziel des Projektes ist die Entwicklung einer funktionalen Typescript-Applikation, die im Browser durch die Kompilierung des verwendeten JavaScript über eine *index.html* verwendet werden kann. Die Art der Applikation liegt dabei im Entscheidungsfreiraum der Autorin. Diese entschied sich für ein Registrierungsformular, das einen *Username-Generator*, sowie einen *Password-Checker* beinhaltet. Diese Applikation *form-validator* wurde aus der vom Dozierenden empfohlenen Liste mit Projekten ausgewählt (Traversy, 2020). Dieses wurde lediglich zur Ideenfindung genutzt. Es wurde sich allerdings nicht an dem darin verwendeten Code orientiert, sondern ein eigener Ansatz frei von dem Beispielprojekt entwickelt. Das Vorhaben wurde außerdem mit dem Dozierenden abgeklärt und von diesem bestätigt.

Der *Username-Generator* nutzt den im Formular von NutzerInnen zuvor eingegebenen Uservor- und nachnamen, inklusive einer zufällig erzeugten Zahl und generiert einen Usernamen. Der *Password-Checker* überprüft zwei eingegebene Passwörter im Formular auf Similarität. Die Überprüfung wird durch das Absenden des gesamten Formulars über den Registrierungsbutton ausgeführt. Ist diese erfolgreich wird ein Bestätigungstext auf der NutzerInnenoberfläche ausgegeben und das Formular abgesendet. Ist die Überprüfung nicht erfolgreich verliert der Button seine Funktion Daten zu übermitteln und teilt NutzerInnen mit, dass sie die Passworteingabe korrekt wiederholen sollten um die Registrierung abzuschließen.

Kriterien für die Entwicklung der App sind die Verwendung von TypeScript (*folgend TS*), was die Anwendung typescript-typischer Features wie z.B. Typisierung mit einschließt (*Bewertungskriterien aus Moodle*). Aus den in der Vorlesung genannten Kriterien geht hervor, dass Interfaces oder Klassen verwendet werden sollen. Des Weiteren sollen die verschiedenen TS-Dokumente mit einem Modulebundler gebündelt werden. Die Umsetzung der Kriterien folgt in den weiteren Abschnitten Vorkenntnisse und Vorbereitung, sowie Dokumente und Klassen.

Vorkenntnisse und Vorbereitung

Die Vorkenntnisse zum Thema TS beliefen sich vor Beginn des Projektes auf die theoretischen Vorlesungsinhalte, die zuvor vom Dozierenden vermittelt wurden. Da TS auf der Logik von JavaScript und Java aufbaut sind zumindest Grundkenntnisse vorhabden. Jedoch müssen die Kenntnisse zu Themen bezüglich der speziellen TS-Syntax und TS-features erweitert werden. Dazu wurde hauptsächlich ein sehr ausführliches Video-Tutorial mit zusätzlichen Übungen verwendet (*TypeScript Course for Beginners 2020*, n.d.). Zusätzlich wurde sich vor Beginn des Projektes nochmal über die Vorteile der Nutzung von TS für die Entwicklung informiert (Pramono, 2019). Da die Typisierung der hauptsächliche Vorteil ist wurde nochmal tiefer über

die in TS zu verwendenden Typen recherchiert (*Basic Types in TypeScript*, n.d.). Dadurch konnte das theoretische Wissen um einen ersten Überblick in der praktischen Anwendung erweitert werden. Dies erleichterte den Einstieg in das praktische Aufsetzen des Projektes immens und besonders das ausführliche Video-Tutorial ist sehr weiterzuempfehlen für TS-EinsteigerInnen.

Die grundlegende Struktur für das Projekt wurde sich durch Clonen des, vom Dozenten zur Verfügung gestellten, Git-Repositories auf *GitHub* angelegt (Slezak, 2020). Da es keine Vorkenntnisse im Umgang mit Formularen generell gibt wird zuerst eine allgemeine Recherche rund um das Thema *Form Validation* durchgeführt. Die Recherche zeigt, dass es verschiedene Arten, sowie unterschiedliche Herangehensweisen gibt. Es wird sich für eine *Client-side Form Validation* entschieden. Diese wird außerdem extern mit TS durchgeführt und an wenigen Stellen mit zusätzlicher *Built-In Form Validation* innerhalb des HTML-Dokumentes ergänzt (*Client-Side Form Validation by MDN*, n.d.).

Um einen ersten Überblick über das Vorhaben zu erhalten wird im ersten Schritt eine größtenteils funktionslose HTML-Datei inklusive zugehörigem CSS-Dokument erstellt. Die Kenntnisse im grundlegenden Umgang mit HTML und CSS sind an diesem Punkt des Studiums bereits verfestigt. Trotzdem bedurfte es Nachrecherchen in Bezug auf den Aufbau eines Formulars mittels Formula- und Input-Tags (*Html - Background Text in Input Type Text*, n.d.; *Change Input Placeholder Color*, n.d.; *HTML Input Required Attribute*, n.d.).

Damit die den HTML-Elementen mittels TS hinzugefügten Funktionen auch abgerufen werden können wird der *TypeScript-Compiler* im Projekt installiert. Darüber hinaus wird das Projekt mit einem *lite-server*, ein zur freien Verfügung stehender Server für Entwicklungsvorhaben, verbunden und gleichzeitig eine automatische Synchronisierung zwischen diesem und den Dateien des Projektes eingebaut. Dazu werden die entsprechenden Anpassungen in der *package.json* Datei im TS-Projekt gemacht. Dies kostet im ersten Schritt Zeit, erspart aber das händische Kompilieren und Aufrufen im Browser bei jeder Änderung während des Projektes (*TypeScript Course for Beginners 2020*, n.d.).

Da bei Bearbeitung des Projektes keine bis wenige Kenntnisse über die praktische Durchführung des *Bundlens* von Modulen vorliegt werden auch dazu weitere Informationen eingeholt (*Rollup.js Dokumentation*, n.d.; *Java Script Rollup Bundle Tutorial*, n.d.). Diese beziehen sich auf die Installation des verwendeten Modulebundler *rollup.js*, sowie die Verknüpfung von Dateien und Klassen mit Hilfe von *import* und *export* innerhalb der Dateien (*Modules in TypeScript*, n.d.). Auf Basis dessen können der gewählte Modlebundler erfolgreich eingebaut und die Module gebundled werden. Anschließend wird seine Konfiguration in der *rollup.config.js*-Datei so angepasst, dass bei jeder Kompilierung die Datei *index.ts* zu einer kombinierten *bundle.js*-Datei wird. Des Weiteren werden in der *ts.config*-Datei die *Root*- und *Source*-Ordner für die Kompilierung angegeben, sowie die Version des im Projekt

angewandten *ECMAScript* auf *ES6* eingestellt.

Mit diesen Grundeinstellungen als Basis können im nächsten Schritt die Funktionalitäten der App-elemente mit TS implementiert werden. Zur leichteren Verfolgung der weiteren Schritte dient folgender Screenshot der Applikation als Orientierung:

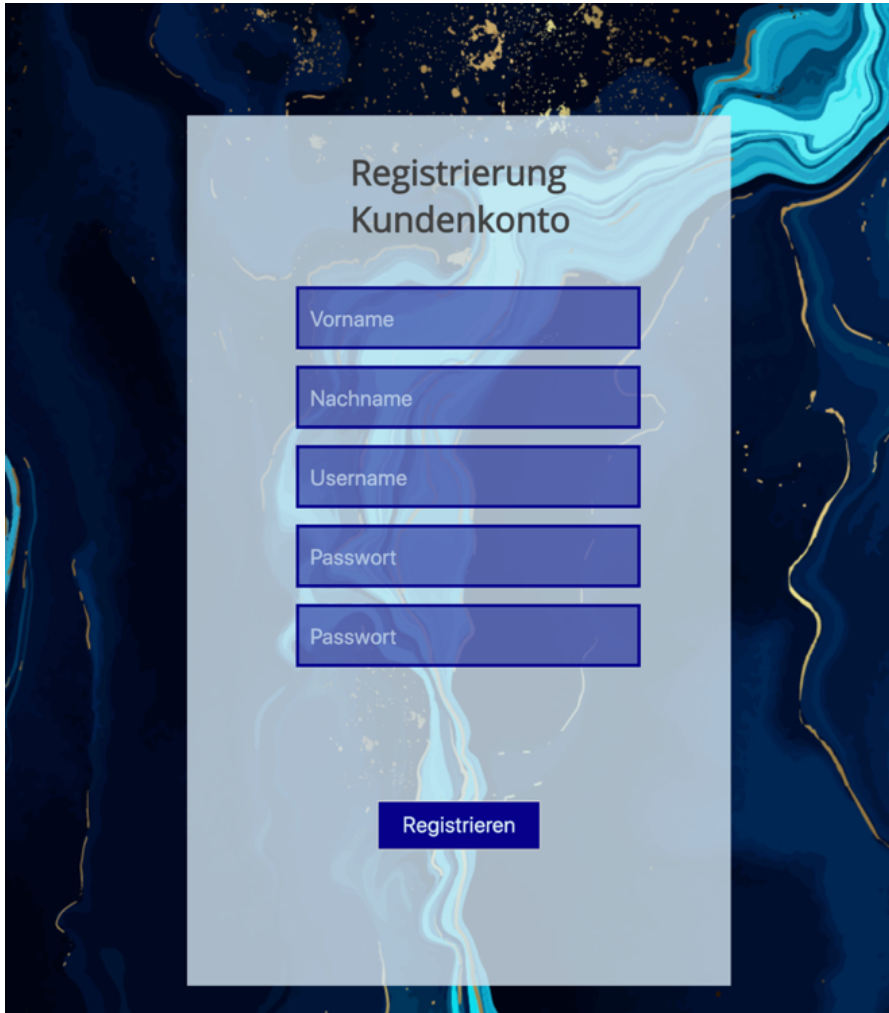
The image shows a web application interface for customer registration. It features a light blue background with a subtle, abstract pattern. The registration form is centered and consists of several input fields and a button. The fields are labeled 'Vorname', 'Nachname', 'Username', and 'Passwort' (twice). The button is labeled 'Registrieren'. The overall design is clean and modern.

Abbildung 1: Screenshot TypeScript Applikation, 09.05.2020

Dokumente und Klassen

Dokumentenstruktur und -funktionalität

Die Dokumentenstruktur ist folgendermaßen aufgebaut:

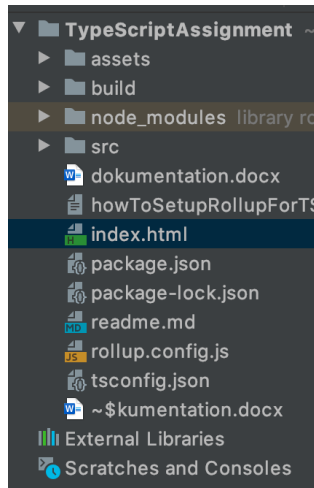


Abbildung 2: Ordnerstruktur innerhalb der IDE: oberste Ebene, Screenshot IDE 09.05.2020

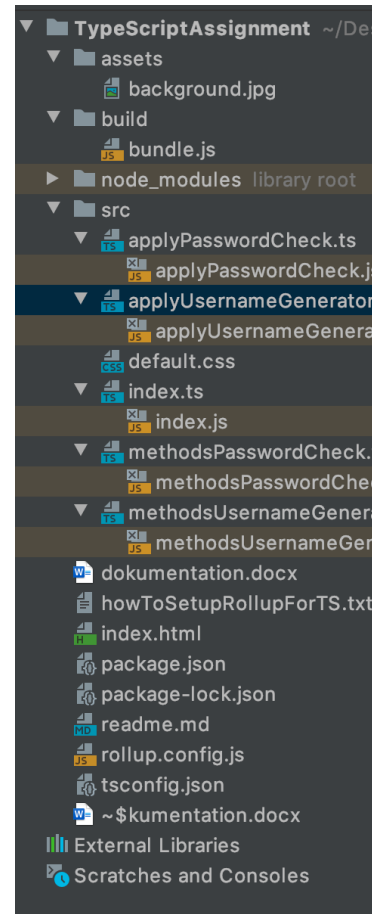


Abbildung 3: Ordnerstruktur innerhalb der IDE: gesamt, Screenshot IDE 09.05.2020

Das Projekt besteht aus 3 Ebenen: auf der niedrigsten Ebene befinden sich die kompilierten JavaScript-Dokumente der einzelnen Klassen (*src-Ordner*). Diese wurden aus den darüber, auf der zweiten Ebene, liegenden TS-Dokumenten kompiliert. Auf dieser Ebene befindet sich außerdem die *index.ts*. Diese führt alle einzelnen Module in einem TS-Dokument zusammen. Ausgehend von diesem wird auf der ebenfalls zweiten Ebene innerhalb des darüberliegenden *build-Ordners* das *gebundelte bundle.js*-Dokument erstellt. Des Weiteren befinden sich auf der zweiten Ebene weitere verwendete Inhalte wie die Dokumentation im *assets-Ordner*. Auf der oberen Ebene sind nun die übergreifenden Ordner wie *assets*, *build*, *src* angelegt. Eine Besonderheit stellt der *node-modules*-Ordner dar, der standardmäßig versteckt wird. Er beinhaltet alle *Node*-Module, die zusätzlich verwendet werden können. Dies trifft in diesem Projekt z.B. auf den *Lite-Server* zu. Da der Ordner entsprechend viele Unterordner enthält wird

er hier nur in eingeklappter Version dargestellt. Auf der obersten Ebene gipfelt dann die Kette des *Modulebundlings* in der *index.html*. In dieser ist die darunter, im *build*-Ordner liegende, *bundle.ts* mittels eines `<script>`-Tags verbunden. Somit werden hier die aus den einzelnen TS-Klassen kompilierten und *gebundenen* Dateien in die HTML-Datei, die im Browser angesteuert wird, integriert. Außerdem ist neben der Dokumentation auf der obersten Ebene eine Datei zu finden, die die Installation des TypeScript-Compilers beschreibt. Daneben sind des Weiteren die Konfigurationsdateien für den Modulebundler und den *TypeScript-Compiler* zu finden. Der *Bundleprozess* kann noch einmal bildlich im nächsten Abschnitt *Klassenfunktionalität* nachvollzogen werden (Abb. 4).

Klassenstruktur und -funktionalität

Um den Umgang mit mehreren TS-Modulen demonstrieren zu können wurden für das Projekt mehrere Klassen angelegt, die teilweise voneinander erben, also Funktionalität übernehmen können. Die TS-Klassenhierarchie baut sich folgendermaßen auf:

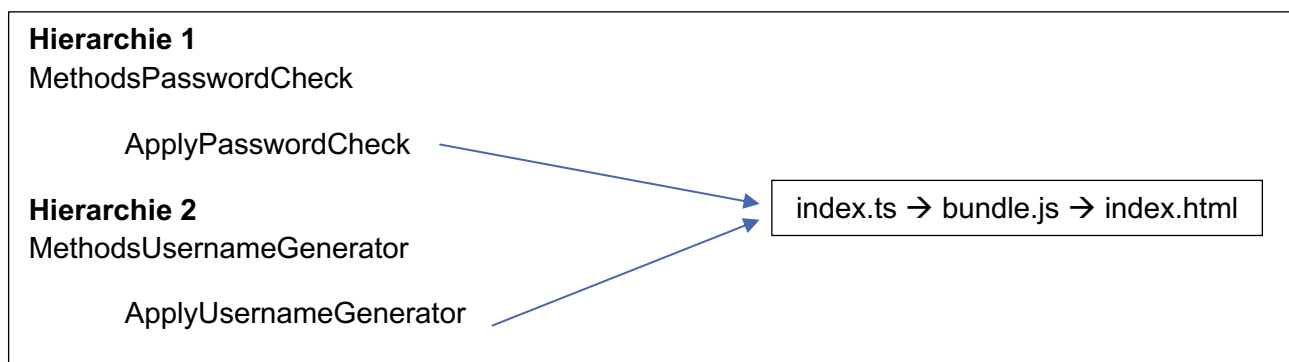


Abbildung 4: Klassenstruktur und Bundleprozess

Die *Methods*-Module legen dabei die grundlegenden Methoden fest, die an die *Apply*-Klassen weitervererbt werden. Diese führen die Methoden an HTML-Objekten aus. Die beiden Modularten werden miteinander verbunden, indem sie exportiert bzw. importiert werden (*Modules in TypeScript*, n.d.). Hierbei ist wichtig zu beachten, dass nicht die gesamte Datei importiert wird, sondern lediglich die Klasse innerhalb dieser (*Import Typescript Files*, n.d.). Der Dateienimport wird in der *index.ts* verwendet um die *Apply*-Klassen, die über Vererbung mit den *Methoden*-Klassen verbunden sind, als Dateien zu importiert. Diese *index.ts* bildet die Grundlage für das *Bundlen* in die *bundle.js*-Datei, die wiederum das HTML-Dokument im Browser speist.

Nun liegt die grundlegende Verbindung zwischen TypeScript, JavaScript und HTML. Allerdings muss der Prozess nicht nur in eine Richtung zu dem HTML-Dokument hin funktionieren, sondern auch zurück. Die HTML-Objekte müssen innerhalb der TS-Klassen angesprochen und mit Funktionen ausgestattet werden. Dazu werden in der jeweilig verwendeten Klasse erst Variablen deklariert, sowie typisiert und anschließend mit dem

entsprechenden HTML-Element durch die Methode `document.getElementById()` initialisiert (*Connect InnerHTML with JavaScript*, n.d.). Wichtig ist hierbei der verwendete Typ. Grundlegend werden in diesem Projekt die Typen *HTMLElement*, *HTMLButtonElement* und *HTMLInputElement* für die Ansprache von HTML-Objekten verwendet (*HTMLInputElement*, n.d.). Dabei wird auf eine spezielle Schreibweise bei der Initialisierung zurückgegriffen. Mit Hilfe des Ausrufezeichens `!` wird dem Compiler zugesichert, dass dieses HTML-Element in der *index.html* existiert. Ohne diese Schreibweise kam es im Verlauf dieses Projektes zu Fehlermeldungen. Das Ausrufungszeichen ist ein sogenanntes *Workaround* um diese (*TypeScript Course for Beginners 2020*, n.d.). Das HTML-Element ist mit diesen Schritten initialisiert, typisiert und liegt in der Klasse als weiter zu verwendende Variable ab.

Grundlegend besteht jede TS Klasse aus einem Konstruktor, den zu verwendenden Variablen und Methoden (Motto, n.d.). Des Weiteren wurde während der Entwicklung *Logging* zur Qualitätssicherung des Codes verwendet. Die entsprechenden Zeilen sind zur besseren Nachvollziehbarkeit für den Dozierenden auskommentiert, würden aber bei einem Einbau des Codes in ein reales Projekt aus der Arbeitswelt entfernt werden.

Klasse MethodsUsernameGenerator

Diese Klasse bezieht sich auf den Umgang mit den Eingabefelder *Vorname*, *Nachname* und *Username*. Durch *Build-In Form Validation* wurde den ersten beiden Feldern das HTML-Attribut *required* und dem letzteren *disabled* zugewiesen. Dadurch wird die NutzerInneneingabe in den ersten beiden Feldern verpflichtend, wogegen es nicht erlaubt ist etwas in das dritte Feld einzutragen.

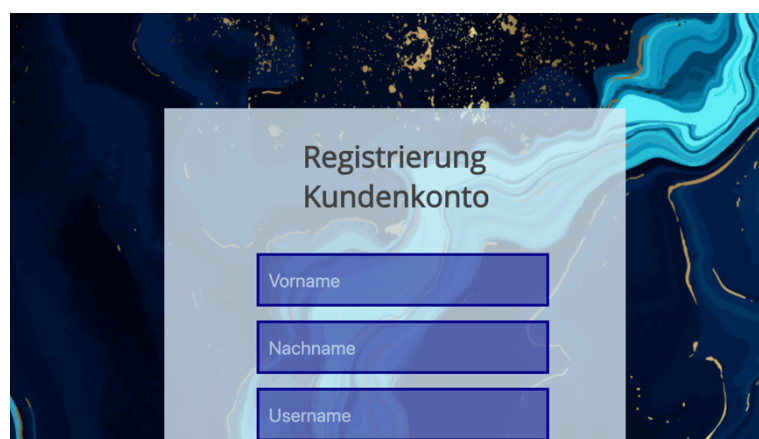
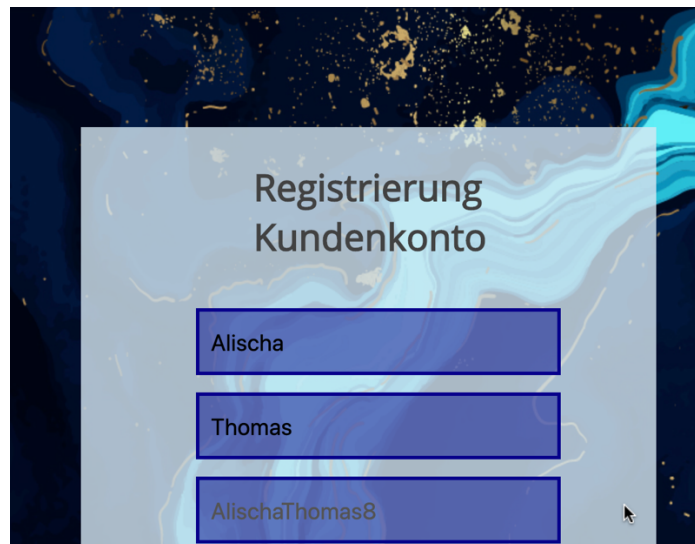


Abbildung 5: Durch Klasse MethodsUsernameGenerator bediente Felder, Screenshot TypeScript App, 09.05.2020

Im dritten Feld wird basierend auf der vorangegangenen Eingabe aus dem Vornamen, Nachnamen und einer zufällig generierten Zahl ein Username erstellt.



**Abbildung 6: generierter Username,
Screenshot TypeScript App, 09.05.2020**

Dazu wird die Methode `getRandomInt()` erstellt, die eine zufällig generierte Zahl zurückgibt (Cardillo, 2018). Diese wird innerhalb der Methode `addInput()` aufgerufen. An diese werden die beiden NutzerInnenangaben als *string* übergeben und mit der zufällig generierten Zahl kombiniert. Wurde der Username generiert, wird außerdem ein Text in der Konsole ausgesteuert, der darüber informiert, ob dieser Prozess erfolgreich war (im aktuellen Code auskommentiert/disabled). Ausgeführt werden die beiden Methoden in der erbenenden Klasse *ApplyUsernameGenerator*.

Klasse ApplyUsernameGenerator

Damit die Klasse die Methoden von *MethodsUsernameGenerator* erben kann, wird diese importiert. Anschließend werden die anzusprechenden HTML-Elemente (die ersten drei Eingabefelder) mit der Methode `document.getElementById()` Variablen zugewiesen und entsprechend typisiert. Damit der Prozess der Generierung eines Usernames angestoßen wird, wird die Ausführung der dazu nötigen Methoden an ein *blur*-Event angehängen (*HTML DOM Event Object*, n.d.). Dieses wird durch eine *EventListener()-Methode* an das zweite Eingabefeld geheftet. Verlassen NutzerInnen dieses, wird der neue Username automatisch in dem dritten Feld angezeigt. Es ist keine zusätzliche Interaktion mit der NutzerInnenoberfläche notwendig. Innerhalb des *Eventlisteners()* wird die Klasse *MethodsUsernameGenerator* instanziiert und einer Konstanten überwiesen. Dabei werden die Werte der Eingabefelder an die Methoden der Klasse weitergegeben, sowie diese aufgerufen und ausgeführt. Der Wert des dritten Eingabefeldes wird anschließend mit dem soeben generierten Username überschrieben und dadurch auf dem Screen sichtbar gemacht.

Somit kann nun mit den Methoden aus der Klasse *MethodsUsernameGenerator* und deren Anwendung auf die entsprechenden HTML-Objekte in der Klasse *ApplyUsernameGenerator* automatisch ein Username für NutzerInnen im Registrierungsprozess generiert werden.

Klasse `MethodsPasswordCheck`

Der Password-Checker ist in diesem Projekt simpel aufgebaut und vergleicht ein eingegebenes Passwort mit einem wiederholt eingegebenen auf Similarität. In der Praxis wird dieser Vorgang verwendet um Schreibfehler im Passwort bei der Registrierung zu vermeiden um NutzerInnen nach der Registrierung einen reibungslosen Login-Prozess mit ihrem gewählten Passwort zu ermöglichen.

Die Klasse besteht aus einer großen Methode `validatePassword()`. Diese greift die beiden eingegebenen Passwörter aus den HTML-Inputfeldern auf und prüft sie bei Klick auf den Registrierungsbutton. Je nach Ergebnis der Überprüfung wird eine Bestätigungsnachricht oder eine Fehlermeldung ausgegeben. Ist der Passwort-Check erfolgreich werden die Daten mit Hilfe des Buttons übermittelt, ist er fehlerhaft, findet keine Übermittlung statt bis der Fehler korrigiert wurde.

Dazu werden die drei in diesen Prozess verwickelten HTML-Objekte Eingabefeld 4 und 5, sowie der Registrierungsbutton mit der Methode `document.getElementById()` innerhalb des Konstruktors der TS-Klasse einer Variablen zugewiesen und entsprechend typisiert. Die Methode `validatePassword()` erwartet die beiden Parameter aus den Inputfeldern. Mittels einer If-Else-Bedingung werden diese beiden übergebenen Parameter auf Similarität geprüft. Ist die Bedingung (Passwörter sind ungleich) erfüllt, wird der erste Block angesteuert. Darin wird der Style, der im HTML-Dokument angelegt und im CSS gestylt, Meldungen mit `display = „block“` und `display = „none“` so angepasst, dass die Fehlermeldung erscheint und die Bestätigungsmeldung nicht sichtbar ist (*Show Error Message without Alert Box in Java Script*, n.d.; *Activate/Deactivate CSS on Button Click*, n.d.; *Show/Hide Multiple Divs Based on Input Value*, n.d.). Außerdem wird der Typ des Buttons auf `„button“` gesetzt, wodurch er die Funktion verliert Daten zu übermitteln.

Ist die Bedingung nicht erfüllt, wird der zweite Else-Block angesteuert. Dieser hat die gleiche Grundstruktur, setzt das Styling mittels CSS aber so, dass die Bestätigungsmeldung angesteuert wird und der Button den Typ `„submit“` erhält. Dadurch werden die Registrierungsdaten erfolgreich übertragen.

Klasse `ApplyPasswordCheck`

In dieser Klasse wird ähnlich zur Klasse `ApplyUsernameGenerator` vorgegangen. Die Methoden aus der Klasse `MethodsPasswordCheck` werden durch einen Import der gesamten Klasse in der Klasse `ApplyPasswordCheck` für die Anwendung zugänglich gemacht. Die drei in diesen Prozess verwickelten HTML-Objekte Eingabefeld 4 und 5, sowie der Registrierungsbutton werden Variablen zugeordnet und typisiert. Anschließend wird eine Methode `EventListener()` an den Button angeheftet, der auf dessen Klick hört. Dieser Klick löst dann die Methodenfunktionalität der Klasse `MethodsPasswordCheck` mit dem `new`-Operator

aus und führt die Überprüfung der Passwörter durch. Wichtig ist dabei, dass nicht die HTML-Objekte an sich als Parameter übergeben werden, sondern die darin enthaltenen Werte mit `.value`. Abschließend wird die Klasse selbst nochmal mit dem `new`-Operator aufgerufen damit die Inhalte der Klasse ausgeführt werden. Entsprechend werden dann die Bestätigungs- bzw. Fehlermeldungen angesteuert und die Datenübermittlung angestoßen oder zurückgehalten wie in Klasse *MethodsPasswordCheck* beschrieben.

Somit liegt nun mit den Methoden aus der Klasse *MethodsPasswordCheck* und deren Anwendung auf die entsprechenden HTML-Elemente in der Klasse *ApplyPasswordCheck* ein valider Passwordcheck entsprechend des Kriteriums der Similarität vor.

Persönliche Reflexion

Gleich beim Einstieg in das Projekt verbrachte ich viel Zeit damit richtig nachzuvollziehen wie das *Bundling* funktioniert und wie ich mit den Konfigurationsdateien umgehen kann. Ohne diese Punkte geklärt zu haben, fiel es mir schwer mit dem Projekt zu starten. Davor war der ganze Prozess wie ein Black Box, in der die Magic passiert, wenn ich einen Befehl eingebe. Durch gründliche Recherche passiert dort immernoch die Magic. Nun weiß ich aber wie ich deren Richtung kontrollieren kann.

Die Erstellung der Klassenstruktur erfolgte nicht intuitiv und benötigte einige Iterationen. Es fiel besonders schwer Klassen für eine solch kleine Applikation zu bauen. Nach Rücksprache mit meinem Betreuer aus dem Ausbildungsbetrieb wurde die Klassenstruktur, so wie sie im Projekt vorhanden ist, letzten Endes auch angelegt. Allerdings konnten wir beide und nach Rückmeldung aus dem Studiengang auch viele andere die Idee dieser „künstlich erzeugten“ Zerstückelung in Module nicht ganz nachvollziehen. Bis zum Ende des Assignments wurde mir der Mehrwert dessen in einem so kleinen Projekt nicht ganz bewusst. Die Idee ist klar, dass der generelle Umgang mit dem Modulebundler und Klassen erlernt werden soll. Allerdings war es eine Herausforderung Code in einzelne Teile zu zerlegen, die ich intuitiv nicht zerlegt hätte.

Zu Beginn war mir ebenfalls nicht ganz klar wie ich anfangen sollte. Der häufigste Satz, den ich während dieses Assignments als Tipp gehört habe war „das ist eigentlich JavaScript, nur typisiert“. Dies trifft größtenteils zu bis es zu den 100 Ausnahmen kommt, die es in TypeScript gibt durch die spezielle Codestruktur. Somit versuchte ich meine Codeteile in JavaScript vorzuschreiben und dann in TypeScript durch Typisierung zu erweitern, woran ich aufgrund verschiedenster Empfindlichkeiten von TypeScript scheiterte. Letzten Endes halfen kleinere private Tutoriumsstunden mit einem Kollegen aus der Entwicklung meines Ausbildungsbetriebes. Dieser zeigte mir Beispiele für klassische TypeScript-Syntax und half mir durch Reviews und Remotetipps beim Einstieg in das Projekt. Denn oftmals lag es nicht an der Funktionalität, sondern an der Syntax. Diese Problematik zu googlen ist schwierig und

wurde durch den Erfahrungswert meines Tutors wettgemacht. Es ist zu betonen, dass dieser ausschließlich eine beratende Funktion tätigte.

Neben den Syntaxproblemen war mir zu Beginn ebenfalls schleierhaft klar wie ich die einzelnen Dokumente miteinander verknüpfen und z.B. HTML-Elemente erfolgreich typisieren konnte. Speziell zu diesem Punkt war mir völlig unbekannt, dass es spezifische Typen für HTML-Elemente gibt bis ich zufällig in einem Online-Tutorial darauf stieß. Darüber hinaus verbrachte ich hauptsächlich die Zeit damit kleine TypeScript Wehwehchen nachzuvollziehen und zu beheben, anstatt mich mit meiner grundlegenden Codefunktionalität auseinanderzusetzen.

So komplex und herausfordernd es in diesem Projekt war den Überblick über all die Dokumente und deren Funktionen zu behalten und Fehler darin lokalisieren und beheben zu können, so hat es doch meine bisher erlernten Java- und Frontend-Kenntnisse zu HTML, CSS und JavaScript vereint und alles in ein umfassenderes Big Picture gesetzt. Ich habe durch Code Reviews gelernt wie ich meinen Code kürzen und weiter verbessern kann. Darüber hinaus habe ich gelernt, dass man längst nicht alles googlen kann, sondern viele Kenntnisse in der Entwicklung auf Geduld und Erfahrung basieren. Daher finde ich es umso ärgerlicher, dass für diese Veranstaltung so viele verschiedene Aspekte des Frontends angesprochen wurden statt darauf zu setzen die Kenntnisse punktuell zu vertiefen durch wiederholte Übungen. Die großen Assignments wie dieses haben mein Zeitmanagement vorangebracht, jedoch wurde auch viel wieder durch fehlende Wiederholung verwirkt. Das zeigt mir noch mehr wie wichtig es ist die komplexen Aufgaben aus diesem Kurs zuhause vereinzelt zu wiederholen und die Thematik für mich selbst, was die praktische Anwendung betrifft, nochmals weiter aufzusplittern.

Insgesamt wurde mein übergreifendes Wissen durch dieses Assignment ausgeweitet. Allerdings fühle ich mich nicht in der Lage dazu selbständig TypeScript Aufträge umsetzen zu können.

Literaturverzeichnis

Activate/deactivate CSS on button click. (n.d.). Stack Overflow. Retrieved May 8, 2020, from

<https://stackoverflow.com/questions/38639604/how-do-i-activate-deactivate-css-on-button-click>

Basic Types in TypeScript. (n.d.). Retrieved May 8, 2020, from

<https://www.typescriptlang.org/docs/handbook/basic-types.html#number>

Cardillo, J. (2018, August 28). *Using Math.random() in JavaScript.* Medium.

<https://medium.com/@josephcardillo/using-math-random-in-javascript-c49eff920b11>

Change Input Placeholder Color. (n.d.). Retrieved May 8, 2020, from

https://www.w3schools.com/howto/howto_css_placeholder.asp

Client-side form validation by MDN. (n.d.). MDN Web Docs. Retrieved May 8, 2020, from

https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation

Connect innerHTML with JavaScript. (n.d.). Stack Overflow. Retrieved May 8, 2020, from

<https://stackoverflow.com/questions/7674194/how-to-get-innerhtml-of-this-element-in-javascript>

HTML DOM Event Object. (n.d.). Retrieved May 8, 2020, from

https://www.w3schools.com/jsref/dom_obj_event.asp

HTML input required Attribute. (n.d.). Retrieved May 8, 2020, from

https://www.w3schools.com/tags/att_input_required.asp

html—Background text in input type text. (n.d.). Stack Overflow. Retrieved May 8, 2020,

from <https://stackoverflow.com/questions/13071791/background-text-in-input-type-text>

HTMLInputElement. (n.d.). MDN Web Docs. Retrieved May 8, 2020, from

<https://developer.mozilla.org/en-US/docs/Web/API/HTMLInputElement>

Import typescript files. (n.d.). Stack Overflow. Retrieved May 8, 2020, from <https://stackoverflow.com/questions/12930049/how-do-i-import-other-typescript-files>

Java Script Rollup Bundle Tutorial. (n.d.). Retrieved May 8, 2020, from <https://learnwithjason.dev/blog/learn-rollup-js/>

Modules in TypeScript. (n.d.). Retrieved May 8, 2020, from <https://www.typescriptlang.org/docs/handbook/modules.html>

Motto, T. (n.d.). *TypeScript Classes and Constructors.* Retrieved May 8, 2020, from <https://ultimatecourses.com/blog/typescript-classes-and-constructors>

Pramono, D. (2019, October 16). *Validate JavaScript Object Better with TypeScript.* Medium. <https://medium.com/@djoepramono/how-to-validate-javascript-object-better-with-typescript-e43314d97f9c>

Rollup.js Dokumentation. (n.d.). Retrieved May 8, 2020, from <https://rollupjs.org/guide/en/>

Show error message without alert box in Java Script. (n.d.). Stack Overflow. Retrieved May 8, 2020, from <https://stackoverflow.com/questions/14702190/to-show-error-message-without-alert-box-in-java-script>

Show/hide multiple divs based on input value. (n.d.). Stack Overflow. Retrieved May 8, 2020, from <https://stackoverflow.com/questions/34110766/how-to-show-hide-multiple-divs-based-on-input-value>

Slezak, J. (2020). *MysterieDev/TypeScriptAssignment* [TypeScript]. <https://github.com/MysterieDev/TypeScriptAssignment> (Original work published 2020)

Traversy, B. (2020). *Bradtraversy/vanillawebprojects* [JavaScript]. <https://github.com/bradtraversy/vanillawebprojects> (Original work published 2020)

TypeScript Course for Beginners 2020. (n.d.). Retrieved May 8, 2020, from <https://www.youtube.com/watch?v=BwuLxPH8IDs&feature=youtu.be>