Implementierung einer technischen Unterstützung der Organisation des Berufsinformationstags.

Facharbeit von David Kopczynski
Jahrgangsstufe 12 – 2020

Kurzfassung

Die Facharbeit der „Implementierung einer technischen Unterstützung der Organisation des Berufsinformationstags“ befasst sich mit einem Programm für die Sortierung der Schüler des Berufsinformationstags, wobei der Kern der Arbeit aus der Beschreibung der Anwendung und wichtigen Teilen des Quellcodes besteht. Dabei wird ebenfalls die Entwicklungsumgebung mit Electron nähergebracht, wobei diese unmittelbar mit dem Programm in Verbindung steht.

Das Programm ist in der Lage, aus einer Liste von Berufen und dazu passenden Schülerwünschen, eine Einteilung und Umsortierungen vorzunehmen. So können die Kursgrößen untereinander ausgeglichen, diese Kurse bei Bedarf komplett geleert oder nach Minimal- bzw. Maximalwerten umsortiert werden.

Zum Zeitpunkt der Erstellung ist das Programm auf GitHub unter dem Link: https://github.com/David-Kopczynski/Facharbeit aufzufinden, wobei für die Benutzung unter dem Ordner *dist* und dem persönlichen Betriebssystem, eine ausführbare Datei und zuzügliche, wichtige Programmdaten lokalisiert sind. Diese können heruntergeladen und schließlich benutzt werden, wobei Beispieldaten anbei liegen.

Inhaltsangabe

Einleitung

Meine Facharbeit der „Implementierung einer technischen Unterstützung der Organisation des Berufsinformationstags“ sollte, wie es der Name schon indiziert, die Sortierung und Umverteilung von Schülern in ihren gewünschten Kursen der Berufsinformation, einer Veranstaltung für Schüler über jegliche Berufsbereiche, welche sich über den ganzen Tag erstreckt, auf dem MPG vereinfachen. Dabei werden die Schülerwünsche an Berufen von den Lehrern eingesammelt, wobei teilweise Kurse ihre Maximalkapazitäten überschreiten und ausgeglichen werden müssen, dies jedoch ohne unnötige Freistunden zu kreieren.

Dafür sollte ein Programm gegeben sein, welches anhand von Excel-Dateien zunächst die Schüler in ihre Kurse einteilt, die Gesamtzahlen dieser ausgibt und gegebenenfalls mithilfe von Benutzereingaben die Kurse in ihren Größen ausgeglichen bzw. bei zu geringem Bedarf komplett geleert werden. Letzten Endes sollte nun eine Datei erstellt werden, welche die angepassten Kurse erneut an die Lehrer ausgibt, damit diese die Termine an die Schüler weitergeben können.

Zielsetzung

Zunächst hatte ich eine Entgegennahme der Daten mithilfe einer eigenen Webseite geplant, welche anhand von einmaligen Schlüsseln (private Keys) die Identifikation von Schülern ermöglicht und somit verifizierte Wünsche in eine große Liste für den weiteren Verlauf gespeichert werden. Dies würde zwar den Lehrern die Arbeit einer eigenen Entgegennahme dieser Wünsche abnehmen, jedoch trotzdem das Problem mit sich bringen, dass die Schlüssel an die Schüler weitergegeben werden müssten und einige Schüler womöglich keine oder verspätete Wünsche abgeben würden und diese somit fortlaufende Komplikationen verursachen. Deswegen wurde mir geraten, diese Facharbeit nur aus dem folgenden Kern zu konzipieren und die Entgegennahme wie in den letzten Jahren, über den Lehrern und Auswahlzetteln, zu belassen.

Als geplanten zweiten Teil meiner Facharbeit, war nun das tatsächliche Sortierprogramm bedacht. Dieses sollte Daten sowohl zuordnen, aber auch umverteilen können, wobei ich mir zunächst nicht allzu viele Gedanken über den Sortieralgorithmus gemacht habe. Mir war für den Anfang nur die Umsetzung meines Programmes in Electron von großer Bedeutung, da ich anhand meiner Vorkenntnisse in JavaScript und des einfaches Designs mithilfe von HTML mit CSS, einen großen Teil an Arbeit abnehmen konnte und somit im späteren Verlauf mehr Zeit für die tatsächliche Fähigkeit des Programms verbleibt. So wurde mir schnell klar, dass ich Daten anhand von Excel annehmen muss, diese sowohl in Angebot an Berufen und Schülerwünsche aufteilen und schließlich eine eigene Oberfläche für die Sortierung und deren Einstellungen anbieten sollte.

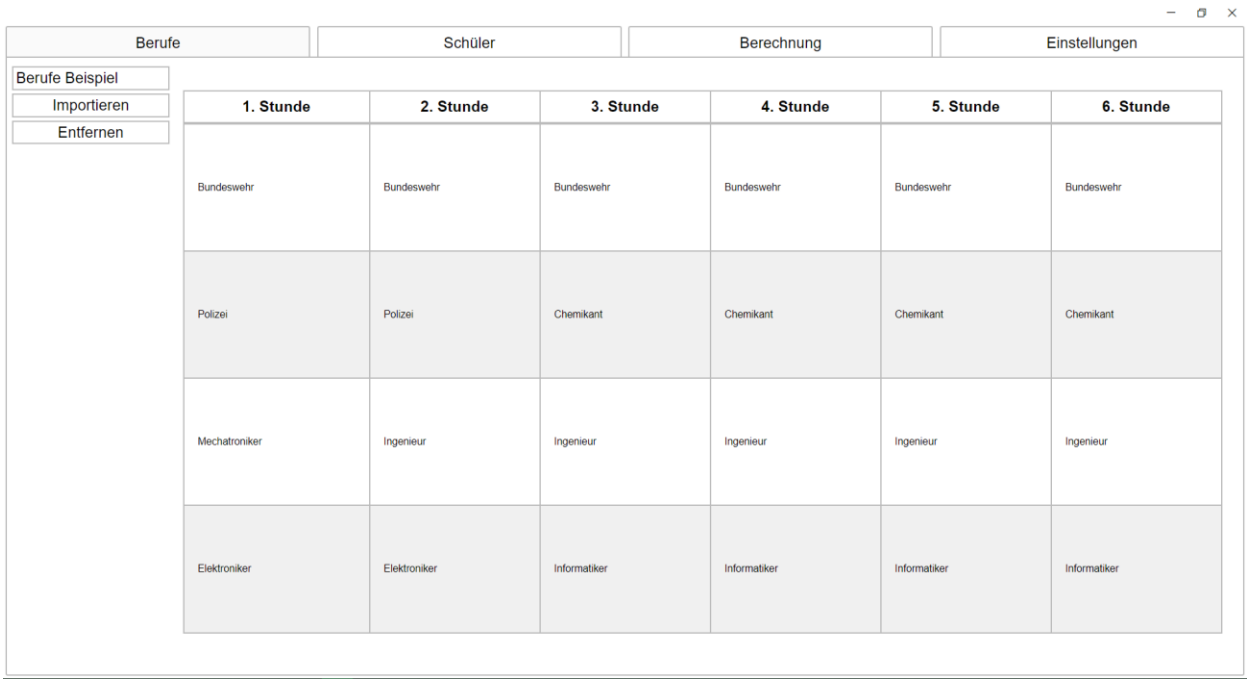
Umsetzung

Da wie beschrieben eine eigene Webseite und deren Verwaltung zu aufwendig sei, habe ich mich ausschließlich mit meinem zweiten Teil der Facharbeit befasst. Dieser ist insofern ausgearbeitet, dass auf verschiedenen Karteireitern die einzelnen besprochenen Oberpunkte wie *Berufe*, *Wünsche* und *Berechnung*, aber auch die *Einstellungen* vertreten sind. Nachdem nun alle wichtigen Eingaben getätigt wurden, die angegebenen Pfade korrekt sind und auch keine Kompatibilitätsfehler zwischen den Wünschen und den angebotenen Berufen bestehen, so kann anhand der gegebenen Excel-Dateien, eine neue Tabelle mit den Schüleranzahlen in den einzelnen Kursen erzeugt werden. Diese bietet die Möglichkeit nach gewünschten Sortiermöglichkeiten, wie Ausgleichungen mit oder ohne Minimal- bzw. Maximalwerten, aber auch manuellen Kursleerungen auf Knopfdruck, die Schüler zu verteilen. Zu beachten ist noch, dass bei der Sortierung zunächst nur versucht wird, anhand von Zeitänderungen eine bessere Kursgroße zu erreichen und somit bei der Umverteilung nur eine Zeitabweichung, aber keine Wunschabweichung, der einzelnen Schüler geschieht. Diese treten erst bei der manuellen Kursleerung ein, da durch jegliche Widersprüche der Schüler, sei es durch ungleichmäßige Verteilungen oder Kettenreaktionen durch die Sortierung, nicht immer ein perfekter Zeitplan möglich ist. Dafür wurde zusätzlich dem Benutzer ein Interface gegeben, um anhand von

prozentualen Angaben die Zeit- bzw. Wunschabweichung, durch die Sortierung, mitzuteilen. Am Ende kann eine neue Excel-Datei exportiert werden, welche die angepassten Wünsche beinhaltet.

Mustervorgang

Im Folgenden werde ich visualisiert den generischen Vorgang einer beispielhaften Sortierung, mit allen Features, demonstrierten. Außerdem werden potenzielle Fehlermeldungen erklärt und allgemein der Ablauf der Sortierung mit meinem Programm etwas nähergebracht.



5.1 Programm

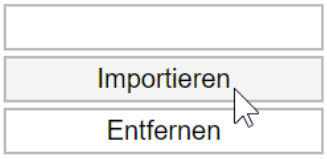
5.1 Programm zeigt wie das Programm starten würde, wobei in den nächsten Kapiteln die Datenerfassung und Validation erklärt wird.

Datenerfassung

Zunächst werden alle benötigten Dateien über die Karteireiter *Berufe* und *Schüler* eingespeist. Dabei ist es möglich, mithilfe der *Importieren*-Buttons, jeweils für die Berufe und für die Schüler, eigene Tabellen zu erstellen und auch im Nachhinein, selbst nach einem Neustart des Programmes, zwischen verschiedenen geladenen Dateien zu wechseln bzw. auf diese zurückzugreifen.

6.1 Karteireiter

Dafür ist, wie in Abbildung *6.1 Karteireiter* zu sehen, ein Menü gegeben, welches dem Benutzer die Möglichkeit bietet, zwischen den *Berufen*, *Schülern*, der *Berechnung* und den *Einstellungen* zu wechseln und somit einzelne eigene Fenster zu bedienen.



6.2 Dateiauswahl

In den Unterpunkten Berufe und Schüler sind, wie bereits angesprochen und in *Abbildung 6.2 Dateiauswahl* zu erkennen, sowohl Knöpfe für das Importieren von Excel-Dateien, aber auch der Löschung dieser gegeben. Sollte nun eine Anzahl an Dateien vorliegen, wird die momentan verwendete Excel-Datei mit ihrem Namen in der oberen Box angegeben, wobei bei Knopfdruck dieses Feldes, ein Dropdown-Menu geöffnet wird und somit eine Auswahl der gewünschten Datei möglich ist.

Datenvalidität

Damit das Programm ohne Probleme funktionieren kann, wird ein bestimmtes Dateiformat benötigt, wobei auch unter den Berufen und Schülerdaten keinerlei Tippfehler auftreten dürfen.

	A	B	C	D	E	F
1	1. Stunde	2. Stunde	3. Stunde	4. Stunde	5. Stunde	6. Stunde
2	Bundeswehr	Bundeswehr	Bundeswehr	Bundeswehr	Bundeswehr	Bundeswehr
3	Polizei	Polizei	Chemikant	Chemikant	Chemikant	Chemikant
4	Mechatroniker	Ingenieur	Ingenieur	Ingenieur	Ingenieur	Ingenieur
5	Elektroniker	Elektroniker	Informatiker	Informatiker	Informatiker	Informatiker

7.1 Excel-Datei Berufe

	A	B	C	D	E	F	G
1		1. Stunde	2. Stunde	3. Stunde	4. Stunde	5. Stunde	6. Stunde
2	students0	Polizei	Bundeswehr	Chemikant	Chemikant	Informatiker	Chemikant
3	students1	Bundeswehr	Elektroniker	Informatiker	Chemikant	Ingenieur	Bundeswehr
4	students2	Polizei	Bundeswehr	Informatiker	Ingenieur	Informatiker	Bundeswehr
5	students3	Mechatroniker	Polizei	Bundeswehr	Chemikant	Chemikant	Chemikant
6	students4	Polizei	Ingenieur	Ingenieur	Informatiker	Chemikant	Bundeswehr

7.2 Excel-Datei Schüler

Wie in *7.1 Excel-Datei Berufe* und *7.2 Excel-Datei Schüler* dargestellt, braucht das Programm einen einheitlichen Aufbau der Daten, sodass in der oberen Spalte die Uhrzeiten und in den folgespalten, abhängig von der Excel-Datei, die Schüler und Berufe erfasst werden. Allgemein lautet der Aufbau für die Berufe, dass in den folgespalten die Angebote in Abhängigkeit der, in der obersten Spalte genannten, Uhrzeit gelistet werden, wobei bei den Schülern links der einzigartige Name des Schülers und in den folgenden Reihen dessen Wünsche in Abhängigkeit der Uhrzeiten stehen. Sollte trotzdem während dem Ladevorgang ein Fehler auffallen, so werden diese Meldungen im Folgenden anstatt der Tabelle ausgegeben und müssen zunächst abgearbeitet werden.

Fehler: Der Pfad **'F:\Facharbeit_Berufsinformation\error.xlsx'** konnte nicht gefunden werden oder ist ungültig.

8.1 Fehlermeldung Dateiimport

Beispielhaft können fehlerhafte Dateiformate umbenannt und importiert bzw. einstig korrekte Daten gelöscht und somit nicht mehr vom Programm auffindbar gemacht werden. Dabei wird der in *8.1 Fehlermeldung Dateiimport* demonstrierte Fehlertext ausgegeben, wodurch die Untersuchung des Fehlers, mithilfe des Pfades der Datei, erleichtert wird. Dabei müsste die Datei wiederhergestellt oder mit Excel repariert werden, da ansonsten ein auslesen und somit dessen Verarbeitung nicht möglich ist.

1. Stunde	2. Stunde	3. Stunde	4. Stunde	5. Stunde	6. Stunde
Bundeswehr	Bundeswehr	Bundeswehr	Bundeswehr	Bundeswehr	Bundeswehr
Polizei	Polizei	Chemikant	Chemikant	Chemikant	Chemikant
Mechatroniker	Ingenieur	Ingenieur	Ingenieur	Ingenieur	Ingenieur
Elektroniker	Elektroniker	Informatiker	Informatiker	Informatiker	Informatiker

8.2 Berufstabelle

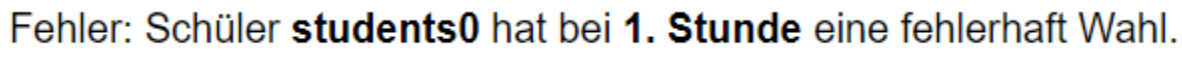
	1. Stunde	2. Stunde	3. Stunde	4. Stunde	5. Stunde	6. Stunde
students0	Polizei	Bundeswehr	Chemikant	Chemikant	Informatiker	Chemikant
students1	Bundeswehr	Elektroniker	Informatiker	Chemikant	Ingenieur	Bundeswehr
students2	Polizei	Bundeswehr	Informatiker	Ingenieur	Informatiker	Bundeswehr
students3	Mechatroniker	Polizei	Bundeswehr	Chemikant	Chemikant	Chemikant

8.3 Schülertabelle

Sind nun sowohl die Schüler- als auch Berufsdaten erfolgreich importiert, so werden diese, wie in den Abbildungen *8.2 Berufstabelle* und *8.3 Schülertabelle*, in ihren Tabellen dargestellt und können gegebenenfalls manuell überprüft werden. Anzumerken ist hier noch, dass bei einer zu

geringen Fenstergröße bis zu zwei Schiebebalken rechts und unter der Tabelle erscheinen können und somit ein einfaches Manövrieren möglich wird.

Im nächsten und auch letzten Schritt der Verifizierung, werden unter dem Reiter *Berechnung* die Berufe- und Schülerdaten verglichen, wobei geprüft wird, ob die einzelnen Wünsche zu den Uhrzeiten angeboten werden bzw. der allgemeine Aufbau übereinstimmt. Dafür muss auf den Knopf *Berechnen* in *10.1 Berechnung Menü* gedrückt werden, auf welchen im Folgenden noch genauer eingegangen wird.

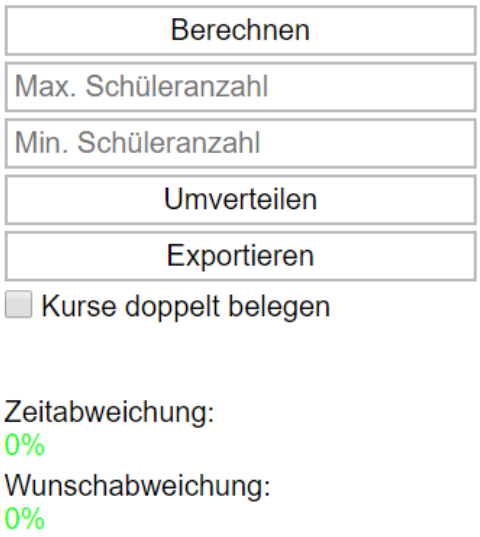


9.1 Fehlermeldung Schülerwunsch

Sollte unerwarteterweise ein Beruf nicht stimmig sein oder auch nur einen Tippfehler aufweisen, so werden diese einzelnen Fehler, wie in *9.1 Fehlermeldung Schülerwunsch*, aufgezeigt und müssen manuell in den Excel-Dateien korrigiert werden. Dabei wird als Hilfestellung der Name des fehlerhaften Schülers, als auch dessen Uhrzeit angegeben, um erneut die Korrektur für den Benutzer zu erleichtern. Außerdem werden bei mehreren Fehlern, diese untereinander gelistet, sodass nicht andauernd das Programm auf neue Fehler prüfen muss.

Datenverarbeitung

Im Karteireiter *Berechnung* können schließlich die Wünsche, abhängig von dem Benutzer, umverteilt werden, wobei zusätzlich ein weiteres Menü gegeben ist.



10.1 Berechnung Menü

In dem Menü *10.1 Berechnung Menü*, unter dem Reiter *Berechnung*, können nun im ersten Schritt die Mini- und Maximalwerte der Kursgrößen angegeben werden, wobei man diese auch noch im folgenden Verlauf anpassen kann. Trotz allem sollten diese nicht allzu strenge Werte annehmen, da sie einen größeren Einfluss auf die Berechnung haben und somit teils zu starken Abweichungen führen können.

1. Stunde	2. Stunde	3. Stunde	4. Stunde	5. Stunde	6. Stunde
Bundeswehr 8	Bundeswehr 4	Bundeswehr 7	Bundeswehr 4	Bundeswehr 3	Bundeswehr 7
Polizei 7	Polizei 8	Chemikant 11	Chemikant 12	Chemikant 9	Chemikant 11
Mechatroniker 7	Ingenieur 7	Ingenieur 8	Ingenieur 8	Ingenieur 10	Ingenieur 5
Elektroniker 8	Elektroniker 11	Informatiker 4	Informatiker 6	Informatiker 8	Informatiker 7

10.2 Ausgewertete Tabelle

Sind nun keine weiteren Fehler zwischen den Berufs- und Schülerdaten aufzufinden, so wird eine neue Tabelle, wie *10.2 Ausgewertete Tabelle*, generiert, welche zusätzlich zu der in den Berufen vorliegenden Tabelle, die Anzahl der Schüler ausgibt. In diesem Beispiel wurde die

Tabelle mit dem Button *Berechnen* generiert, wobei die Minimalgröße auf *5* und die Maximalgröße auf *10* gestellt wurde. Aus diesem Grund haben sich auch die Farben der Anzahlen angepasst, um dem Benutzer ein schnelles Feedback der vorliegenden Kursgrößen zu ermöglichen. Somit soll grün eine passende Größe, gelb Randkurse und die Farbe Rot über- oder unterfüllte Kurse markieren, welche im Folgenden mit dem Knopf *Umverteilen* in mehr oder weniger passende Kurse umverteilt werden. Hier ist noch zu beachten, dass durch einen Multiplikator, die Randkurse teilweise den Maximalwert überschreiten können, da in den meisten Fällen strikte, nicht zu überschreitende, Werte der Minimal- bzw. Maximaleingaben nicht sinnvoll sind.

Anfänglich kann mit der Umverteilung, in Abhängigkeit der Maximal- und Minimalwerte gearbeitet werden, da diese lediglich die Uhrzeit, aber nicht die Wünsche der Schüler verändern. Dabei wird auffallen, dass die *Zeitabweichung* im *Menü 10.1 Berechnung Menü* höhere Werte annehmen kann oder auch in eine alternierende, also abwechselnd auf- und absteigende, Folge gerät. In diesen Fällen ist es teilweise nicht möglich mit den gegebenen Werten eine perfekte Umverteilung zu ermöglichen, da durch dutzende grundlegende Widersprüche der Schüler, nicht immer die Gegebenheiten passend sind, wodurch über- oder unterfüllte Kurse nicht mehr mit ihrer Zeitverteilung angepasst werden können, sondern per Knopfdruck die Wünsche von manchen Schülern verändert werden müssen.

11.1 Kursleerung ohne doppelte Belegung

So wird bei dem ersten Knopfdruck, sofern dieser den Wert übersteigt, der Kurs bis zu dem gegebenen Maximalwert geleert und schließlich bei dem zweiten Knopfdruck komplett von allen Schülern befreit. Dies ist Schematisch in *11.1 Kursleerung ohne doppelte Belegung* dargestellt, wobei der Kurs hier bereits beim ersten Klick von allen Schülern geleert wird.

12.1 Kursleerung mit doppelter Belegung

Jedoch kann es vorkommen, dass wie in diesem Beispiel *12.1 Kursleerung mit doppelter Belegung*, ein Schüler bereits alle anderen Kurse belegt hat, wodurch dieser entweder im späteren Verlauf gar nicht in einem Beruf anwesend sein muss oder bei Wunsch des Benutzers, einen Kurs doppelt belegt. Dafür kann der Haken in *Kurse doppelt belegen* gesetzt und erneut auf den Kurs geklickt werden, wobei dadurch der Schüler denselben Kurs zu zwei verschiedenen Zeiten besucht. Alternativ ist es auch möglich den Haken nicht zu setzten und mit der Taste *Steuerung* und zugleich *Linksklick* auf den Kurs, allen präsenten Schülern eine Freistunde einzutragen. Schließlich ist der Kurs auf der Schüleranzahl *0* und dieser auch für weitere Umverteilungen gesperrt.

Spätestens an diesem Zeitpunkt wird auffallen, dass auch die *Wunschabweichung* steigt, weshalb mit diesem Feature mit Bedacht umgegangen werden sollte. Dieses sollte eher bei Notfällen benutzt werden, weshalb das Programm diesen Schritt nicht automatisiert vornimmt.

Datenausgabe

Sollte nun alles passend und gegebenenfalls mit verschiedenen Ansätzen sortiert sein, kann die neue Tabelle unter dem Menü *10.1 Berechnung Menü* mithilfe des Knopfes *Exportieren* an einem beliebigen Ort gespeichert werden. Dabei wird automatisch eine Excel-Datei erstellt, wobei nur ein neuer Name für die Datei angegeben werden muss.

Da diese neue Datei denselben Aufbau, wie die ursprünglichen Schülerdaten besitzt, kann diese bei Bedarf erneut mit dem Programm sortiert werden. Dafür ist lediglich ein Import nötig, wodurch verschiedene Sortieriterationen angefertigt werden können.

Das Programm

Nachdem ich die Benutzung des Programms ausführlich dokumentiert habe, werde ich im Folgenden auf die benötigte Software und dessen Nutzen eingehen. Dabei wird auf Node.js bzw. NPM, als auch Electron, die Kernstücke meines Programms eingegangen, um anhand dieses Vorwissens vereinfacht die HTML, dessen CSS und im genaueren den Sortieralgorithmus darzustellen.

NPM

Der „Node package manager“ oder auch kurz NPM, ist ein Package-Manager, welcher dem Benutzer die Möglichkeit bietet, wie es auch nativ unter Linux möglich ist, Module für bestimmte Zwecke herunterzuladen und diese für verschiedene Projekte zu nutzen. Da jegliche Pakete Open-Source sind oder auch erworben werden können, ist eine Nutzung für kommerzielle Firmen möglich, was einfach an dem Beispiel von Discord gezeigt werden kann und somit Programmierern viel Spielraum für Entwicklungen bietet.

Dieses Modul kann einfach mit Node.js installiert werden, wobei NPM die Grundlage für Electron und somit meines Projektes geboten hat.

Electron

Bei Electron hält es sich um ein Packet, der in der NPM besprochenen Library für JavaScript-Applikationen, wobei diese mit *npm install electron* dem momentanen Verzeichnis hinzugefügt und für die Benutzung vorbereitet werden kann. Sollte die *package.json* passend eingerichtet sein, so kann mithilfe der *main.js* und der *index.html*, das erste primitive Fenster unter *npm start* gestartet werden. Dabei wird eine neue Chromium-Instanz gestartet, welche zudem durch Node.js ausführliche Schreib- bzw. Leserechte besitzt und mit dem System interagieren kann. So fungiert der eigene Computer ähnlich wie ein lokaler Server, wodurch auch weitere Skriptdaten geladen und somit die Programmierung beginnen kann. Diese stützt sich hauptsächlich auf JavaScript, wobei jedoch noch die HTML und CSS-Dateien für das Design und die Benutzeroberfläche benötigt werden. Auch liefert Electron ein paar Eigenheiten, welche

schnell angeeignet und mit allerlei Vorteilen gespickt sind. So ist es z.B. möglich mit einem *dialog*-Statement ein Dialogfenster für die Speicherung bzw. dem Ladevorgang von Daten zu ermöglichen, ohne dabei ein aufwendiges Menü entwickeln zu müssen.

Da Electron nur in Abhängigkeit von anderen Libraries installiert werden kann, wird in den Quellen nicht die gesamte Liste aller Packages, sondern nur Electron und einzelne zusätzlich, von mir direkt benötigte, Libraries gelistet.

Electron-Packager

Um das Programm ohne Node.js oder anderen für den Programmierer wichtigen Einrichtungen starten zu können, wird eine Art Compiler, in meinem Fall der Electron-Packager, benötigt.

Dieser bietet die Möglichkeit eine Datei zu erstellen, welche nicht nur das Programm zusammenfasst und es auch auf anderen PCs ausführbar macht, sondern auch für Windows, MacOS und Linux individuell exportieren lässt. Dabei wird dem Entwickler einiges an Arbeit abgenommen, wodurch dessen Anwendungen für jedes Betriebssystem kompatibel werden.

In meinem Fall habe ich die größten Betriebssysteme gewählt, wodurch eine Reihe an Exporten zur Verfügung steht und eine Inkompatibilität zu dem momentanen Zeitpunkt ausgeschlossen werden sollte. Nun mal war es mir nicht möglich, jedes Betriebssystem ausgiebig zu testen, wobei es jedoch egal sein sollte, ob ich nun mein Programm auf einem Windows-Rechner oder z.B. einem Linux-Computer starten möchte. Leider ist es mir jedoch nicht möglich Apple zu unterstützten, da ich dafür mein Programm auf einem dieser Rechner exportieren müsste.

Quellcode

Der Programmcode stützt sich grundlegend auf drei Teilen, wobei unabhängig der CSS, HTML und JavaScript, auch noch JSON-Dateien und Bilder verwendet wurden.

HTML

Die „Hypertext Markup Language“ bzw. HTML fasst, wie jede Webseite, alle wichtigen semantischen Dateien an einen Ort, damit diese beim Start durch den Browser interpretiert und ausgeführt werden können. Es handelt sich um eine Strukturierung von elektronischen Dokumenten, wobei ein grundlegender Aufbau auffällt, welcher folglich am Anfang alle Daten, ob CSS oder JS, lädt und anschließend in dem Body die Benutzeroberfläche aufweist. Diese ist dabei so ausgelegt, dass die Menüs eigene individuelle IDs zugewiesen bekommen, damit nicht nur über dem Code eine Erfassung erleichtert wird, sondern auch die CSS mit diesen Daten umgehen kann.

CSS

Die CSS, also die „Cascading Style Sheets“, sind für das Styling des Dokumentes zuständig. Diese können allgemeine Regeln beinhalten oder auch, abhängig von Tags oder Klassen bzw. IDs, Regeln übermitteln.

In meinem bestimmten Beispiel habe ich mir dies zunutze gemacht, um mit nur ein paar kleinen Regelungen, sowohl einen Dark-Mode und Bright-Mode, anzubieten. Dabei sind die CSS-Dateien so konstruiert, dass sich gegebenenfalls Regeln überschreiben und somit die Farbe erneut von der *lightMode.css* angepasst wird. Aus diesem Grund ist es mir möglich mit der Deaktivierung dieser Datei, alle überschreibenden Regeln aufzuheben und somit einen einfachen Switch-Button zu implementieren.

JavaScript

JavaScript-Dateien oder auch kurz JS-Daten genannt, umfassen die gesamte Logik des Programmes, wobei diese während der Laufzeit interpretiert und ausgeführt werden. Dabei wurde in den verschiedenen Daten, ob *berechnungContent.js*, *berufeSchülerContent.js*, *excel.js*, *functions.js*, *main.js*, *runtime.js*, *settingsContent.js* und *table.js*, zwischen dessen verschiedenen Aufgaben getrennt, um eine Übersichtlichkeit dieser beizubehalten. So ist es mir möglich für die

verschiedenen Aufgaben auf verschiedene Dateien zuzugreifen und somit folglich getrennt die *berechnungContent.js*, das Herzstück meines Programmes, zu erklären.

Der Sortieralgorithmus

Im Folgenden ist der Kern der Sortierung und wichtige helfende Funktionen des Programmcodes erklärt. Dabei spielen die *sort*, als auch die *sortByMinMaxPull*, Funktionen die größte Rolle und haben einen starken Einfluss auf den Ablauf. Jedoch wird auch hinterher schematisch der Algorithmus dargestellt, um eine visuelle Hilfeleistung zu bieten.

Die Sortierung

Zunächst werden für die Umverteilungen der Schüler in ihren Kursen, die durchschnittlichen Kursgrößen benötigt, wobei mithilfe dieser berechnet werden kann, wie wichtig dessen Abgabe bzw. Aufnahme von Schülern ist, um entweder gleichmäßig aufgeteilt zu sein oder den Minimal- bzw. Maximalwerten des Benutzers zu entsprechen.

```
1.  // Get average students for courses if no max and min value is given
2.  let averageCourses = Object.fromEntries(Object.entries(data.courseCouples)
3.      .map(([course, dates]) => {
4.          return [course, dates.reduce((current, date) => {
5.              return current + data.schülerwünsche[date][course].length;
6.          }, 0) / dates.length];
7.      }));
8.
9.  // Get sizes of courses to distribute and sort for iteration
10. getSizeCourses(data.schülerwünsche, averageCourses)
11.     .sort((a, b) =>
12.
13.         // Sorts by importance -- min and max values or average
14.         sortByMinMaxPull(a.students, b.students,
15.            averageCourses[a.course], averageCourses[b.course])
16.
17.     )
18.     .some((sizeCoursesSorted) => {
19.
20.         // Sort students
21.         sort(sizeCoursesSorted, averageCourses, studentWishes);
22.     });
```

16.1 Iteration aller Kurse

So wird, wie gerade angesprochen, in Zeile 2 bis 7 aus *16.1 Iteration aller Kurse* mithilfe von vordefinierten Kurskombinationen, dessen Größe erfasst und in der *sortByMinMaxPull*-Funktion in Zeile 11 und 17 nun bestimmt, wie wichtig die Sortierung des Kurses, in Abhängigkeit der Durchschnittswerte und der momentanen Schüleranzahl, ist. Schließlich werden diese sortierten Kurse in Zeile 18 bis 22 iteriert und die besagte *sort*-Funktion aufgerufen.

sortByMinMaxPull

Diese Sortierung der Kurse tritt bei jeder Kurserfassung auf, wodurch diese zusätzlich in der *sort*-Funktion von Nöten ist und daher vorab erklärt werden sollte.

```
1.  // Sorting method for min and max values
2.  function sortByMinMaxPull(a, b, averageA, averageB) {
3.      if (
4.          (data.min && (a < data.min || b < data.min)) ||
5.          (data.max && (a > data.max || b > data.max))
6.      )
7.          return
8.              ((data.min || 0) - Math.min(a, b) >
9.              Math.max(a, b) - (data.max || Math.max(a, b))) ? a - b : b - a;
10.     else return Math.abs(1 - (b / averageB)) - Math.abs(1 - (a / averageA));
11. }
```

17.1 sortByMinMaxPull

Wird nun *sortByMinMaxPull* aufgerufen, so werden zwei gegebene Kurse entweder, sofern möglich, mit den Minimal- bzw. Maximalwerten abgeglichen oder diese direkt untereinander auf ihre Abweichung zu den Durchschnittsgrößen verglichen.

Sollte also ein Minimal- bzw. Maximalwert vorliegen, so sind zwei Möglichkeiten relevant. Entweder ist der tatsächliche Abstand zwischen dem Minimalwert und den Kursen größer, wodurch, wie in Zeile 9 zu sehen, Kurs *a* von Kurs *b* abgezogen wird und somit der kleine Kurs nach vorne sortiert wird, oder der Abstand zu dem Maximalwert ist größer, woraus entgegengesetzt Kurs *b* von Kurs *a* abgezogen wird und der kleine Kurs hinten landet. So entsteht eine Liste sortiert nach den größten Abständen zu den Inputdaten, wobei auch beachtet wurde, dass womöglich nur einer dieser Werte angegeben wird.

Andererseits kann es auch sein, dass der Benutzer nur die Kursgrößen ausgleichen möchte, wodurch mithilfe der Durchschnittswerte in Zeile 10, die prozentuale Abweichung zwischen den Kursgrößen und dessen Mittelwert von 100% abgezogen wird und mithilfe der *Math.abs*-Statements erfasst, welcher der beiden Kurse eine größere Differenz dieser aufweist. Daraus wird eine Liste kreiert, welche nach dem Abstand der Schüleranzahl zu den Durchschnittswerten sortiert ist.

Es liegt daher nun eine Liste der am wichtigsten zu sortierenden Kursen vor, wobei folglich diese nacheinander sortiert werden können.

sort

Die Sortierung besteht aus drei Hauptteilen, wobei diese jeweils eine Liste, in Abhängigkeit der vorangestellten Iterationen, iteriert. Dies hat zur Folge, dass eine große Reihe an Daten geprüft werden muss, weshalb zur Performanceverbesserung die Funktion *skipIteration* verwendet wurde. Diese überspringt, sofern der Kurs korrekt sortiert wurde, die momentane Schleife und beginnt den nächsten Vorgang. Wichtig ist noch zu sagen, dass diese Funktion *sort*, zwar die Wunschzeit der Schüler verändert, jedoch keineswegs die Wunschkurse und somit beliebig oft ausgeführt werden kann. Sollten Widersprüche entstehen, so muss gegebenenfalls die Funktion *clearCourse*, für die Umsortierung in jegliche Kurse, verwendet werden.

```
1.  // Seek for same course in couples
2.  Object.entries(data.courseCouples[sizeCoursesSorted.course].reduce((current, date) => {
3.
4.      // Remove courses which are blocked to sort into and are the same date
5.      if (date != sizeCoursesSorted.date && (!data.blocked[date] ||
6.          !data.blocked[date].includes(sizeCoursesSorted.course)))
7.              current[date] = data.schülerwünsche[date][sizeCoursesSorted.course];
8.      return current;
9.
10. }, {}))
11. .sort(([courseA, studentsA], [courseB, studentsB]) =>
12.
13.     // Sorted courses by user data
14.     sortByMinMaxPull(studentsA.length, studentsB.length, 1, 1)
15.
16.     // Iterate same courses
17. )
18. .some(([orderedCourse]) => {
```

18.1 Iteration aller selben Kursen

Im ersten Schritt *18.1 Iteration aller selben Kursen* werden dieselben Kurse der anderen Uhrzeiten erfasst, wobei, wie in Zeile 5 bis 7 zu sehen, der momentan vorliegende und zur Sortierung gesperrte Kurse nicht dieser Liste angefügt werden. Diese Liste der Kurse durchlaufen danach den bereits besprochenen Sortierprozess der *sortByMinMaxPull*-Funktion und werden schließlich iteriert. Da es sich in Zeile 14, bei *sortByMinMaxPull*, um dieselben Kurse handelt, wird einfachhalber *1* als Richtwert mitgegeben.

```
1.  // Skip iteration if course is in great condition -- "true" is replied
2.  return skipIteration(sizeCoursesSorted.date, sizeCoursesSorted.course, averageCourses,
    () => {
3.
4.      // Seek for smallest courses to swap with
5.      Object.entries(Object.entries(data.schülerwünsche[sizeCoursesSorted.date][sizeCours
           esSorted.course].reduce((current, student) => {
6.
7.          // Do not add to list if it is the same course or it has been assigned already

8.          // Do not add course if the current time doesnt offer it
9.          if (!current[studentWishes[orderedCourse][student]] &&
10.             studentWishes[orderedCourse][student] != sizeCoursesSorted.course &&
11.             Object.keys(data.schülerwünsche[sizeCoursesSorted.date])
12.                 .includes(studentWishes[orderedCourse][student]))
13.                     current[studentWishes[orderedCourse][student]]
14.                         = data.schülerwünsche[orderedCourse][studentWishes[orderedCours
                                e][student]].length;
15.         return current;
16.
17.     }, {}))
18.     // Remove blocked courses from list
19.     .reduce((current, [course, students]) => {
20.
21.         // Add to current if initial date is not in blocked or not the course
22.         if (!data.blocked[sizeCoursesSorted.date] ||
23.             !data.blocked[sizeCoursesSorted.date].includes(course))
24.                 current[course] = students;
25.
26.         return current;
27.
28.     }, {}))
29.     .sort(([courseA, studentsA], [courseB, studentsB]) =>
30.
31.         // Sorted courses by user data for iteration
32.         sortByMinMaxPull(studentsA, studentsB, averageCourses[courseA],
33.         averageCourses[courseB])
34.
35.     )
36.     // Iterate swap courses
37.     .forEach(([orderedSwapCourse]) => {
```

19.1 Iteration der Alternativkurse

Im nächsten Schritt *19.1 Iteration der Alternativkurse* wird abhängig der sortierten, selben Kursen zu anderen Zeiten, eine Liste aus allen möglichen Kursen derselben Zeit gefertigt, welche die Möglichkeit bieten Schüler aufzunehmen und ebenfalls nicht von der Sortierung gesperrt sind. Dabei müssen die Kurse derselben Zeit ebenfalls in der alternativen Zeit vorliegen, um im Folgeschritt die Umsortierung zu ermöglichen. Wie auch im ersten Schritt werden diese Kurse nach ihrer Relevanz der *sortByMinMaxPull*-Funktion sortiert und iteriert, wobei nun auch erneut die Durchschnittswerte eine große Rolle spielen und daher benutzt werden.

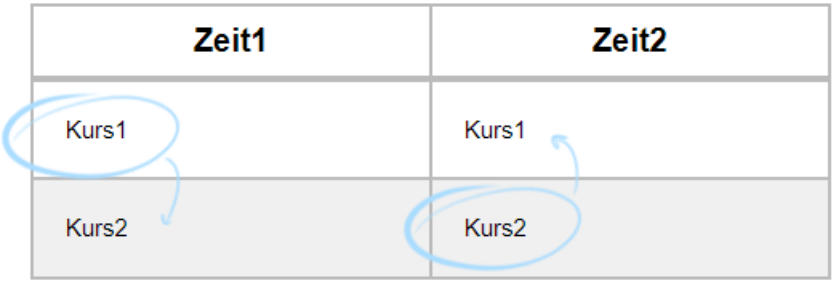
```
1.  // Iterate students out of orderedSwapCourses
2.  data.schülerwünsche[orderedCourse][orderedSwapCourse].some((studentInSwapCourse) => {
3.
4.      // Breaks iteration if "true"
5.      return skipIteration(sizeCoursesSorted.date, sizeCoursesSorted.course, averageCours
           es, () => {
6.
7.          // Checks if student in new time and new course is present in initial
8.          let cutIndex = data.schülerwünsche[sizeCoursesSorted.date][sizeCoursesSorted.co
                urse].indexOf(studentInSwapCourse);
9.
10.         if (cutIndex != -1) {
11.
12.             // Swap courses -
                  - cuts from initial course to new course in same time and cuts from diffe
                  rent course to initial course in different time
13.             data.schülerwünsche[sizeCoursesSorted.date][orderedSwapCourse]
14.                 = data.schülerwünsche[sizeCoursesSorted.date][orderedSwapCourse]
15.                     .concat(data.schülerwünsche[sizeCoursesSorted.date][sizeCoursesSort
                         ed.course]
16.                         .splice(cutIndex, 1));
17.
18.             data.schülerwünsche[orderedCourse][sizeCoursesSorted.course]
19.                 = data.schülerwünsche[orderedCourse][sizeCoursesSorted.course]
20.                     .concat(data.schülerwünsche[orderedCourse][orderedSwapCourse]
21.                         .splice(data.schülerwünsche[orderedCourse][orderedSwapCourse]
22.                             .indexOf(studentInSwapCourse), 1));
23.         }
24.     });
25. });
```

20.1 Iteration der Schüler

Im letzten Schritt *20.1 Iteration der Schüler* werden alle Schüler der alternativen Zeit im alternativen Kurs in einer Schleife durchlaufen, wobei in Zeile 8 geprüft wird, ob ein Schüler dieses Kurses auch in dem Ausgangskurs anwesend ist. Sollte dies der Fall sein, so wird in Zeile 13 bis 16 der Schüler aus der ursprünglichen Zeit des ursprünglichen Kurses in den alternativen

Kurs derselben Zeit verschoben, wobei der Schüler in Zeile 18 bis 22 ebenfalls in der alternativen Zeit des alternativen Kurses in den ursprünglichen Kurs der alternativen Zeit geschoben wird. Dies hat zur Folge, dass der Schüler zwar in den Zeiten nicht mehr seinen Wunsch aufweist, jedoch die gewünschten Kurse unverändert bleiben.

Möchte man sich dies nun in einer Skizze verbildlichen, so kann man von einem Schüler ausgehen, welcher zur *Zeit1* momentan *Kurs1* und zur *Zeit2* den *Kurs2* besucht.



21.1 vereinfachte Darstellung der Sortierung

Nachdem nun die Paare von *Kurs1* und *Kurs2* erfasst wurden, würde das Programm diese ablaufen und in der entgegengesetzten Zeit nach Alternativen suchen. Hier könnte dies *Kurs1* in *Zeit1* sein, wobei schließlich in *Zeit2* auch *Kurs2* erfasst wurde und herausgefunden, dass dieser Kurs ebenfalls in *Zeit1* liegt. Dies hätte zur Folge, dass das Programm die Schüler aus *Kurs2* in *Zeit2* durchläuft und erfasst, dass derselbe Schüler bereits in *Kurs1* mit der *Zeit1* anwesend ist. Dadurch würde er den Schüler aus *Zeit1* in *Kurs1* entfernen, nach *Kurs2* schieben und dementsprechend in *Zeit2* mit *Kurs2* den Schüler in *Kurs1* packen.

So würde der Schüler, wie es auch in *21.1 vereinfachte Darstellung der Sortierung* dargestellt wurde, lediglich in den Kursen springen, wobei der Wunsch erhalten bleibt und nur die Zeit der Wünsche verändert wird.

clearCourse

Sollte eine Umsortierung unter der Beibehaltung der ursprünglichen Wünsche nicht möglich sein, so ist es nur noch möglich, die Schüler gegen ihren Willen in andere Kurse zu stecken. Dafür wurde die Funktion *clearCourse* entwickelt, welche, sofern möglich, alle Schüler aus einem Kurs entfernt.

```
1.  // Search for ideal courses
2.  data.schülerwünsche[date][course].reduce((current, student) => {
3.
4.          // Break if max value is given and current course size is lower than max value
5.          if (startedOverMax && data.schülerwünsche[date][course].length - current.length
                <= data.max) return current;
6.
7.          // Look up best alternative course
8.          Object.keys(data.schülerwünsche[date])
9.              .sort((a, b) =>
10.
11.                 // Sorted courses by user data
12.                 sortByMinMaxPush(data.schülerwünsche[date][a].length, data.schülerwünsc
                        he[date][b].length)
13.
14.             )
15.             .some((alternativeCourse) => {
16.
17.                 // Check that student is not redistributed into same course, that he is
                        not going into the same course at two times and checks if the cu
                        rrent course isnt blocked
18.                 if (alternativeCourse != course &&
19.                     (!studentWishesArray[student].includes(alternativeCourse) || data.d
                         ouble) &&
20.                     !data.blocked[date].includes(alternativeCourse)) {
21.
22.                     data.schülerwünsche[date][alternativeCourse].push(student);
23.                     current.push(student);
24.                     return true;
25.                 }
26.             });
27.
28.         return current;
29.     }, [])
30.     // Remove students from former list
31.     .forEach((student) => data.schülerwünsche[date][course].splice(data.schülerwünsche[
            date][course].indexOf(student), 1));
```

22.1 clearCourse

So ist in *22.1 clearCourse* zu sehen, dass zunächst in Zeile 5 geprüft wird, ob sich die Kursgröße über dem Maximalwert befindet und daher nur Schüler bis zu diesem Wert entfernt werden sollen, wobei im Folgenden die besten, Alternativen Kurse iteriert und die Schüler, solange sie

nicht bereits in diesem Kurs sind oder diesen Kurs bereits zu einer anderen Zeit besetzten, in diesen umverteilt werden.

```
function clearCourseFreeLesson(course, date) {

    // Removes all students from course
    data.schülerwünsche[date][course] = [];

    // Load new table
    schülerwünscheIntoTabe(data.schülerwünsche);
}
```

23.1 clearCourseFreeLesson

Sollte kein Kurs mehr möglich sein, so kann entweder der Schüler einen Kurs doppelt belegen oder mithilfe der *clearCourseFreeLesson*-Funktion aus *23.1 clearCourseFreeLesson*, Freistunden eingetragen bekommen.

```
// Add to blocked courses
data.blocked[date] ? data.blocked[date].push(course) : data.blocked[date] = [course];
```

23.2 Blockierte Kurse

Letzten Endes wird noch, wie in *23.2 Blockierte Kurse* zu erkennen, der zu leerende Kurs in die Liste für blockierte Kurse eingetragen, um eine Umverteilung während der nachfolgenden Sortierung in diese zu vermeiden. Dadurch bleiben leere Kurse auch noch nach der Umsortierung leer, obwohl diese starke Abweichungen aufweisen können.

Fazit

Schlussfolgernd lässt sich sagen, dass mein Programm verlässliche Daten liefert, obwohl bei geringen Schüleranzahlen nicht immer die besten Resultate entstehen. Dies ist hauptsächlich durch die starken Abhängigkeiten der Schülerwünsche zu begründen, wobei sich umso bessere Ergebnisse mit erhöhten Anzahlen an Schülern ergeben. Außerdem lässt sich durch die Iterationen der einzelnen Fächer und Schülern eine doch sehr schnelle Umsortierung verwirklichen, wobei sich Ladezeiten von gerademal ein paar Millisekunden ergeben.

Nach ausgiebigen Testreihen aus zufällig erstellten Daten, war es mir möglich, mit meistens nur einer Iteration der Umverteilung, die Schülerdaten, im Zusammenhang der Minimal- und Maximalwerte, zu sortieren. Lediglich die manuelle Umsortierung der Kursleerung verlangt etwas mehr Aufwand, da durch die entstehende Wunschabweichung dieses Prozesses, auf einen automatisierten Vorgang verzichtet wurde. Dieser soll reflektiert verwendet werden, da nicht immer eine totale Leerung oder die strikte Obergrenze vom Maximalwert von Nöten ist.

Teilweise kann es besser sein, die Wünsche der Schüler über die Beschränkungen zu stellen und somit auch Kurse mit z.B. nur einer geringen, aber dafür interessierten, Anzahl an Schülern zu erlauben. Im Umkehrschluss ist es aber auch möglich, dass es sehr beliebte Kurse gibt und trotz Vollbelegung, nicht alle Wünsche der Schüler berücksichtigt werden können. Hier würde sich empfehlen, lieber das Kursangebot zu erweitern, als Schüler in Kurse zu drängen, für welche sich diese nicht interessieren.

Schließlich stellt sich die Frage, inwiefern das Programm erweitert werden kann. Dabei wäre es möglich, mithilfe von einem Punktesystem, verschiedene Iterationen zur selben Zeit zu durchlaufen und somit dem Benutzer nur die beste Sortierung weiterzugeben. Dies hätte zur Folge, dass noch bessere Umverteilungen ablaufen würden und im Umkehrschluss weniger Schüler in unerwünschte Kurse gehen müssen. Außerdem wäre es auf lange Sicht ratsam, die Schülerdaten trotz des momentan funktionierenden Systems, über eine Webseite zu sammeln und somit den bürokratischen Aufwand der Lehrer zu ersparen.

Anhang

Verwendete Software

Software

https://nodejs.org/ (Node.js, 24.6.2020)

http://www.planetb.ca/syntax-highlight-word (planetB, 24.6.2020)

Libraries

Electron (MIT-Lizenz)

Electron-Packager („BSD 2-Clause "Simplified"“-Lizenz)

jQuery (MIT-Lizenz)

Node XLSX („Apache-2.0“-Lizenz)

Entwicklung

https://github.com/binaryfunt/electron-seamless-titlebar-tutorial (GitHub, 24.6.2020)

https://stackoverflow.com/questions/17067294/html-table-with-100-width-with-vertical-scroll-inside-tbody (Stack Overflow, 24.6.2020)

https://stackoverflow.com/questions/7837456/how-to-compare-arrays-in-javascript (Stack Overflow, 24.6.2020)

Dokumente

https://www.uni-paderborn.de/fileadmin/muwi/PDFs/Selbststa__ndigkeitserkla__rung.pdf (Universität Paderborn, 24.6.2020)

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit mit dem Titel selbstständig und ohne unerlaubte fremde Hilfe angefertigt, keine anderen als die angegebenen Quellen und Hilfsmittel verwendet und die den verwendeten Quellen und Hilfsmitteln wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Ort, Datum

Unterschrift