|  |
| --- |
| Alexander Christopher Bock |
| IPWA01-01 |
| https://github.com/m0m007/Programmierung-von-Webanwendungen |

|  |
| --- |
| Moritz Döring  Informatik  32210824  10.10.2023 |
|  |

Inhalt

1. [Abbildungsverzeichnis III](#_Toc155607908)
2. [Konzept 3](#_Toc155607909)
3. [Umsetzung 3](#_Toc155607910)
4. [HTML-Datei 3](#_Toc155607911)
5. [HTML-Kopf 3](#_Toc155607912)
6. [Header 4](#_Toc155607913)
   1. [Contentbereich 5](#_Toc155607914)
   2. [Footer 6](#_Toc155607915)
7. [Script-Element 7](#_Toc155607916)
8. [Javascript-Datei 8](#_Toc155607917)
9. [getFile() 8](#_Toc155607918)
10. [sortTableByAlphabet() 9](#_Toc155607919)
11. [sortTableByNumbers() 9](#_Toc155607920)
12. [istUnsichtbar() 9](#_Toc155607921)
13. [suchen() 10](#_Toc155607922)
14. [selectedRadio() 10](#_Toc155607923)
15. [zeilenAusblenden() 10](#_Toc155607924)
16. [filtern() 11](#_Toc155607925)
17. [alleZeilenEinblenden() 11](#_Toc155607926)
18. [updateTabelle() 11](#_Toc155607927)
19. [isRtl() 11](#_Toc155607928)
20. [changeToRtl() 12](#_Toc155607929)

# Abbildungsverzeichnis

[Abbildung 1: Ansicht der fertigen Internetseite mit eingezeichneten Bestandteilen 4](#_Toc155607330)

[Abbildung 2: Ansicht der Javascript-Datei 8](#_Toc155607331)

# Konzept

Für die Fallstudie habe ich mir die Aufgabenstellung 1.1 „CO2-Footprint“ ausgesucht. Dafür soll die Internetpräsenz einer Non-Profit-Organisation erstellt werden. Die Organisation beschäftigt sich mit dem Klimawandel und veröffentlicht CO2-Emissionsdaten von Unternehmen und Ländern. Das Design der Internetseite soll die Bemühungen der Organisation widerzuspiegeln.

Die Seite soll dafür modern, aber schlicht sein. Die Seite soll größtenteils Weiß gehalten sein, da zum Beispiel bunte Hintergründe die Seriosität untergraben könnten. Die Seite soll anderseits Farbakzente besitzen, um nicht langweilig zu wirken. Es wurden auf Grund dem Bezug zum Klimaschutz der Organisation ein grünes Farbkonzept gewählt. Das Logo der Internetseite ist eine Fabrik, aus der ein Blatt aus dem Schornstein emporsteigt. Es nimmt damit auch wieder Bezug zu dem Klimaschutz und weist auf die Zukunftsvision der Organisation hin.

Die fertige Internetseite kann unter folgenden Link betrachtet werden:

<https://m0m007.github.io/Emission-Sleuth/>

# Umsetzung

Für die Erstellung der Internetseite wurde Visual Studio Code und Bootstrap verwendet.

# HTML-Datei

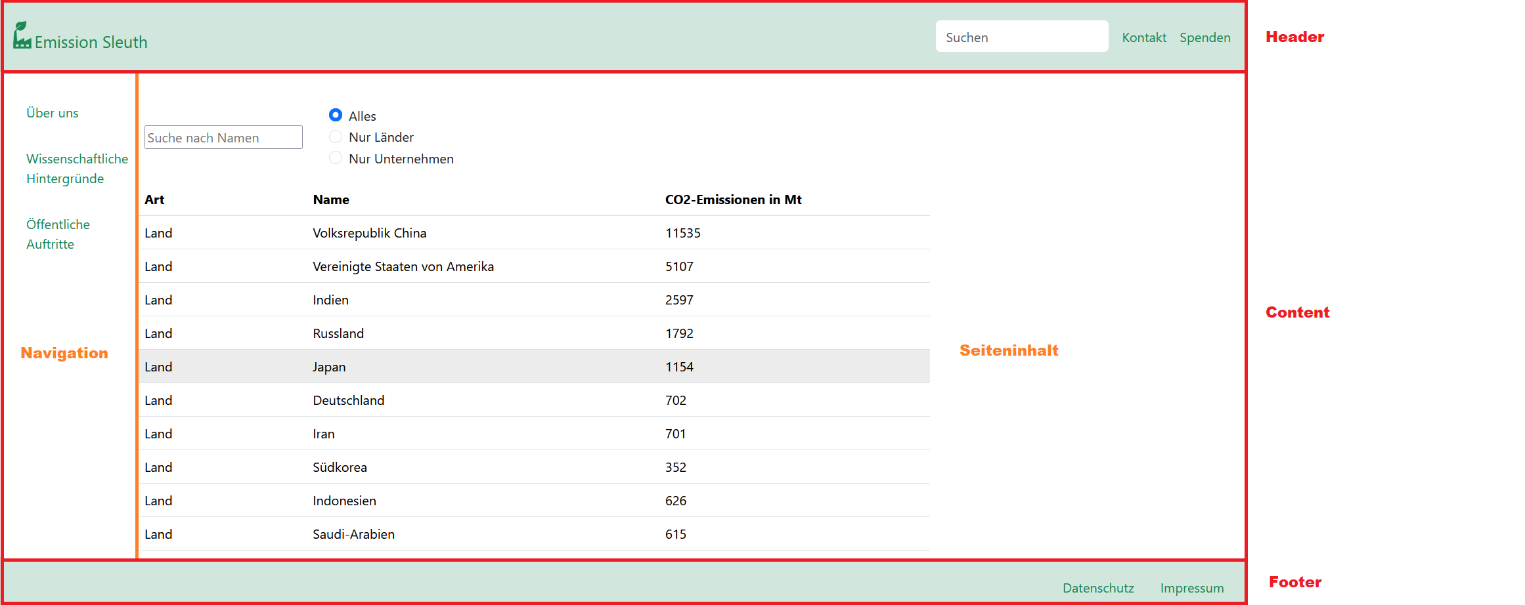
# HTML-Kopf

Die folgenden Elemente sind für jede Seite der Website gleich und werden deshalb nur am Beispiel der Startseite erklärt.

In HTML-Dateien wird als erstes angegeben, dass es sich um eine HTML-Datei handelt (<!DOCTYPE html>). Danach wird deklariert, in welcher Sprache das Dokument verfasst ist (<html lang=“de“). Nach diesen beiden Angaben wird der HTML-Kopf angegeben. Dort werden üblicherweise nur Informationen angegeben, die nicht im Anzeigebereich des Browsers dargestellt werden sollen. Als erstes wird in diesem Fall die Zeichenkodierung, hier UTF-8, angegeben. Danach wird mit dem Viewport der Anzeigebereich der Website angepasst. Mit „width=device-width“ passt sich der Anzeigebereich auf die Bildschirmbreite des Gerätes an. Das Titel Element setzt den Namen in der Tableiste. In der nächsten Zeile wird Bootstrap per CDN eingebunden. Danach wird das Stylesheet „navbar text.css“ eingebunden und zum Schluss noch das Logo.

Jetzt folgt der sichtbare Teil der Internetseite. Dieser besteht aus einem Header, einem Contentbereich und einem Footer. Der Contentbereich besteht selber noch aus einer Navigationsleiste und dem eigentlichen Inhalt der Seite.

**Abbildung 1: Ansicht der fertigen Internetseite mit eingezeichneten Bestandteilen**



Quelle: eigene Darstellung

# Header

Auf der linken Seite des Headers kann man das Logo und den Namen der Organisation sehen (siehe Abbildung 1). Die Standardfarbe für Unternehmensnamen in Navbars ist schwarz und passte damit leider nicht zu dem restlichen Design des Headers. Ich konnte den Text nicht in einem Paragraphen-Element (<p> </p>) einfügen und die Farbe mit der „text-success“ Klasse auf grün setzen, da der Text sonst nicht auf einer Höhe mit den anderen Elementen des Headers ist. In der CSS Datei „navbar text.css“ wird daher die Standardfarbe von Bootstrap für diese spezielle Navbar Art auf grün überschrieben.

Rechts in dem Header stehen zwei Links. Die Links verweisen auf die Seite mit den Kontaktangaben und zu der Seite mit den Angaben für Unterstützerspenden. Der Header ist nicht „sticky“. Das bedeutet, dass wenn man herunterscrollt der Header nach oben verschwindet. Die Bootstrap Klasse „navbar“ hat bereits viele Funktionen implementiert. Ich konnte dadurch leicht die Farbe des Headers ändern und ihn auf die komplette Bildschirmbreite setzen. Durch die „collapse“ Klasse werden die Links in einem aufklappbaren Menü versteckt, falls der Bildschirm zu klein wird.

## Contentbereich

Um den Contentbereich responsive zu gestalten, wurde er mit Flexboxen umgesetzt. Der Header ist zwar ebenfalls eine Flexbox, aber im Gegensatz zu dem Contentbereich ist die Position immer die gleiche und es gibt auch keine zwei Flexboxen nebeneinander.

Dadurch, dass die Navigationsleiste sich direkt neben dem eigentlichen Inhalt der Seite befindet, muss man ein paar Dinge beachten (siehe Abbildung 1). Bootstrap nutzt für die Flexboxen das Gridsystem. Hierbei werden die einzelnen Flexboxen nach dem zur Verfügung stehenden Platz angeordnet. Die Boxen werden wie in einer Tabelle nach Zeilen und Spalten angeordnet. Falls sich die Fenstergröße ändert, ordnet Bootstrap die Boxen neu an und positioniert nebeneinanderliegende Boxen in einer Zeile aus Platzgründen manchmal automatisch untereinander.

Aus diesen Gründen befindet sich der komplette Contentbereich in einer Flexbox. Die erste Spalte beinhaltet die Navigationsleiste. Sie hat eine mit 120px eine feste breite und dazu noch ein Padding von 1, damit die Inhalte selbst bei verschiedenen Bildschirmgrößen nicht ineinander rutschen. Die Navigationsleiste ist in einem <ul>-Element und implementiert die „nav“ Klasse. Die Schrift wird mit der Klasse „text-success“ auf grün gesetzt.

Die zweite Spalte beinhaltet den Seiteninhalt. Alle Seiten bis auf die Startseite beinhalten nur Beispieltext. Die Startseite hingegen beinhaltet die Tabelle mit den Emissionsdaten. Für die Tabelle wird in der Spalte eine neue Flexbox erzeugt. Die erste Reihe dieser Box ist für das Suchfeld der Tabelle und der Filterauswahl da. Das Suchfeld löst nach jedem Key-Up die updateTabelle()-Funktion aus. Die Schaltflächen für die Filter werden im englischen Radios genannt. Sie lösen bei Knopfdruck ebenfalls die updateTabelle()-Funktion aus. Um die Funktionen ausführen zu können besitzt jedes Element dieser Zeile eine eigene individuelle ID. Die Funktionen werden weiter unten separat erklärt.

Nachdem in der ersten Zeile die Eingabe und Filterfunktionen aufgeführt werden, beinhaltet die zweite Zeile die eigentliche Tabelle. Die Tabelle ist mit einem <table>-Element erzeugt worden. Die Tabelle besitzt die ID „tabelle“ die für mehrere Funktionen benötigt wird und ist eine Tabelle der Klasse „table-hover“. Dadurch wird die jeweilige Zeile farblich hervorgehoben, über der die Maus aktuell liegt. <thead> kennzeichnet den Tabellenkopf. Die Zeilen einer Tabelle werden mit <tr> (Table Row) markiert. Die erste Zeile beinhaltet die Überschriften der Tabelle. Durch <th> (Table Head) werden die Einträge fettgedruckt. Die einzelnen Überschriften lösen beim Klicken die Sortierfunktion sortTableByAlphabet() bzw. sortTableByNumbers() aus. Die restlichen Zeilen einer Tabelle sind der Tablebody (<tbody>). Der Tablebody dieser Tabelle ist leer und nur mit der ID „Eintraege“ markiert, da die Tabelle automatisch aus einer JSON-Datei erstellt wird. Die dazu gehörige Funktion lautet getFile().

Da die Internetseite die Position der Navigationsleiste anhand der Sprachkultur der Besucher festlegen soll, müssen noch weiter Änderungen an dem Gridsystem vorgenommen werden. Die erste Spalte beinhaltet, wie bereits erwähnt, normalerweise die Navigationsleiste und die zweite Spalte den Seiteninhalt. Falls die Sprache des Besuchers von Rechts-Nach-Links (RTL) gelesen wird, soll die Navigationsleiste rechts neben den Seiteninhalt dargestellt werden. Das bedeutet, dass in diesem Fall nur die Spalten getauscht werden müssen. Um dies zu tun, gibt es in Bootstrap die „Order“-Eigenschaft. Sie kann bei der Erzeugung einer Spalte als Klassenkürzel angegeben werden. Wenn man zum Beispiel der Spalte des Seiteninhalts die Klasse „order-1“ und die Spalte der Navigationsleiste „order-2“ zuweist, könnte man die Menüausrichtung an andere Sprachkulturen anpassen. Beim Aufruf der Seite wird die Sprache des Browsers ausgelesen und anhand dieser den Spalten entweder „order-1“ oder „order-2“ zugewiesen. Jedoch ergibt sich dadurch für RTL Sprachen das Problem, dass ab einer zu kleinen Bildschrimbreite die Navigationsleiste unter der Tabelle angezeigt wird, da die nebeneinanderliegenden Spalten der Reihe nach untereinander gesetzt werden. Diese Version der Internetseite war deshalb nicht benutzbar. Um dieses Problem zu beheben, muss daher programmatisch auf die Bildschirmbreite reagiert werden.

Das Gridsystem von Bootstrap besitzt auch bereits implementierte Bildschrimbreitenschwellwerte, an denen automatisch Änderungen ausgelöst werden können. Die Bildschrimbreite extra small (xs) beschreibt alle Breiten von 0-576 px, alles bis 767px gehört zu small (sm), alles bis 991px zu medium (md), alles bis 1199px zu large (lg), alles bis 1399px zu extra large (xl) und alles ab 1400px zu extra extra large (xxl). Mithilfe der Order Eigenschaft und der Schwellwerte können die Anordnung der Spalten responsiv verändert werden. Dazu muss der Spalte eine Klasse hinzugefügt werden, die das Stichwort „order“, die betreffende Bildschirmbreite und die Position beinhaltet. Zum Bespiel „order-xs-1“.

Die Funktion changeToRtl() überprüft, ob die Leserichtung der Sprache und fügt abhängig davon den Spalten weitere Klassen hinzu, damit das Menü bei größeren Bildschirmen auf der linken oder rechten Seite erscheint.

Die Reihenfolge der Spalten für Bildschirme der Größe xs und sm wird, unabhängig der Leserichtung der Sprache, immer gleich gehalten. Dadurch ist die Navigationsleiste bei kleinen Bildschirmbreiten immer oben und die Tabelle immer unten. Deshalb besitzt die Navigationsleistenspalte die Klassen „order-xs-1“ und „order-sm-1“ und die Tabellenspalte die Klassen „order-xs-2“ und „order-sm-2“.

## Footer

Der Footer ist das unterste Element einer Internetseite (siehe Abbildung 1). Er ist auf jeder Seite gleich und beinhaltet normalerweise rechtliche Hinweise wie zum Beispiel das Impressum oder ähnliches. Der Footer befindet sich in einer eigenen Zeile, dadurch können die Abstände zu anderen Inhalten leichter eingestellt werden. Der Footer verwendet das <footer>-Element von Bootstrap. Außerdem werden die Klassen „d-flex“, „fixed-bottom“, „justify-content-end“ und „bg-success-subtle“ verwendet. Dadurch erscheint der Footer mit grünem Hintergrund und die Links auf der rechten Seite des Bildschirms. „fixed-bottom“ sorgt dafür, dass der Footer immer am unteren Rand des Bildschirms liegt. Das ist wichtig, da durch die Filter- und Suchoptionen die Tabelle kürzer sein kann und die Internetseite dann nur die Hälfte der Bildschrimhöhe einnehmen würde. Das wäre auch der Fall bei anderen Seiten, die nicht genug Inhalt besitzen.

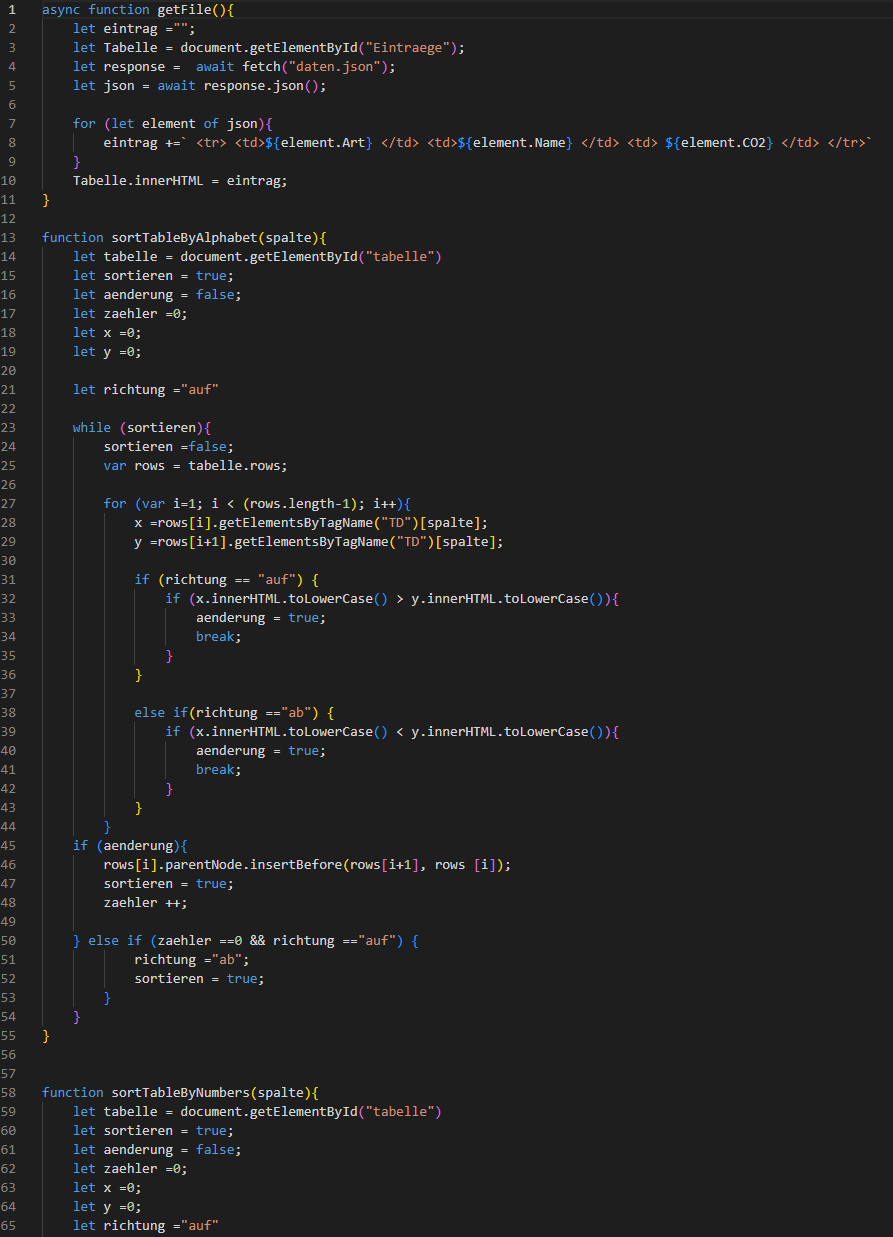
# Script-Element

Mit dem <script>-Element kann zum Beispiel Javascript direkt in einer HTML-Datei geschrieben werden oder auch externe Javascript-Dateien eingebunden werden. In diesem Fall wird als erstes ein Bootstrap Javascript-Paket eingebunden. Danach wird meine eigene Javascript-Datei mit allen Funktionen der Internetseite eingebunden. Zum Schluss wird die getFile() und changeToRtl() Funktion ausgeführt. Da HTML-Seiten von oben nach unten aufgebaut werden, werden diese Funktionen als aller letztes ausgeführt.

# Javascript-Datei

Wie in „Script-Element“ bereits erwähnt wurde kann Javascript-Code direkt in einer HTML-Datei geschrieben werden. Ich habe mich aber dafür entschieden eine eigenständige Javascript-Datei zu erstellen. Meiner Meinung macht das beide Dateien lesbarer, da die Sprachen nicht vermischt werden. Die HTML-Datei beinhaltet alle sichtbaren Elemente und die Javascript-Datei enthält alle zugrunde liegenden Funktionen der Seite.

**Abbildung 2: Ansicht der Javascript-Datei**



Quelle: eigene Darstellung

Im Folgenden werden alle Funktionen der esFunctions.js-Datei erklärt.

# getFile()

Diese Funktion liest die Emissionsdaten aus einer JSON-Datei und erstellt daraus den Tabellenkörper auf der Startseite her. Die getFile()-Funktion ist eine asynchrone Funktion, da sie den await Operator verwendet. Dieser Operator kann die Ausführung von Codezeilen, die von seinem Ergebnis abhängig sind verzögern. Als erstes wird der Tabellenkörper über dessen ID aufgerufen. Danach wird die Datei geöffnet. Die Antwort auf die fetch-Anfrage, die in response gespeichert ist, kann durch die .json() Methode in ein Javascript Objekt umgewandelt werden. Mit einer For-Schleife wird nun durch jeden Eintrag der JSON-Datei iteriert. Bei jedem Durchlauf wird ein String mit dem HTML-Format für eine Tabellenzeile erzeugt und an die vorherige angehängt. Durch zwei Backticks (`) kann man mehrere Strings aneinanderhängen ohne den + Operator verwenden zu müssen. Zum Schluss wird der so erzeugte Tabellenkörper in die Tabelle eingefügt.

# sortTableByAlphabet()

Diese Funktion sortiert die Einträge in der Tabelle entweder aufsteigend oder absteigend. Durch den Parameter „Spalte“ kann man entscheiden welche Spalte der Tabelle sortiert werden soll. Als erstes wird das erste und zweite Element aus der Tabelle entnommen. Hierfür wird wieder die ID der Tabelle verwendet. Der Text der Elemente wird in Kleinbuchstaben umgewandelt und mit dem > Operator verglichen. Falls das vorherige Element größer als das zweite Element ist, wird „aenderung“ auf true gesetzt, das bedeutet, das eine Änderung an der Reihenfolge dieser Elemente vorgenommen werden muss. Dann bricht die For-Schleife ab und dreht die Reihenfolge dieser Elemente um. Falls das vorherige Element nicht größer ist, muss das erste Element nicht verschoben werden und es wird das zweite mit dem dritten Element der Tabelle verglichen. Dies ergibt eine absteigende Sortierung. Die Anzahl aller notwendigen Änderungen wir in „zaehler“ gespeichert.

Wenn die Tabelle bereits absteigend sortiert sein würde, würde der Zähler, nach einem Durchlauf mit der oben beschriebenen Methode, den Wert null besitzen. Wenn also die Sortierrichtung absteigend ist und der Zähler nach einem Durchlauf den Wert null hat, soll nochmals sortiert werden, aber dieses Mal aufsteigend. Der einzige Unterschied dabei ist, dass mit < (kleiner als) anstatt > (größer als) verglichen wird.

# sortTableByNumbers()

sortTableByNumbers() funktioniert fast genauso wie sortTableByAlphabet(). Diese Funktion musste eingeführt werden, da sortTableByAlphabet() Zahlen nur nach der Größe der Ziffern sortieren würde. Zum Beispiel wäre 99 „größer“ als 9000. Daher wurde die komplette Funktion übernommen, aber der Zahlenstring wird mit der parseFloat-Funktion in eine Zahl umgewandelt und dann verglichen.

# istUnsichtbar()

Die Funktion gibt ein true zurück falls, ein Element die Displayeigenschaft auf „none“ gesetzt hat. Dies ist notwendig, da die Such- und Filterfunktionen die einzelnen Einträge nicht aus der Tabelle löschen, sondern sie nur unsichtbar machen. Mit jeder Änderung wird die komplette Tabelle wieder sichtbar gemacht und danach blendet jede Funktion alles aus, was nicht ihren Anforderungen entspricht. Die einzelnen Such- und Filterfunktionen werden nacheinander durchgeführt und bereits ausgeblendete Zeilen werden ignoriert. Dadurch werden unnötige Vergleiche vermieden und die Funktionen schneller.

# suchen()

Für die Suche mit der Suchleiste wird diese Methode benötigt. Als erstes wird der Suchbegriff in der Suchleiste über die ID ermittelt und in Kleinbuchstaben in der Variable „filter“ gespeichert. Danach wird auch die Tabelle über die ID aufgerufen und die einzelnen Reihen durch den „tr“ Tag ermittelt. Die einzelnen Zeilen werden in der Variable „zeilen“ abgelegt. In einer For-Schleife wird durch alle Zeilen durchiteriert. Es wird jeweils nur die zweite Spalte, also die Zeile mit dem Index 1 verglichen. Wenn ein Eintrag vorhanden ist wird anschließend geguckt, ob dieser Eintrag bereits unsichtbar gemacht wurde. Dafür wird die eben beschriebene istUnsichtbar()-Funktion verwendet.  
Falls der Eintrag unsichtbar ist, wird er übersprungen. Falls der Eintrag sichtbar ist, wird der Eintrag mit dem Suchbegriff verglichen. Der Eintrag wird dafür über .textContent oder .innerText aufgerufen, da dieser über verschiedene Wege dort abgelegt werden kann. Einer dieser beiden Methoden kann aber den Text aus dem Eintrag ziehen. Der eigentliche Vergleich findet über .indexOf statt. Wenn der Suchbegriff in einem der Einträge teilweise oder komplett vorhanden ist wird dieser sichtbar gelassen und ansonsten ausgeblendet. Durch .textContent und .innerText wird der im Suchfeld eingegebene text nicht wie HTML-Code behandelt. Dadurch kann man keinen HTML-Code in die Webseite injizieren.

# selectedRadio()

Die Filterfunktionen können über drei sogenannte Radios ausgewählt werden. Es gibt die Auswahl alles, nur die Länder oder nur die Unternehmen anzeigen zu lassen. Es ist immer genau eine Option ausgewählt. Im Standardfall wird nicht gefiltert und somit alles angezeigt. selectedRadio() ermittelt welche der drei Optionen ausgewählt ist. Die Radios wurden alle mit dem Namen „radio“ versehen. Die Funktion .getElementsByName() wird mit dem Parameter „radio“ ausgeführt und erstellt daraufhin ein Array mit allen Elementen mit dem Namen „radio“. Dieses Array wird mit einer For-Schleife durchiteriert und es wird kontrolliert, ob die Eigenschaft .checked wahr ist. Diese Eigenschaft ist immer wahr, wenn dieses Radio ausgewählt ist. Wenn dies der Fall ist wird der Index dieses Radios mittels der Funktion parseInt() in einen Integer umgewandelt. Der Parameter „10“ gibt an, dass der Integer eine Zahl zur Basis 10 sein soll.

# zeilenAusblenden()

Diese Funktion ist wichtig für die Filterfunktion der Tabelle. Als Parameter kann man einen Modus übergeben. Der Modus gibt an welche Art Daten man betrachten möchte. In unserem Fall gibt es Unternehmen und Länder. Die Funktion ruft wieder, wie oben erklärt, die Tabelle auf und speichert die einzelnen Zeilen in eine Variable. Die einzelnen Zeilen werden wieder durchiteriert, aber dieses Mal wird nur die erste Spalte betrachtet, also die mit dem Index 0. Wenn die Art des Datenpunktes, aufgerufen durch .innerText, nicht mit dem Modus übereinstimmt, wird diese Zeile ausgeblendet.

# filtern()

Die Funktion filtern() ermittelt als erstes über selectedRadio() welche Filteroption ausgewählt wurde. Dieser Wert wird an einem Switch übergeben. Ein Switch ist eine Struktur, die anhand einer Zahl entscheidet welcher Code ausgeführt werden soll. Mit ihr kann man Fallunterscheidungen ermöglichen. In diesem Fall wird durch den switch entschieden welcher Modus für zeilenAusblenden() verwendet werden soll.

# alleZeilenEinblenden()

Diese Funktion iteriert wie mehrfach beschrieben durch alle Zeilen der Tabelle und setzt jede .style.display Eigenschaft der Zeilen auf „“, dadurch sind sie sichtbar.

# updateTabelle()

Diese Funktion bündelt fast alle bisherigen Funktionen. Sie wird mit jeder Änderung der Such- und Filterfunktionen ausgeführt. Als erstes werden mit alleZeilenEinblenden() alle Zeilen eingeblendet und die Tabelle in den Ursprungszustand zurückversetzt. Danach wird gefiltert mit der filtern()-Funktion und zum Schluss werden die übrigen Zeilen noch durchsucht mit der suchen()-Funktion.

# isRtl()

Die Webseite besitzt die Möglichkeit das Menü je nach Sprachkultur links oder rechts auszurichten. Zu erst muss dafür ermittelt werden welche Sprachkultur vorliegt. Die Funktion isRtl() besitzt eine Liste („tags“) der Sprachen, die von rechts nach links gelesen werden. Diese Liste wurde von mir selber zusammengestellt und basiert auf wikipedia.org und worldatlas.com. Über navigator.language kann die eingestellte Sprache des Browsers ausgelesen werden. Die ausgelesene Sprache wird mit zwei Buchstaben abgekürzt dargestellt. Die Variable „rtl“ wird am Anfang mit false initialisiert. Zum Schluss wird die Liste der Sprachkürzel mit dem ausgelesenen Sprachkürzel nacheinander verglichen und bei einer Übereinstimmung wird die Variable „rtl“ auf true gesetzt. Falls es keine Übereinstimmung gibt wird rtl nie auf true gesetzt. Die Funktion gibt rtl zurück.

# changeToRtl()

Wie unter Contentbereich angemerkt fügt changeToRtl() den Spalten im Gridsystem Klassen zu, damit die Navigationsleiste je nach Sprachkultur rechts oder links dargestellt wird. Dafür werden die beiden Spalten des Gridsystem über ihre individuelle ID aufgerufen. Die danach folgende If-Schleife prüft mit isRtl() welche Sprachkultur vorliegt und fügt daraufhin Klassen mit der richtigen Order-Eigenschaft hinzu.

# Fazit

Mithilfe von Visual Studio Code und Bootstrap konnte eine anschauliche und übersichtliche Internetseite erstellt werden. Die Internetseite passt sich nicht nur der Fenstergröße, sondern auch der Sprachkultur der Nutzer an. Durch Bootstrap ist das Farbdesign und die Platzierung der Elemente leicht veränderbar und man kann jeden etwaigen Kundenwunsch leicht umsetzen. Javascript ist sehr verbreitet. Es gibt viele Beispiele zu allen möglichen Anwendungen und viele Foreneinträge mit Lösungsbeispielen. Durch diese Anwendungen und Programmiersprachen konnte ich ohne Vorwissen schnell eine funktionsfähige Internetseite erstellen und zukünftige Änderungsvorschläge können leicht umgesetzt werden.

# Literaturverzeichnis

Wikipedia. (2023). *IETF language tag.* https://en.wikipedia.org/wiki/IETF\_language\_tag

WorldAtlas. (2018). *Which Languages Are Written From Right To Left?.* https://www.worldatlas.com/articles/which-languages-are-written-from-right-to-left.html