

Layoutumsetzung

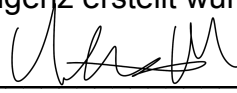
„Formular Validierung & Konzept“

Modulnummer: 4FSC0WD002 0923
Modulname: Front-End Development
Abgabedatum: 10.04.2024
Abschluss: Bachelor of Web Development
Semester: Juli 3
Name: Lazar Minkov
Campus: Zürich
Land: Schweiz
Wortanzahl: 1421

Selbstständigkeitserklärung:

Hiermit bestätige ich, dass ich die vorliegende Arbeit eigenständig erarbeitet habe und alle Hilfsmittel sowie Inhalte von Dritten vollständig angegeben wurden. Hierunter fällt zum Beispiel die korrekte Belegarbeit für die direkte oder sinngemäße Übernahme von Texten, Bild-, Audio- und Videomaterial sowie Code etc. Weiterhin die klare Kennzeichnung von Inhalten, die von anderen Personen oder technischen Hilfsmitteln wie z.B. einer künstlichen Intelligenz erstellt wurden.

10.04.2024
Ort, Datum


Unterschrift Student/in

Formular Validierung & Konzept

25.06.2024

Diese Dokumentation erklärt das Vorgehen der Funktionalitäten bei der Form Validierung meines Projektes.

Übersicht

HTML Struktur

Als erstes wurden die HTML Struktur und CSS für das Styling der Seite genutzt.

Die HTML Struktur aufgebaut. Es gibt insgesamt ein Formelement mit acht Inputs, davon zwei Radio Buttons für die Anrede und 6 Inputs für die Daten des Nutzers, zusätzlich gibt es noch ein Textarea-Element für eine Nachricht.

```
<form>

  <div class="radio-group">
    <label>
      <input type="radio" name="title" value="mrs"> Mr
    </label>
    <label>
      <input type="radio" name="title" value="ms"> Ms
    </label>
  </div>

  <div class="first-name">
    <label for="first-name">First name</label>
    <input type="text" id="first-name" />
  </div>
  <div class="last-name">
    <label for="last-name">Last name</label>
    <input type="text" id="last-name" />
  </div>

  <div class="message">
    <label for="message">Message</label>
    <textarea id="message"></textarea>
  </div>

</form>
```

```

</div>

<div class="email">
  <label for="email">E-Mail</label>
  <input type="text" id="email" />
</div>

<div class="address">
  <label for="address">Address</label>
  <input type="text" id="address" />
</div>

<div class="postcode">
  <label for="postcode">Postcode</label>
  <input type="text" id="postcode" />
</div>

<div class="city">
  <label for="city">City</label>
  <input type="text" id="city" />
</div>

<div class="message">
  <label for="message">Message</label>
  <textarea id="message"></textarea>
</div>
<button>Submit</button>
</form>

```

JavaScript Funktionalitäten

Die zweite Schritt war die Inputs zu Validieren und Funktionalitäten einzubauen. Die Idee ist, einzelne Inputs mit JavaScript zu validieren, um zu überprüfen, ob die Eingaben der Nutzer die Vorgaben entsprechen.

Die Form Validierung muss bestimmte Anforderungen erfüllen:

1. Die Eingaben müssen sofort nach dem Ausfüllen validiert werden.
2. Es müssen Sonderzeichen bei bestimmten Inputs nicht zugelassen sein
3. Es gibt Kriterien, die stimmen müssen, damit die Eingaben gültig sind.
4. Alle Felder müssen ausgefüllt werden, um fortzufahren.
5. Der Submit Button validiert nochmal die Daten des Nutzers

Eventlistener für die Button:

```
document.querySelector("button").addEventListener("click", validateForm);
```

Der Eventlistener wartet für ein "Click" Event. Wenn der Submit button geklickt wurde, wird die Form abgeschickt.

Eingabefelder und Textarea:

```
document.querySelectorAll("input, textarea").forEach((input) => {  
  input.addEventListener("blur", validateInput); });
```

Der "`querySelectorAll`" Methode wählt alle Inputs und Textareas aus. Der `blur` event wird in der Funktion verwendet, um die Eingabefelder nur dann zu validieren, wenn der Input von dem Nutzer verlassen wurde.

Input Validation

Die `validateInput` hat `event` als Parameter, welche aktiviert wird, wenn der User mit der Input interagiert. Die Eingaben werden individuell validiert, basierend auf ihrer ID.

```
function validateInput(event) {  
  const input = event.target;  
  const value = input.value;  
  let errorMessage = "please fill the inputs";  
  
  switch (input.id) {  
    case "first-name":  
      errorMessage = validateName(value, "First name");  
      break;  
    case "last-name":  
      errorMessage = validateName(value, "Last name");  
      break;  
    case "email":  
      errorMessage = validateEmail(value);  
      break;  
    case "message":  
      errorMessage = validateMessage(value);  
      break;  
  }  
}
```

```

    case "address":
        errorMessage = validateAddress(value);
        break;
    case "postcode":
        errorMessage = validatePostcode(value);
        break;
    case "city":
        errorMessage = validateCity(value);
        break;
}

displayError(input, errorMessage);
}

```

`const input = event.target;` nimmt das HTML Element, welches von dem Event aktiviert wurde.

`const value = input.value;` ruft der Value welchen von der User ab, welches in der Input eingegeben wurde

`let errorMessage = "the validation has failed";` definiert ein normale Fehlermeldung

In der `validateInput` wird `switch` statement (Stellung) verwendet, um zu überprüfen, welche Validierung Funktionen für die verschiedenen Inputs ausgerufen sein müssen, wie z.B. für E-Mail Adresse, Postleitzahl und Telefonnummer.

```
displayError(input, errorMessage);
```

dient als Fehlermeldung, wenn die Validierung nicht abgerufen werden kann.

Form Validierung

Die Funktion `validateForm` verhindert, dass das Formular gesendet wird, wenn Validierungsfehler vorliegen. Es sammelt die Daten aus dem Formular. Es prüft jedes Feld und prüft, ob alle Felder richtig sind. Oder es zeigt Fehlermeldungen für die nicht ausgefüllten Felder.

```
function validateForm(event) {  
  event.preventDefault() ;  
}
```

Hiermit wurden die Funktionen definiert und mit `.preventDefault` wird die Standard-Einreichung der Form verhindert.

```
let data = {};  
let validationErrors = {};
```

mit diese Variablen werden objekte eingerichtet, um die Formulardaten und Validierung Meldungen zu speichern

Sammeln der Formdaten: Hier werden die HTML- Elemente der Form selektiert und der Input der User wird in dem ``data`` Objekt gespeichert.

```
data.firstName = document.querySelector("#first-name").value;  
data.lastName = document.querySelector("#last-name").value;  
data.email = document.querySelector("#email").value; data.message =  
document.querySelector("#message").value; data.address =  
document.querySelector("#address").value; data.postcode =  
document.querySelector("#postcode").value; data.city =  
document.querySelector("#city").value;
```

Mit dieser If statement werden Fehlermeldungen(error messages) entfernt, nur die aktueller Resultate der Eingaben werden an den Nutzer gezeigt

```
if (document.querySelector("form span")) {  
  document.querySelectorAll("form span").forEach((element) => {  
    element.remove(); }); }  
}
```

Jede Zeile Validieren: Validiert jedes Formularfeld mit hilfe spezifischer Validierung Funktionen und speichert Fehlermeldungen im ``validationErrors`` Variable.

```
validationErrors.firstName = validateName(data.firstName, "First
name");
validationErrors.lastName = validateName(data.lastName, "Last
name");
validationErrors.email = validateEmail(data.email);
validationErrors.message = validateMessage(data.message);
validationErrors.address = validateAddress(data.address);
validationErrors.postcode = validatePostcode(data.postcode);
validationErrors.city = validateCity(data.city);
```

Fehlern überprüfen und Einreichung: Mit der if und else Conditions(Bedingungen) werden die Daten überprüft. Wenn keine Errors vorkommen, werden die Daten in der Konsole eingeloggt. Die Errors werden mit der 'displayErrors' funktion angezeigt

```
if (Object.keys(validationErrors).every(key =>
!validationErrors[key])) {
    console.log(data);
} else {
    displayErrors(validationErrors);
}
```

Diese Stellung hilft, die Eingaben der Nutzer nur dann in ein Objekt auszuführen, wenn sie gültig sind.

Validierungsfunktionen: Die schwierigste Teil ist, die Funktionen für die Validierung zu programmieren.

Für jeden Input gibt es Funktionen mit `if` und `else` conditions(Bedingungen). Für die Funktionen werden Conditional Statements verwendet.

```
function validateName(name, fieldName) {
    if (!name) {
        return `No ${fieldName.toLowerCase()} provided`;
    } else if (name.length <= 3) {
        return `${fieldName} must be at least 3 characters long`;
    }
}
```

```

    }
    return "";
}

```

1. Die erste Funktion `validateName` validiert die **Vor- und Nachname**, die beiden Inputs entsprechen einer `name` Parameter, aus diesem Grund ist nur eine Funktion für die beiden Namen der Nutzer notwendig, da die Anforderung für die Inputs gleich sind.

Das erste if Statement wird dann ausgeführt, wenn das Feld vom User leer ist. Es wird ein error message an die Nutzer verschickt.

Das Else if Statement ist eine weitere Kondition, welche überprüft, ob der Name der Nutzer mindestens aus 3 oder mehr Zeichen besteht. Wenn die Eingaben der Nutzer den Anforderungen entsprechen, wird keine Fehlermeldung angezeigt.

2. Die Funktion für die Validierung der E-Mail-Adresse der Nutzer `validateEmail` muss sicherstellen, dass es sich um eine gültige Adresse handelt. Mit Hilfe von Regex oder Regular Expression, ist eine Zeichenfolge, die ein Suchmuster gebildet. Es wird hauptsächlich für den String-Abgleich und die Manipulation verwendet.

```

function validateEmail(email) {
    if (!email) {
        return "No email provided";
    } else {
        const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
        if (!emailRegex.test(email)) {
            return "Email is invalid, please check again";
        }
    }
    return "";
}

```

Diese Funktion ist auf derselben Logik wie die Namen Validierung Funktion programmiert, bis auf das sie mit Regex die Gültigkeit der E-Mail vom User überprüft. Wenn der Input leer ist, wird eine Fehlermeldung an den User geschickt. Wenn eine E-Mail-Adresse angegeben wurde, aber die E-Mail nach dem Regex Test nicht gültig ist, wird der Nutzer darauf hingewiesen. Wenn der E-Mail gültig ist,

wird keine Meldung auftauchen.

```
function validateMessage(message) {  
  if (message !== "" && message.length < 31) {  
    return "The message must be at least 30 characters long";  
  }  
  return "";  
}
```

3. Funktion `validateMessage` kontrolliert, ob die Nachricht der Nutzer die Anforderungen erfüllt mindestens einunddreissig Zeichen lang zu sein. Der if statement schickt eine Fehlermeldung, wenn keine Nachricht eingegeben ist oder nicht die Mindestlänge einhält.

```
function validateAddress(address) {  
  if (!address) {  
    return "No address provided";  
  }  
  return "";  
}
```

4. `validateAddress`: Bei dieser Funktion geht es um die Wohnadresse der Nutzer, welche vielseitig sehr unterschiedlich sein kann, in die verschiedenen Ländern, deswegen kann man nicht Anforderungen in dieser Funktion einbauen. Es wird nur geprüft, ob eine Adresse eingegeben wurde.

```
function validatePostcode(postcode) {  
  if (!postcode) {  
    return "No postcode provided";  
  } else {  
    const swissPostcodeRegex = /^[1-9]\d{3}$/;  
    if (!swissPostcodeRegex.test(postcode)) {  
      return "Postcode is invalid";  
    }  
  }  
  return "";  
}
```

5. `validatePostcode` überprüft die Postleitzahl der Nutzer, die Struktur dieser Funktion ist ähnlich wie bei der `validateEmail`, bis auf der Regex, welches für PLZ definiert ist, besonders für schweizer Postleitzahlen, aufgrund dieses Projektes, was es innerhalb der Schweiz entwickelt worden ist. Für die Zukunft werden die Funktionen verarbeitet, um solche Adressen innerhalb Europas validieren zu können. Für diesen Zeitpunkt wird überprüft, ob eine gültige schweizer Postleitzahl angegeben wurde.

6. `validateCity` überprüft, dass eine Stadt von der Nutzer angegeben wurde. Wenn keine Stadt angegeben wurde, wird eine Fehlermeldung zugeschickt. Wenn ein dieses Feld ausgefüllt ist, wird keine Fehlermeldung vorkommen.

```
function validateCity(city) {  
  if (!city) {  
    return "No city provided";  
  }  
  return "";  
}
```

Folgende Funktionen sind in der `validateForm` eingebaut.

Fehleranzeige Funktionen:

```
function displayError(input, errorMessage) {  
  if (errorMessage) {  
    let errContainer = input.nextElementSibling;  
    if (!errContainer || errContainer.tagName !== "SPAN") {  
      errContainer = document.createElement("span");  
      input.after(errContainer);  
    }  
    errContainer.innerHTML = errorMessage;  
  } else {  
    const errContainer = input.nextElementSibling;  
    if (errContainer && errContainer.tagName === "SPAN") {  
      errContainer.remove();  
    }  
  }  
}
```

`displayError` Diese Funktion stellt sicher, dass Fehlermeldungen korrekt neben den entsprechenden Eingabefeldern angezeigt und entfernt werden, wenn keine Fehler vorliegen.

Im Bezug zu der Parameter werden der `input`, steht für das Eingabefeld, was validiert sein muss und `errorMessage`, für die Fehlermeldung, welches gezeigt oder entfernt werden sollte.

Falls ein `errorMessage` vorkommt, überprüft die Funktion, ob ein Spannelement neben dem Input existiert.

Wenn kein solches `span` existiert, wird ein solches Element erstellt und die Fehlermeldung wird als Inhalt des Elementes gesetzt.

Wenn ein solches Element bereits vorhanden ist, wird es entfernt, um die Fehlermeldung zu löschen, wenn keine Fehler vorkommen.

```
function displayErrors(errors) {  
  for (const key in errors) {  
    if (errors[key]) {  
      const input = document.querySelector(`#${key}`);  
      displayError(input, errors[key]);  
    }  
  }  
}
```

`displayErrors` Die Funktion ermöglicht die Anzeige mehrerer Fehlermeldungen für die Eingabefelder.

```
if (Object.keys(validationErrors).every(key =>  
!validationErrors[key])) {  
  console.log(data);  
} else {  
  displayErrors(validationErrors);  
}
```

Die Stellung für die **Fehlern überprüfen und Einreichung**, welches sich innerhalb der ``validateForm`` befindet, nimmt Bezug auf die ``displayErrors`` Funktion.

``displayErrors`` iteriert über jedes "key" in die ``errors`` Objektive. Es wird für jedes "key" überprüft, ob dem Key eine Fehlermeldung zugeordnet ist.

Wenn ein Fehler vorkommt, wird das input element mit der dazugehörigen key mit ``document.querySelector`` ausgewählt. dann wird die ``displayError`` aufgerufen und übergibt das Eingabeelement und die entsprechende Fehlermeldung.

Quellenverzeichnis

Form Validierung Konzept;

Anon. 2024. *day 08 · sribis/wd923_javascript@09e78b4*. [online] Available at:
<https://github.com/sribis/wd923_javascript/commit/09e78b4823c165b65cedfc9d85a01685ca7c9df6> [Accessed 30 June 2024].

Textbearbeitung, Form-Validierung(Debugging) Entwicklung;

Anon. 2024. *ChatGPT*. [online] Available at:
<<https://chatgpt.com>> [Accessed 30 June 2024].

Grammatik-Rechtschreibprüfung;

Anon. 2024. *Rechtschreibprüfung für Deutsch*. [online] LanguageTool. Available at:
<<https://languagetool.org/de/rechtschreibpruefung-deutsch>> [Accessed 30 June 2024].

Code Block in Docs gestalten;

Anon. 2024. *Code Blocks - Google Workspace Marketplace*. [online] Available at:
<https://workspace.google.com/marketplace/app/code_blocks/100740430168>
[Accessed 30 June 2024].