

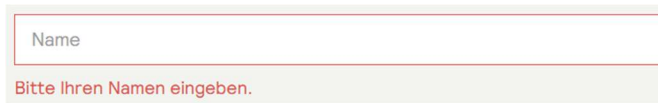
Richtline 3.3 – WCAG 2.2

Inhaltsverzeichnis

3.3.1 – Fehleridentifizierung A	2
Implementierung in Webflow.....	2
Implementierung in HTML.....	5
3.3.2 – Labels & Anweisungen A	7
Implementierung in Webflow.....	7
Beschriftung	7
Positionierung	7
Notwendigkeit	7
Screenreader	8
Anforderungen kommunizieren.....	8
Implementierung in HTML.....	8
Beschriftung	8
Positionierung	8
Notwendigkeit	8
Screenreader	8
Anforderungen kommunizieren.....	9
3.3.3 – Fehlervorschläge AA.....	10
Implementierung in Webflow.....	10
Implementierung in HTML.....	13
3.3.4 – Fehlervermeidung AA	15
3.3.5 – Hilfe AAA.....	16
Implementierung	16
Punkt 2.4 Aus der Richtlinie 3.3.2 umsetzen.....	16
Weitere, theoretische Möglichkeiten	16
3.3.6 - Fehlervermeidung AAA	17
Implementierung	17
3.3.7 - Überflüssige Eingaben A	18
3.3.8 - Barrierefreie Authentifizierung AA.....	19
Implementierung in Webflow.....	19
Implementierung in HTML.....	20
3.3.9 - Barrierefreie Authentifizierung AAA.....	22

3.3.1 – Fehleridentifizierung A

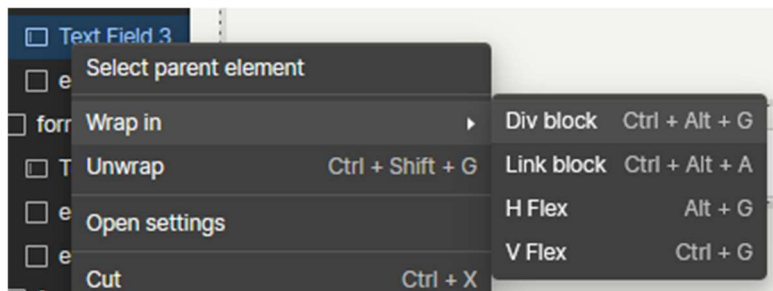
Wenn ein Eingabefehler automatisch erkannt wird, so wird das zugehörige Element identifiziert und der Fehler dem Nutzer in Textform beschrieben.



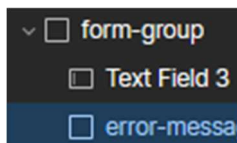
A screenshot of a web form. At the top is a text input field with the placeholder text "Name". Below the input field is a red error message: "Bitte Ihren Namen eingeben."

Implementierung in Webflow

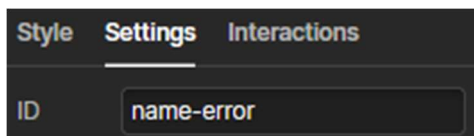
Jedes Textfeld wird einem div untergeordnet. Dieses div bekommt eine Klasse namens form-group.



Unter jedes Textfeld in dem form-group div ein div mit der Klasse error-message einfügen. Hier werden die Fehlermeldungen später angezeigt.



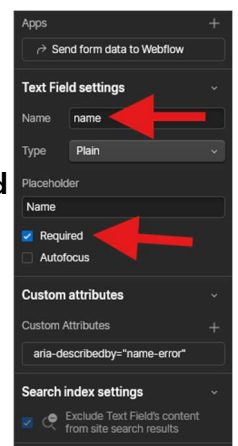
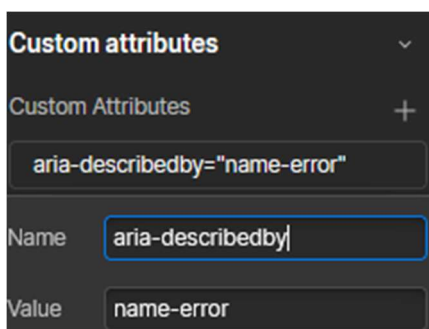
In den Einstellungen des Divs diesem dann eine zu seinem Textfeld zugehörige ID geben. Für die Fehlermeldung des Namens z. B. name-error.



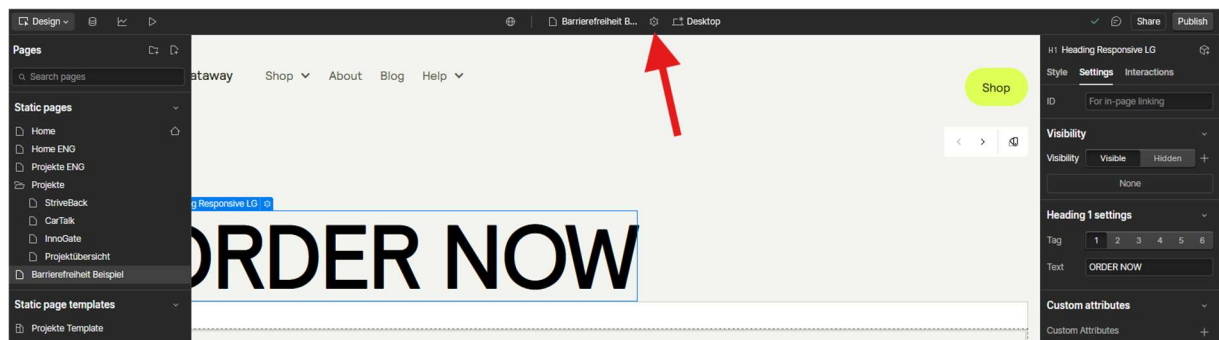
Zu dem **Textfeld** navigieren und in dessen Einstellungen ihm einen passenden Namen geben und die Checkbox "required" anwählen.

Unter der Sektion "Custom attributes" bei Name aria-describedby eintragen und bei Value den Namen des Divs, das die Fehlermeldung anzeigt.

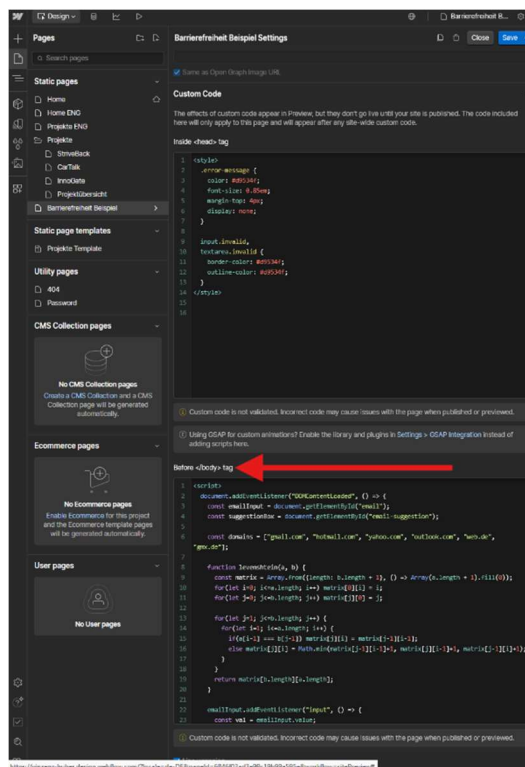
Das ist wichtig, damit Screenreader später den Fehler zusammen mit dem Textfeld vorlesen.



Die Einstellungen der Seite aufrufen.



Zur Sektion "Before </body> tag" navigieren.



Diesen Code einfügen:

```

<script>
document.addEventListener("DOMContentLoaded", function () {
    const form = document.querySelector("form");
    if (!form) return;
    form.setAttribute("novalidate", true);

    const fields = form.querySelectorAll("input, textarea");
    //Hier eigene Nachrichten für deine Elemente eintragen und die Elementnamen anpassen
    const customMessages = {
        "name": "Bitte Ihren Namen eingeben",
        "email": "Bitte eine gültige E-Mail-Adresse eingeben (z. B. max.mustermann@mailservice.com)",
        "message": "Bitte eine Nachricht eingeben.",
        "address": "Bitte eine Adresse eingeben (Format: Musterstr. 7)",
        "city": "Bitte eine Stadt eingeben.",
        "cardnumber": "Bitte eine gültige Kreditkartennummer eingeben (Format: 1234 5678 9123 4567)",
        "ccv": "Bitte die 3-stellige Sicherheitsnummer auf der Rückseite ihrer Kreditkarte eingeben (z. B. 557)",
        "day": "Bitte einen Tag als Nummer eingeben (nicht Montag sondern z. B. 3)",
        "year": "Bitte eine 4-stellige Jahreszahl eingeben (z. B. 2001)",
        "password": "Bitte ein Passwort eingeben",
        "password2": "Bitte das gleiche Passwort eingeben"
    };

    function showError(field, message) {
        const wrapper = field.closest(".form-group"); //Falls anderer Klassenname des unterordnenden Divs genutzt,
        hier ändern
        const error = wrapper?.querySelector(".error-message"); //Falls anderen Klassennamen des Fehlermeldungs-
        Divs genutzt, hier ändern
        if (error) {
            error.textContent = message;
            error.style.display = "block";
        }
        field.classList.add("invalid");
    }

    function clearError(field) {
        const wrapper = field.closest(".form-group");
        const error = wrapper?.querySelector(".error-message");
        if (error) {
            error.textContent = "";
            error.style.display = "none";
        }
        field.classList.remove("invalid");
    }

    function validateField(field) {
        const name = field.getAttribute("name");
        const value = field.value.trim();
        let valid = true;
        if (field.hasAttribute("required") && value === "") {
            showError(field, customMessages[name] || "Bitte füllen Sie dieses Feld aus.");
            valid = false;
        } else if (field.type === "email" && value !== "") {
            const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
            if (!emailRegex.test(value)) {
                showError(field, customMessages["email"]);
                valid = false;
            } else {
                clearError(field);
            }
        } else {
            clearError(field);
        }
        return valid;
    }

    fields.forEach(field => {
        field.addEventListener("blur", () => validateField(field));
    });

    form.addEventListener("submit", function (e) {
        e.preventDefault();
        let allValid = true;
        fields.forEach(field => {
            const isValid = validateField(field);
            if (!isValid) {
                allValid = false;
            }
        });
        if (allValid) {
            form.submit();
        }
    });
});
</script>

```

Implementierung in HTML

Die Struktur deines Formulars sollte so aussehen (alle Namen können ausgetauscht werden, aber so wie sie hier stehen, sind sie auch im Code):

```
<form>
  <div class="form-group">
    <input id="name" name="name" aria-describedby="name-error"
data-name="name" required="">
    <div id="name-error" class="error-message" style="display:
none;"></div>
  </div>
  <div class="form-group">
    <input id="email" name="email" aria-describedby="email-error"
data-name="email" required="">
    <div id="email-error" class="error-message" style="display:
none;"></div>
  </div>
</form>
```

Diesen Code einfügen:

```
<script>
document.addEventListener("DOMContentLoaded", function () {
  const form = document.querySelector("form");
  if (!form) return;
  form.setAttribute("novalidate", true);

  const fields = form.querySelectorAll("input, textarea");
  //Hier eigene Nachrichten für deine Elemente eintragen und die Elementnamen anpassen
  const customMessages = {
    "name": "Bitte Ihren Namen eingeben",
    "email": "Bitte eine gültige E-Mail-Adresse eingeben (z. B. max.mustermann@mailservice.com)",
    "message": "Bitte eine Nachricht eingeben.",
    "address": "Bitte eine Adresse eingeben (Format: Musterstr. 7)",
    "city": "Bitte eine Stadt eingeben.",
    "cardnumber": "Bitte eine gültige Kreditkartennummer eingeben (Format: 1234 5678 9123 4567)",
    "cvc": "Bitte die 3-stellige Sicherheitsnummer auf der Rückseite ihrer Kreditkarte eingeben (z. B. 557)",
    "day": "Bitte einen Tag als Nummer eingeben (nicht Montag sondern z. B. 3)",
    "year": "Bitte eine 4-stellige Jahreszahl eingeben (z. B. 2001)",
    "password": "Bitte ein Passwort eingeben",
    "password2": "Bitte das gleiche Passwort eingeben"
  };

  function showError(field, message) {
    const wrapper = field.closest(".form-group"); //Falls anderer Klassenname des unterordnenden Divs genutzt,
hier ändern
    const error = wrapper?.querySelector(".error-message"); //Falls anderen Klassennamen des Fehlermeldungs-
Divs genutzt, hier ändern
    if (error) {
      error.textContent = message;
      error.style.display = "block";
    }
    field.classList.add("invalid");
  }

  function clearError(field) {
    const wrapper = field.closest(".form-group");
    const error = wrapper?.querySelector(".error-message");
    if (error) {
      error.textContent = "";
      error.style.display = "none";
    }
    field.classList.remove("invalid");
  }

  function validateField(field) {
    const name = field.getAttribute("name");
    const value = field.value.trim();
    let valid = true;
    if (field.hasAttribute("required") && value === "") {
      showError(field, customMessages[name] || "Bitte füllen Sie dieses Feld aus.");
      valid = false;
    } else if (field.type === "email" && value !== "") {
      const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

```

```
        if (!emailRegex.test(value)) {
            showError(field, customMessages["email"]);
            valid = false;
        } else {
            clearError(field);
        }
    } else {
        clearError(field);
    }
    return valid;
}

fields.forEach(field => {
    field.addEventListener("blur", () => validateField(field));
});

form.addEventListener("submit", function (e) {
    e.preventDefault();
    let allValid = true;
    fields.forEach(field => {
        const isValid = validateField(field);
        if (!isValid) {
            allValid = false;
        }
    });
    if (allValid) {
        form.submit();
    }
});
});
</script>
```

3.3.2 – Labels & Anweisungen A

Es werden Labels oder Anweisungen bereitgestellt, wenn Inhalte eine Nutzereingabe benötigen.

Das Ziel ist hierbei deutliche Labels und Anweisungen für Eingabefelder. Labels müssen deutlich beschreibend sein und:

1. Deutlich einem Eingabefeld zugeordnet sein oder bei notwendigen Feldern signalisieren, dass sie noch nicht ausgefüllt sind
2. Sowohl als visueller Text als auch für Screenreader verständlich sein oder Anforderungen an die Eingabe deutlich kommunizieren

Implementierung in Webflow

Beschriftung

Labels sollten bereits durch Richtlinie 2.4.6 (Headings and Labels) deutlich beschreibend sein.

Positionierung

Um Labels deutlich einem Eingabefeld zuzuordnen, sollte es links neben oder direkt über dem Eingabefeld liegen. Hier kann sauberer Einsatz von Margin zu einer visuellen Gruppierung führen.

Date of Birth:

Country of origin:

Notwendigkeit

Ob eine Eingabe notwendig (required) ist, kann durch das Attribut "required" angezeigt werden. In Webflow dafür das Textfeld des Formulars auswählen und in den "Settings" Tab gehen. Dort kann man unter "Text Field settings" die Checkbox "Required" auswählen.

*Zusätzlich sollte der beschreibende Text des Feldes (Label o.ä.) deutlich kommunizieren, dass dieses Feld notwendig ist. Ein roter * ist hierbei nicht ausreichend.*

Text Field

Style Settings Interactions

ID Newsletter-Email

Visibility >

Form settings >

Text Field settings

Name Newsletter Email

Type Email

Placeholder

Enter your email

☒ Required

☐ Autofocus

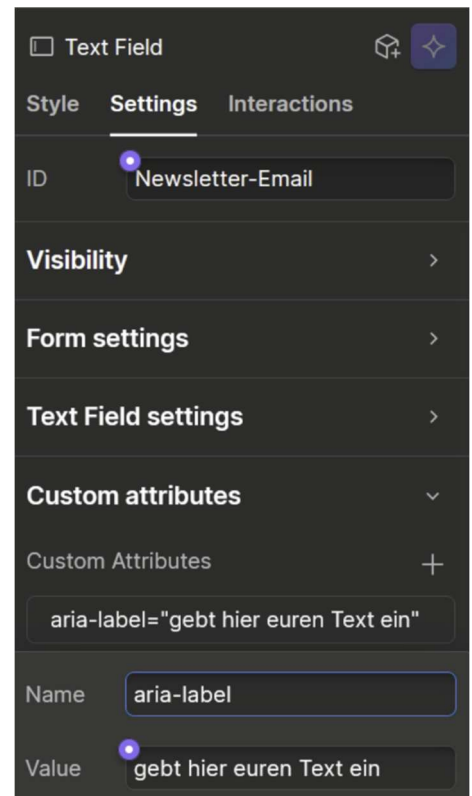
Screenreader

Um die das Inputfeld auch für Screenreader verständlich zu machen, sollte das Attribut "aria-label" beigefügt werden. In Webflow dafür Textfeld des Formulars auswählen und in den "Settings" Tab gehen. Dort kann man unter "Custom attributes" ein Attribut mit dem Namen aria-describedby anlegen. Der Value ist der Text für den Screenreader.

Anforderungen kommunizieren

Wenn eure Eingabe ein gewisses Format benötigt oder Anforderungen folgen muss, sollten diese durch Beschriftung deutlich kommuniziert sein.

- Zum Beispiel:
 - Date: (dd-mm-yyyy)
 - Passwort: (Das Passwort muss mindestens 2 Zahlen, 2 Klein- und Großbuchstaben und 2 Sonderzeichen enthalten.)



Implementierung in HTML

Beschriftung

Labels sollten bereits durch Richtlinie 2.4.6 (Headings and Labels) deutlich beschreibend sein.

Positionierung

Um Labels deutlich einem Eingabefeld zuzuordnen, sollte es links neben oder direkt über dem Eingabefeld liegen. Hier kann sauberer Einsatz von Margin zu einer visuellen Gruppierung führen.

Date of Birth:

Country of origin:

Notwendigkeit

Ob eine Eingabe notwendig (required) ist, kann durch das Attribut "required" angezeigt werden.

In HTML in den Container des Input Feldes gehen und ihm das Attribut "required" geben.

```
<div class="mb-3 col-8">
  <input name="email" required>
</div>
```

*Zusätzlich sollte der beschreibende Text des Feldes (Label o.ä.) deutlich kommunizieren, dass dieses Feld notwendig ist. Ein roter * ist hierbei nicht ausreichend.*

Screenreader

Um die das Inputfeld auch für Screenreader verständlich zu machen, sollte das Attribut "aria-label" beigefügt werden.

In HTML zum Input gehen und diesem das Attribut "aria-label", "aria-describedby" oder "aria-labelledby" geben.

```
<button type="button" aria-label="Close">X</button>

<form>
  <label for="fname">First name</label>
  <input aria-describedby="int2" autocomplete="given-name" id="fname" type="text">
  <p id="int2">Your first name is sometimes called your "given name".</p>
</form>

<div>
  <span id="timeout-label">
    <label for="timeout-duration">Extend time-out to</label>
  </span>
  <input type="text" size="3" id="timeout-duration" value="20"
    aria-labelledby="timeout-label timeout-duration timeout-unit">
  <span id="timeout-unit"> minutes</span>
</div>
```

Anforderungen kommunizieren

Wenn eure Eingabe ein gewisses Format benötigt oder Anforderungen folgen muss, sollten diese durch Beschriftung deutlich kommuniziert sein.

- Zum Beispiel:
 - Date: (dd-mm-yyyy)
 - Passwort: (Das Passwort muss mindestens 2 Zahlen, 2 Klein- und Großbuchstaben und 2 Sonderzeichen enthalten.)

3.3.3 – Fehlervorschläge AA

Erweitert Richtlinie 3.3.1

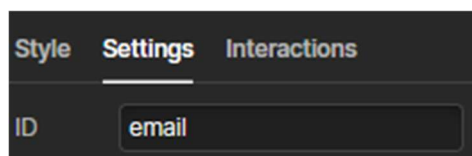
Wenn ein Eingabefehler automatisch erkannt wird und Korrekturvorschläge bekannt sind, dann werden diese dem Nutzer zur Verfügung gestellt, solange diese nicht die Sicherheit oder den Zweck des Inhalts beeinträchtigen würden.

Meinten Sie: beispiel@gmail.com ?

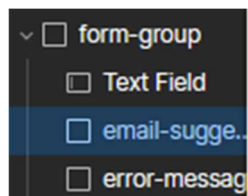
Implementierung in Webflow

Anmerkung: Damit der Code reibungslos funktioniert, treffen wir zusammen ein paar Vorkehrungen. Natürlich würde der Code auch mit anderen Namen der Elemente funktionieren, aber dafür müsste man ihn individuell anpassen. Wo das passiert, ist per Kommentare im Code vermerkt.

Dem E-Mail-Eingabefeld unter den Elementeneinstellungen die ID "email" geben.

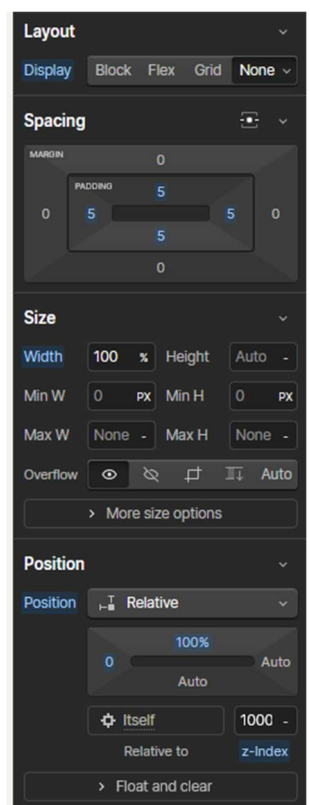
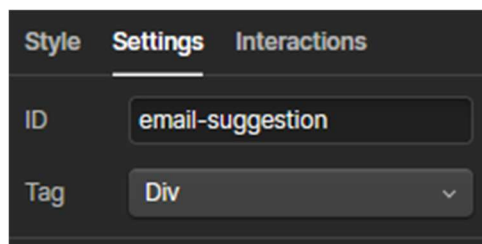


Ein neues Div unter dem Eingabefeld einfügen.

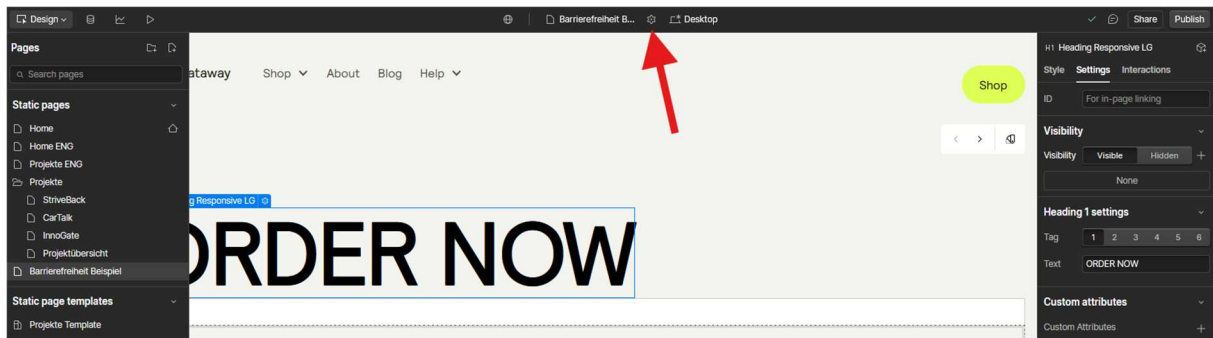


Display auf None stellen und die Position auf Relative mit einem z-index von 1000.

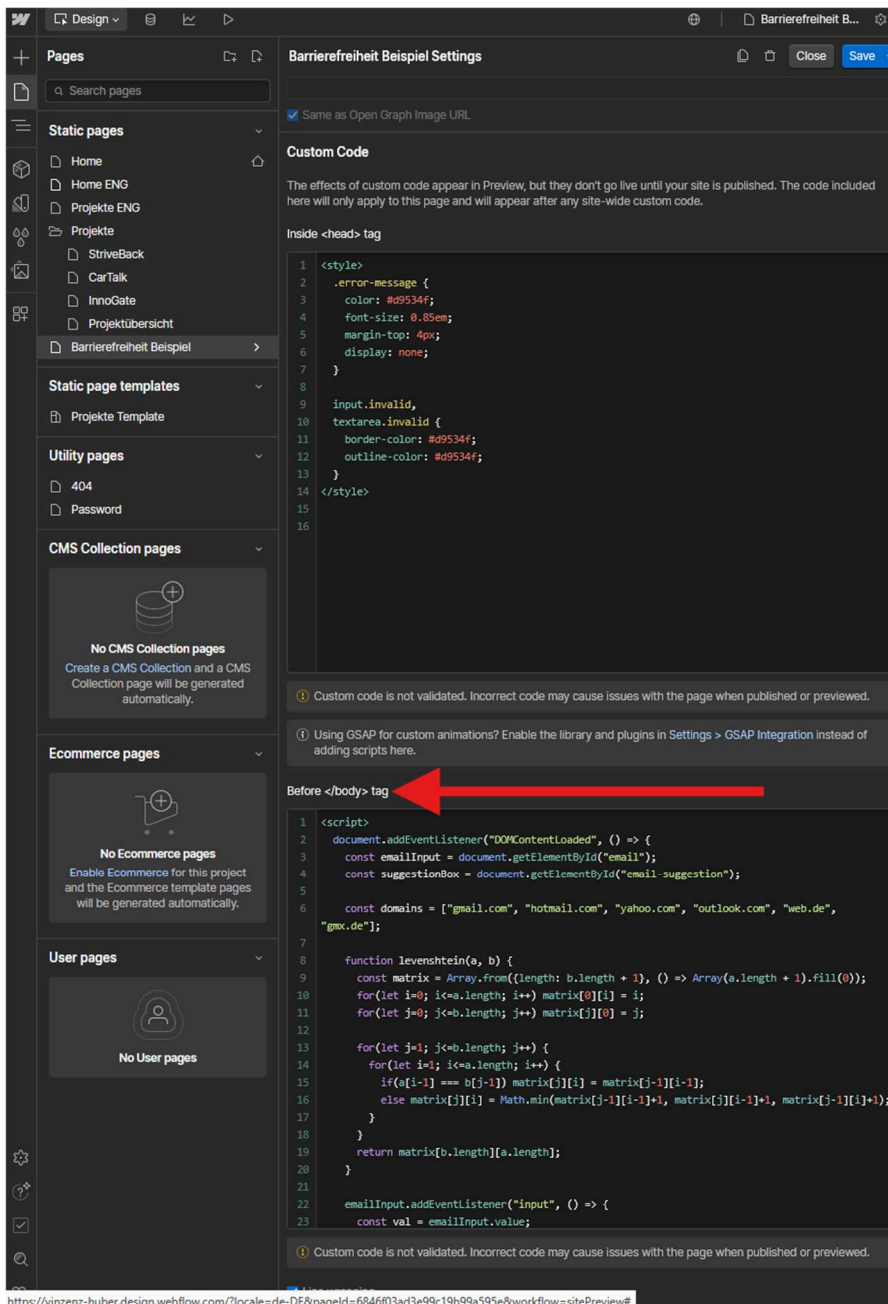
In den Einstellungen des Elements die ID als email-suggestion festlegen.



Die Einstellungen der Seite aufrufen.



Zur Sektion "Before \</body> tag" navigieren.



<https://vinzenz-huber.design.webflow.com/?locale=de-DE&paqid=6846f03ad3e99c19b99a595e&workflow=sitePreview#>

Diesen Code einfügen:

```
<script>
document.addEventListener("DOMContentLoaded", () => {
    const emailInput = document.getElementById("email"); //Hier den Namen deines E-Mail Feldes
    eintragen
    const suggestionBox = document.getElementById("email-suggestion"); //Hier den Namen deines
    Vorschlags-divs eintragen
    const domains = ["gmail.com", "hotmail.com", "yahoo.com", "outlook.com", "web.de", "gmx.de"];

    function levenshtein(a, b) {
        const matrix = Array.from({length: b.length + 1}, () => Array(a.length + 1).fill(0));

        for(let i=0; i<=a.length; i++) {
            matrix[0][i] = i;
        }
        for(let j=0; j<=b.length; j++) {
            matrix[j][0] = j;
        }
        for(let j=1; j<=b.length; j++) {
            for(let i=1; i<=a.length; i++) {
                if(a[i-1] === b[j-1]) {
                    matrix[j][i] = matrix[j-1][i-1];
                } else {
                    matrix[j][i] = Math.min(matrix[j-1][i-1]+1, matrix[j][i-1]+1, matrix[j-
1][i]+1);
                }
            }
        }
        return matrix[b.length][a.length];
    }

    emailInput.addEventListener("input", () => {
        const val = emailInput.value;
        suggestionBox.style.display = "none";
        const atIndex = val.indexOf("@");
        if (atIndex === -1) {
            return;
        }
        const localPart = val.slice(0, atIndex);
        const domainPart = val.slice(atIndex + 1).toLowerCase();
        if (!domainPart) return;
        let bestMatch = null;
        let bestDistance = Infinity;

        for (const domain of domains) {
            const dist = levenshtein(domainPart, domain);
            if (dist > 0 && dist < bestDistance && dist <= 2) {
                bestDistance = dist;
                bestMatch = domain;
            }
        }

        if (bestMatch) {
            const suggestion = `${localPart}@${bestMatch}`;
            suggestionBox.textContent = `Meinten Sie: ${suggestion} ?`;
            suggestionBox.style.display = "block";
        }
    });

    suggestionBox.addEventListener("click", () => {
        const text = suggestionBox.textContent;
        const suggestedEmail = text.match(/Meinten Sie: (.+) \?/)[1];
        emailInput.value = suggestedEmail;
        suggestionBox.style.display = "none";
        emailInput.focus();
    });

    emailInput.addEventListener("blur", () => {
        setTimeout(() => suggestionBox.style.display = "none", 200);
    });
});
</script>
```

Implementierung in HTML

Die Struktur des Formulars sollte so aussehen (alle Namen können ausgetauscht werden, aber so wie sie hier stehen, sind sie auch im Code):

```
<form>
  <div class="form-group">
    <input id="name" name="name" aria-describedby="name-error"
data-name="name" required="">
    <div id="name-error" class="error-message" style="display:
none;"></div>
  </div>
  <div class="form-group">
    <input id="email" name="email" aria-describedby="email-error"
data-name="email" required="">
    <div id="email-suggestion" style="display: none;"></div>
    <div id="email-error" class="error-message" style="display:
none;"></div>
  </div>
</form>
```

Diesen Code einfügen:

```
<script>
document.addEventListener("DOMContentLoaded", () => {
  const emailInput = document.getElementById("email"); //Hier den Namen deines E-Mail Feldes
  eintragen
  const suggestionBox = document.getElementById("email-suggestion"); //Hier den Namen deines
  Vorschlags-divs eintragen
  const domains = ["gmail.com", "hotmail.com", "yahoo.com", "outlook.com", "web.de", "gmx.de"];

  function levenshtein(a, b) {
    const matrix = Array.from({length: b.length + 1}, () => Array(a.length + 1).fill(0));

    for(let i=0; i<=a.length; i++) {
      matrix[0][i] = i;
    }
    for(let j=0; j<=b.length; j++) {
      matrix[j][0] = j;
    }
    for(let j=1; j<=b.length; j++) {
      for(let i=1; i<=a.length; i++) {
        if(a[i-1] === b[j-1]) {
          matrix[j][i] = matrix[j-1][i-1];
        } else {
          matrix[j][i] = Math.min(matrix[j-1][i-1]+1, matrix[j][i-1]+1, matrix[j-
1][i]+1);
        }
      }
    }
    return matrix[b.length][a.length];
  }

  emailInput.addEventListener("input", () => {
    const val = emailInput.value;
    suggestionBox.style.display = "none";
    const atIndex = val.indexOf("@");
    if (atIndex === -1) {
      return;
    }
    const localPart = val.slice(0, atIndex);
    const domainPart = val.slice(atIndex + 1).toLowerCase();
    if (!domainPart) return;
    let bestMatch = null;
    let bestDistance = Infinity;

    for (const domain of domains) {
```

```

        const dist = levenshtein(domainPart, domain);
        if (dist > 0 && dist < bestDistance && dist <= 2) {
            bestDistance = dist;
            bestMatch = domain;
        }
    }

    if (bestMatch) {
        const suggestion = `${localPart}@${bestMatch}`;
        suggestionBox.textContent = `Meinten Sie: ${suggestion} ?`;
        suggestionBox.style.display = "block";
    }
});

suggestionBox.addEventListener("click", () => {
    const text = suggestionBox.textContent;
    const suggestedEmail = text.match(/Meinten Sie: (.+) \?/)[1];
    emailInput.value = suggestedEmail;
    suggestionBox.style.display = "none";
    emailInput.focus();
});

emailInput.addEventListener("blur", () => {
    setTimeout(() => suggestionBox.style.display = "none", 200);
});
});
</script>

```

3.3.4 – Fehlervermeidung AA

Bezogen auf Recht, Finanzielles und Daten.

Für Webseiten, die rechtliche Verpflichtungen oder finanzielle Transaktionen für den Benutzer auslösen, die vom Benutzer kontrollierbare Daten in Datenspeichersystemen ändern oder löschen oder die Antworten auf Benutzertests übermitteln, trifft mindestens einer der folgenden Punkte zu:

- **Umkehrbar:** Versendete Daten sind reversibel
- **Geprüft:** Vom Benutzer eingegebene Daten werden auf Eingabefehler überprüft und der Benutzer erhält die Gelegenheit, diese zu korrigieren
- **Bestätigt:** Es gibt einen Mechanismus, um Informationen zu überprüfen, bestätigen und korrigieren, bevor sie endgültig abgesendet werden

Für die Umsetzung siehe [Implementierung von 3.3.6](#).

3.3.5 – Hilfe AAA

Es wird eine kontextsensitive Hilfe zur Verfügung gestellt. Also eine erweiterte Hilfestellung, wenn das Label nicht ausreichend Informationen zu den Anforderungen der Eingabe gibt.

Implementierung

Punkt 2.4 Aus der Richtlinie 3.3.2 umsetzen

Siehe [Richtlinie 3.3.2](#)

Weitere, theoretische Möglichkeiten

- Es steht eine Hilfeseite zur Verfügung.
- Ein Webassistent steht zur Verfügung
- Rechtschreibkorrektur und Eingabevervollständigung stehen zur Verfügung.

3.3.6 - Fehlervermeidung AAA

Bezogen auf alle Webseiten. Verschärft Richtlinie 3.3.4

Auf Webseiten, die den Benutzer auffordern, Informationen zu übermitteln, trifft mindestens einer der folgenden Punkte zu:

- **Umkehrbar:** Versendete Daten sind reversibel
- **Geprüft:** Vom Benutzer eingegebene Daten werden auf Eingabefehler überprüft und der Benutzer erhält die Gelegenheit, diese zu korrigieren
- **Bestätigt:** Es gibt einen Mechanismus, um Informationen zu überprüfen, bestätigen und korrigieren, bevor sie endgültig abgesendet werden

Implementierung

Diese findet mithilfe eines Modals statt. Die Grundstruktur dessen findet man auf der [zugehörigen Bootstrap-Seite](#). Über den Klick eines Buttons wird dann das Modal angezeigt.

Im Body des Modals zeigt man dann mit JavaScript die eingegebenen Informationen an:

```
<script>
document.getElementById("_mailcheck").innerHTML = "<p> Deine E-Mail: " +
email.value + "</p>"
document.getElementById("_passwordcheck").innerHTML = "<p> Deine Passwort:
" + password.value + "</p>"
</script>
```

Dann den "Abschicken" Button des Modals als Submit Button der Daten definieren.

3.3.7 - Überflüssige Eingaben A

Informationen, die zuvor durch den Nutzer ein-/angegeben wurden und im selben Prozess erneut eingegeben werden sollen, müssen entweder automatisch eingetragen werden oder vom Nutzer ausgewählt werden können. Dies trifft nicht zu, wenn:

- Das erneute Eingeben von Informationen essenziell ist (also den Inhalt fundamental ändern würde)
- Die Informationen benötigt werden, um die Sicherheit des Inhalts zu gewährleisten
- Zuvor eingegebene Informationen nicht mehr gültig sind

3.3.8 - Barrierefreie Authentifizierung AA

Ein kognitiver Funktionstest ist bei keinem Schritt in einem Authentifizierungsprozess nötig, außer dieser Schritt stellt eines der folgenden zur Verfügung:

- **Alternativen:** Andere Authentifizierungsmethoden, die sich nicht auf einen kognitiven Funktionstest verlassen
- **Mechanismen:** Es wird ein Mechanismus zur Verfügung gestellt, der den Nutzer beim Absolvieren des kognitiven Funktionstests unterstützt
- **Objekterkennung:** Der kognitive Funktionstest besteht darin, Objekte zu erkennen
- **Persönliche Inhalte:** Der kognitive Funktionstest besteht darin, nicht-Text Inhalte zu identifizieren, die der Nutzer der Website zur Verfügung gestellt hat

Das Ziel ist es hierbei, Tests zu kognitiven Funktionen zu vermeiden. Kognitive Tests sind dabei solche, in welchen der Nutzer sich Text merken, diesen transkribieren oder manipulieren muss. Für uns relevant sind hierbei Nutzernamen, Passwort und Captchas.

Implementierung in Webflow

Sich ein seitenspezifisches Passwort merken gilt als kognitiver Test. Damit der Nutzer dies nicht machen muss, müssen Passwort-Manager und/oder copy-and-paste auf das Formular angewendet werden können.

Besonders für Passwort-Manager ist es wichtig, dass Input Felder den richtigen "type" haben.

Um dies in Webflow umzusetzen, wählt ihr das Textfeld des Formulars aus und geht in den "Settings" Tab. Dort findet ihr unter "Text Field settings" das Attribut "Type". Wählt hier das Relevante Format aus.

Text Field

Style

Settings

Interactions

ID

Newsletter-Email

Action

(Webflow default)

Method

GET

POST

Apps

+

→ Send form data to Webflow

Text Field settings

▼

Name

Newsletter Email

Type

Email

▼

Implementierung in HTML

Sich ein seitenspezifisches Passwort merken gilt als kognitiver Test. Damit der Nutzer dies nicht machen muss, müssen Passwort-Manager und/oder copy-and-paste auf das Formular angewendet werden können.

Besonders für Passwort-Manager ist es wichtig, das Input Felder den richtigen "type" haben.

In HTML geht ihr in das Input-Feldes eures Formulars. Gebt dem Input das Attribut "type" und verleiht ihm einen Datentyp.

Es gibt 22 verschiedene Datentypen. Für uns relevant sind

Zahlen:

```
<input type="number"> (generisch)
<input type="date">
<input type="time">
<input type="tel">
```

Text:

```
<input type="text"> (generisch)
<input type="password">
```

```
<input type="email">  
<input type="url">
```

Weitere:

```
<input type="file">  
<input type="radio">  
<input type="checkbox">
```

Dies gilt auch für Buttons:

```
<input type="button"> (generisch)  
<input type="reset">  
<input type="submit">
```

3.3.9 - Barrierefreie Authentifizierung AAA

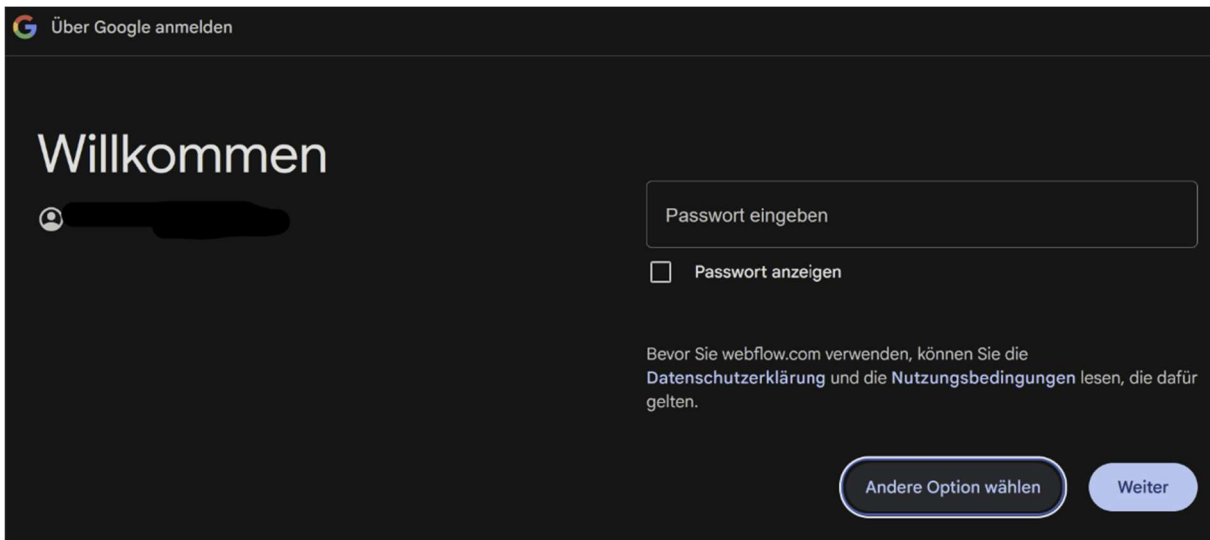
Verschärft Richtlinie 3.3.8

Ein kognitiver Funktionstest ist bei keinem Schritt in einem Authentifizierungsprozess nötig, außer dieser Schritt stellt eines der folgenden zur Verfügung:

- **Alternativen:** Andere Authentifizierungsmethoden, die sich nicht auf einen kognitiven Funktionstest verlassen
- **Mechanismen:** Es wird ein Mechanismus zur Verfügung gestellt, der den Nutzer beim Absolvieren des kognitiven Funktionstests unterstützt

Der entscheidende Unterschied zu 3.3.8 ist, dass die meisten Captchas hiermit ausgeschlossen werden. Viele basieren auf dem Bearbeiten, Merken, oder Verändern von Bild, Audio oder Video, welche durch die Punkte "Object Recognition" in 3.3.8 gedeckt wurden. Ein Gegenbeispiel sind ReCaptchas.

Somit beschränkt sich diese Richtlinie auf erweiterte Authentifizierung. Diese sollte bestmöglich über zwei-Faktor oder E-Mail-Authentifizierung möglich sein. Ein gutes Beispiel ist Google, bei welchen man über zwei-Faktor-Authentifizierung den gesamten Anmeldeprozess überspringen kann.



Auf "Andere Option wählen" gehen.

