Cook Book

Promillo

Für das Modul New Trends in IT und Management der digitalen Transformation
Provadis School of International Management and Technology
von

Ole Grundmann
Moritz Bosch
Ben Zelleröhr

email.ole : Matrikelnummer.Ole
Moritz.Bosch@stud-provadis-hochschule.de : D490
email.ben : Matrikelnummer.Ben

Inhaltsverzeichnis

1 Einleitung **3**

2 Technische Beschreibung **4**

2.1 Blackbox-Sicht 4

2.2 Komponenten 4

3 Cook **8**

1 Einleitung

2 Technische Beschreibung

2.1 Blackbox-Sicht

[Ich warte auf Ole, damit wir nicht Sachen doppelt schreiben]

2.2 Komponenten

Der Workflow ist aus mehreren gedanklichen Komponenten zusammengesetzt, welche wiederum aus kleineren Teilen bestehen.

Nutzerinformationen
Damit der Workflow laufen und eine Antwort liefern kann, muss der Nutzer relevante Informationen bereitstellen. Diese werden über ein Form abgefragt. Die Fragen sind:

- Wie viele Personen wollten trinken?
- Welche Präferenzen an Geschmack haben die Personen?
- Bilder über zur verfügung stehenden Getränken oder anderen Lebensmitteln.

Sollte eine Zahl < 1 eingegeben werden, wird die Anzahl an Personen automatisch auf 1 gesetzt. Anschließend kann der Nutzer auf den Button klicken, damit der Workflow gestartet wird. Im Anschluss wird der Nutzer gebeten zu warten, bis die nächste Komponente abgeschlossen ist.

Bildanalyse
Die Bilder werden an ein KI-Modell übergeben, welches die Bilder analysiert und versucht die enthaltenen Objekte zu erkennen. Wir nutzen dafür das Modell **GPT-4O-MINI** welches wir mit dem Workflow verbunden haben.

Der genutzte Prompt sieht wie folgt aus:

```
Erkenne auf jedem Bild genau eine Flasche
(die im Fokus bzw. im Vordergrund steht)
und gib folgende Angaben im exakt angegebenen Format aus:

{Genaue Bezeichnung des Getränks,z. B.
“Coca-Cola Zero Sugar”,“Gerolsteiner Mineralwasser Medium”,
```

```
“Volvic Mango”},{Maximalvolumen in ml},{Füllstand in %}
```

Die Ausgabe des Modells erfolgt im JSON-Format, für die weitere Nutzung der Daten im Workflow.

Datenvalidierung

Zur korrekten weiterverarbeitung der Daten wird der Nutzer gebeten, diese zu validieren bzw. zu korrigieren. Die Daten, welche von der vorherigen Komponente vom KI-Modell kommen, werden zunächst per JavaScript aufbereitet.

Zum genaueren Verständnis kann der folgende Code in Quellcode 1 betrachtet werden. Es ist zu beachten, dass teilweise n8n Spezifische Syntax verwendet wird.

Anschließend werden dem Nutzer sämtliche Daten im Formular angezeigt, welche er bei bedarf anpassen bzw. überschreiben kann. Wenn keine Veränderungen vorgenomen werden, wird der Wert vom KI-Modell übernommen. Die Darstellung erfolgt wie folgt:

```
{{JSON.stringify($items().map(item => item.json))}}
```

Daraufhin werden die Daten wieder per JavaScript verarbeitet, damit sie im nächsten Schritt von einem weiteren KI-Modell genutzt werden können. Quellcode 2 zeigt den Code, welcher dafür genutzt wird.

Rezeptvorschlag

Die validierten Daten, welche aus der vorherigen Komponente stammen, werden nun an ein weiteres KI-Modell gegeben. Genutzt wird folgender System Prompt:

```
Du bist MixMaster AI, ein hochqualifizierter virtueller Barkeeper
und Cocktailexperte. Deine Aufgabe ist es, auf Basis der vom Nutzer
angegebenen Präferenzen, der vorhandenen Zutaten und der
gewünschten Anzahl an Personen passende Cocktailrezepte vorzuschlagen.
Nutze dazu die Ressource „CocktailDB“ - eine Datenbank mit bekannten
Cocktailrezepten als Tool. Beachte dabei:

1. **Eingaben**
   - Präferenzen (z. B. Geschmack: süß, sauer, herb;
     Spirituosen-Favoriten; alkoholfrei; etc.)
   - Verfügbare Produkte (Liste von Spirituosen, Likören,
```

```
     Sirups, Säften, Bitters, Früchten, etc.)
   - Anzahl Personen, für die gemixt werden soll

2. **Verarbeitung**
   - Greife auf CocktailDB zu, um passende Rezepte zu finden.
   - Nutze nicht zwingend alle verfügbaren Zutaten - wähle die
     besten Kombinationen passend zu den Präferenzen.
   - Falls ein Rezept eine oder mehrere Zutaten enthält,
     die der Nutzer nicht vorrätig hat,
     führe diese als „fehlende Zutaten“ gesondert auf.

3. **Ausgabe**
   Für jedes Rezept erzeuge folgenden strukturierten Output:
   - **Cocktailname**
   - **Benötigte Zutaten** mit Menge pro Cocktail
   - **Zubereitung** (Schütteln, Rühren, Glasart, Garnitur)
   - **Fehlende Zutaten** (Liste der nicht verfügbaren Zutaten;
     falls keine fehlen, „-“)

Verhalte dich stets professionell, freundlich und präzise.
```

und folgender Prompt:

```
Erstelle 5 kreative aber auch gängige Cocktail-Vorschläge
auf Basis der folgenden Angaben:

**Präferenzen**: {{ $('Edit Fields').item.json.Preferenzen }}
**Anzahl Personen**: {{ $('Edit Fields').item.json.personen }}
**Verfügbare Produkte**: {{ $json.products }}

Die Inhalte von Verfügbare Produkte sind {Produkt},
{Maximales Volumen in Milliliter},{Aktueller Inhalt in Prozent}
```

Als Modell wird erneut **GPT-4O-MINI** genutzt. Zudem kommt eine verbundene MongoDB-Datenbank zum Einsatz. Darin können Rezepte gespeichert werden, auf welche das KI-Modell zugreifen kann. Zudem werden diese Rezepte vom Modell bevorzugt ausgegeben, wenn sie mit den vermittelten Daten eine Schnittmenge bilden. Dadurch können Preferenzen des Nutzers besser berücksichtigt werden, als auch die Halluzinationen des Modells reduziert werden.

Die Ausgabe des Modells erfolgt im JSON-Format, in folgendem Schema: Quellcode 3.

Ergebnispräsentation

Die Ergebnisse der vorherigen Komponente werden dem Nutzer präsentiert. Dafür werden die Daten erneut in JavaScript aufbereitet, damit sie in HTML Code umgewandelt werden. Der Quellcode 4 zeigt den Code, welcher dafür genutzt wird.

Anschließend wird der übergebene HTML Code wieder im Formular angezeigt:

```
{{ $json.html }}
```

Dazu kommen noch folgende CSS Styles, um die Darstellung zu verbessern: Quellcode 5.

3 Cook

Für die Reproduktion des Workflows mit identischer Einrichtung werden zwei Programme benötigt: nginx als Reverse Proxy sowie Docker zur Ausführung des n8n-Containers und der MongoDB-Datenbank.

Im bereitgestellten ZIP-Archiv befindet sich die Datei `nginx.conf`. In dieser Datei ist in Zeile 4 und in Zeile 13 der korrekte Domainname einzutragen. In Zeile 10 und in Zeile 11 sind der Pfad zum SSL-Zertifikat und der Pfad zum zugehörigen privaten Schlüssel anzupassen. Nach der Anpassung wird die Datei in das Verzeichnis `/etc/nginx/sites-available` kopiert und mit einem aussagekräftigen Namen, beispielsweise n8n, versehen. Anschließend wird im Verzeichnis `/etc/nginx/sites-enabled` ein symbolischer Link auf die Konfigurationsdatei erstellt:

```
ln -s /etc/nginx/sites-available/n8n /etc/nginx/sites-enabled/n8n
```

Damit die Änderungen wirksam werden, ist der nginx-Dienst neu zu laden:

```
nginx -s reload
```

Im entpackten ZIP-Archiv befindet sich ebenfalls die Datei `.env.example`. Diese ist in `.env` umzubenennen. In der Datei wird der Domainname eingetragen. Der Start von n8n und MongoDB erfolgt über den folgenden Befehl:

```
docker compose up
```

Standardmäßig wird dabei die aktuelle stabile Version von n8n verwendet. Die gewünschte Version kann in der Datei `docker-compose.yml` festgelegt werden. Die zuletzt getestete Version ist `1.105.3`.

Beim ersten Aufruf des n8n-Servers wird die Erstellung eines Benutzerkontos abgefragt. Nach erfolgreichem Abschluss der Registrierung erfolgt die Weiterleitung zur Übersichtsseite, die Abbildung 1 dargestellt ist.

In n8n selber müssen zunächst im „Credentials“ Reiter zwei Zugangsdaten angelegt werden. Für die MongoDB muss in den Feldern „Database“, „User“ und „Password“ die Daten eingetragen werden, die in der `.env` Datei als `MONGO_APP_DB`, `MONGO_APP_USER` und `MONGO_APP_PASS` festgelegt wurden.

Für die KI muss noch ein API Schlüssel von seinem OpenAI Benutzerkonto hinterlegt werden. Dazu wird der zweite Zugangsdatum erstellt.

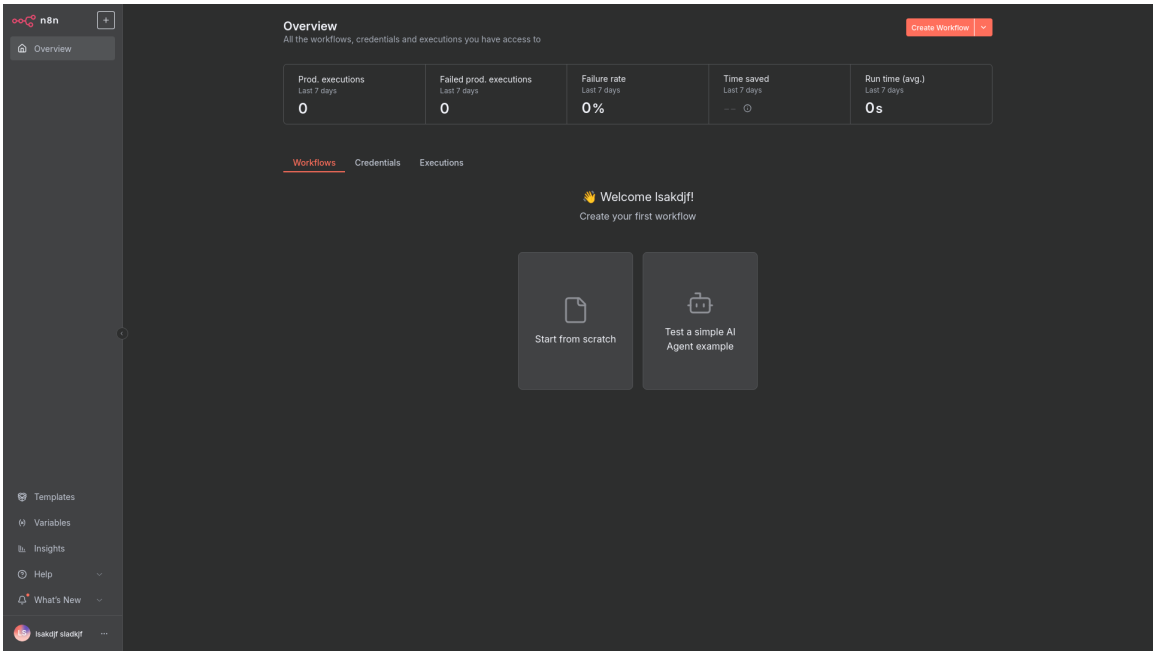

Abbildung 1: n8n Übersichtsseite

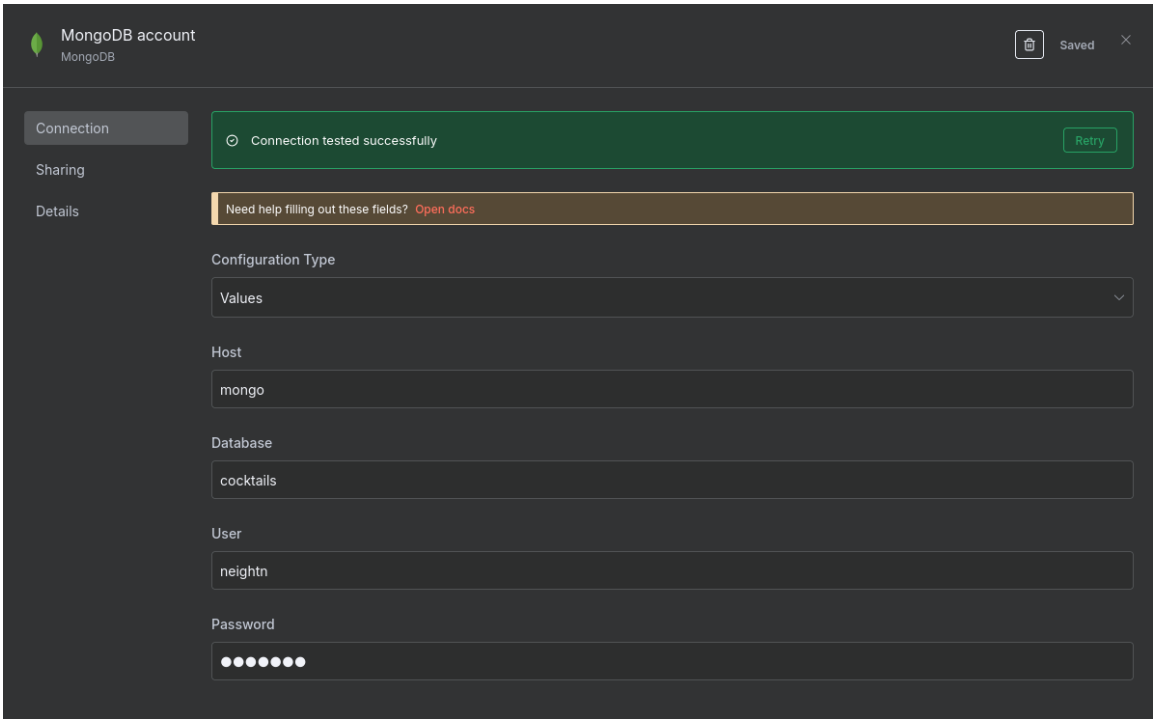

Abbildung 2: n8n MongoDB Zugangsdaten

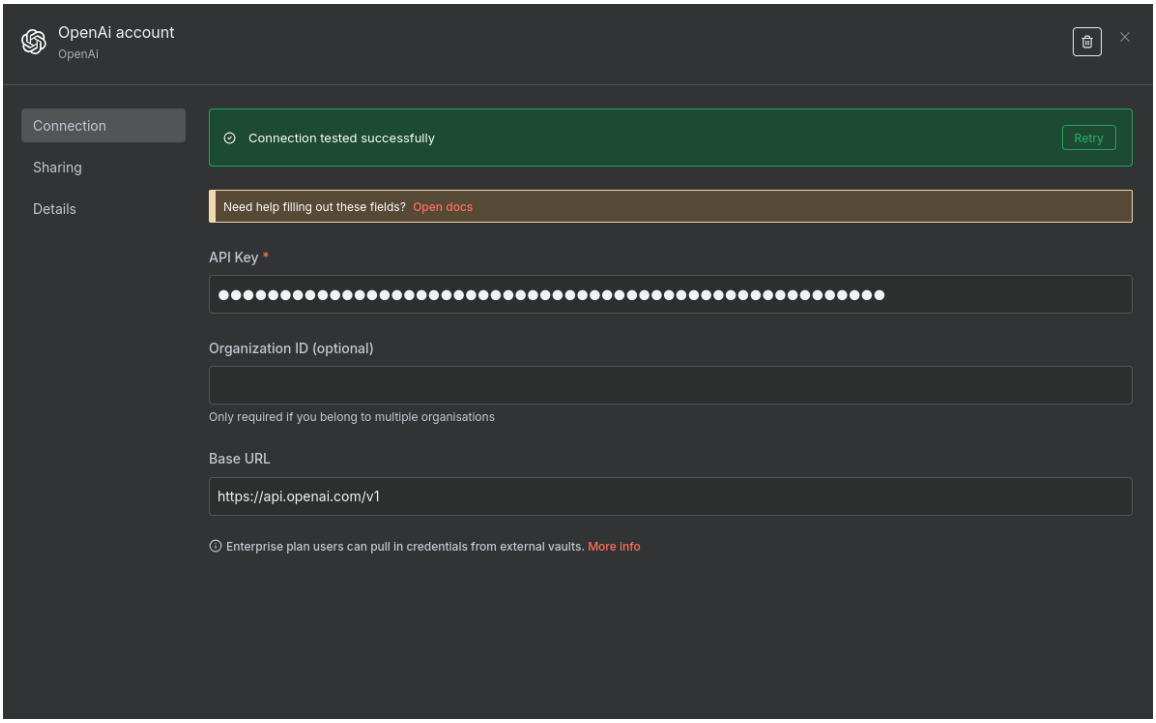

Abbildung 3: n8n OpenAI Zugangsdaten

Nachdem die beiden Zugangsdaten angelegt wurden, kann ein neuer Workflow erstellt werden. Im Bearbeitungsmodus angekommen, befindet sich open rechts im Dreipunktemenü die Auswahl ein vorhandenen Workflow von einer Datei zu importieren. Im ZIP-Archiv ist der Workflow unter dem Namen `Promillo.json` zu finden.

Die Nodes, die Zugangsdaten benötigen werden rot makiert, diese müssen mit Doppelklick einmal geöffnet und in der obenren linken Ecke geschlossen werden. Die Zugangsdaten verwenden zwar den standart Namen, werden von n8n aber mit eindeutigen IDs referenziert. Aufgrund des anlegens der Zugangsdaten werden im Workflow nicht die richtigen IDs referenziert, durch das Öffnen und Schließen werden die diese korriegiert.

Zum Schluss muss der Workflow in der open rechten Ecke gespeichert und aktiviert werden. Der Workflow ist damit unter der eigenen Domaine mit dem Pfad `/form/alcohol` erreichbar.

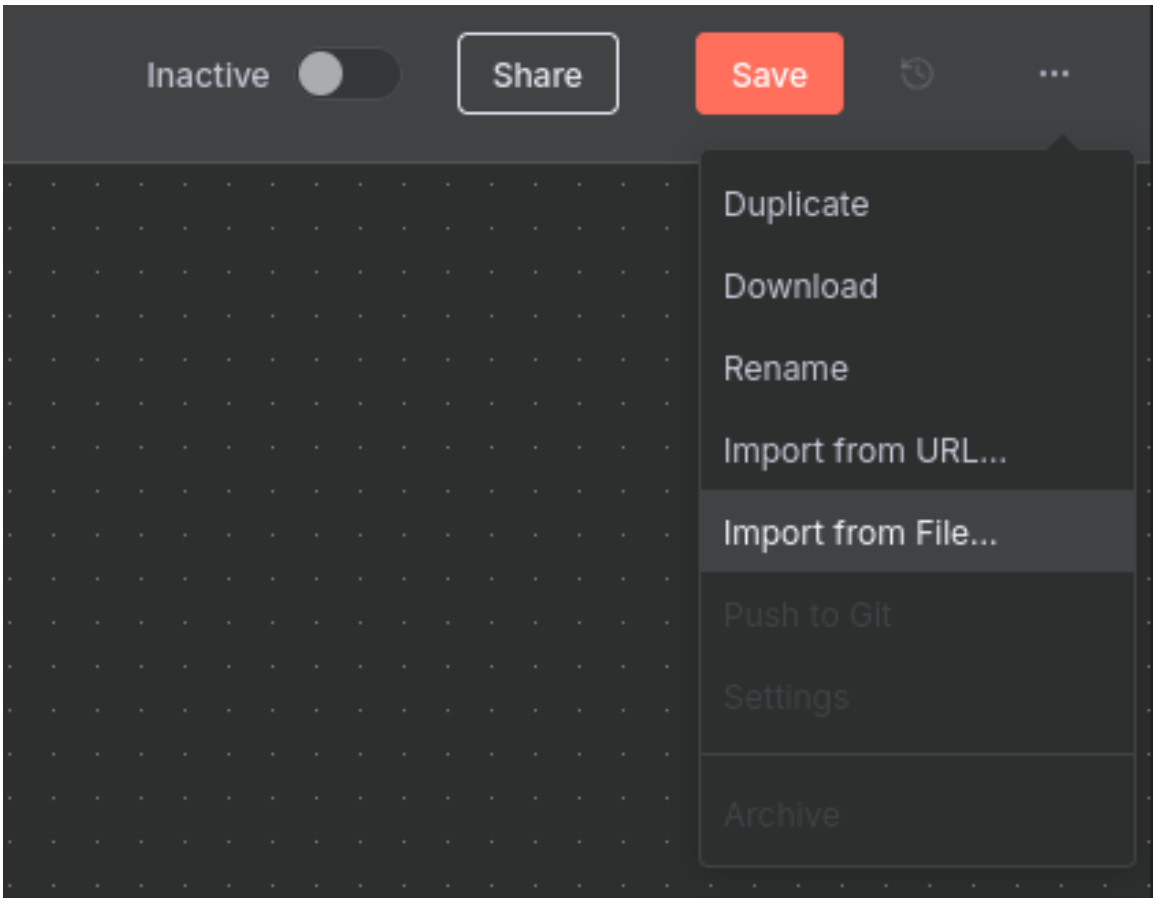

Abbildung 4: n8n Import Workflow

```
const rawData = $json["content"];
const lines = rawData.split('\n');
const formFields = [];

lines.forEach((line, index) => {
  const cleanedLine = line.replace(/^\d+\.\s*/, '');
  const braceMatches = [...cleanedLine.matchAll(/\{([^}]+)\}/g)];
  let rawValues = [];

  if (braceMatches.length > 0) {
    braceMatches.forEach(m => {
      rawValues.push(...m[1].split(',').map(v => v.trim()));
    });
  } else {
    rawValues = cleanedLine.split(',').map(v => v.trim());
  }

  if (rawValues.length < 3) return;

  const [name, max_amount_raw, current_amount_raw] = rawValues;
  const max_amount = parseInt(max_amount_raw, 10);
  const current_amount = parseInt(current_amount_raw, 10);

  if (isNaN(max_amount) || isNaN(current_amount)) return;

  formFields.push({
    fieldLabel: `name: ${name}, max amount: ${max_amount}, current amount: ${curren
    placeholder: 'enter if you want it changed',
    requiredField: false
  });
});

return formFields.map(field => ({ json: field }));
```

Quellcode 1: JavaScript Code zur Aufbereitung der Daten für die Validierung

```
// Loop over input items and add a new field called 'myNewField' to the JSON of eac
let out = { preferenzen: $('Edit Fields').first().json.Preferenzen, personen: $('Ed
for (const item of $input.all()) {
  for (const element in item.json) {
    if (element == "submittedAt" || element == "formMode") {
      delete item.json[element]
      continue
    }
    const placeholder = element
    const product = item.json[element] || placeholder
    if (!product) {
      continue
    }
    out.products.push(product)
  }
}
return out;
```

Quellcode 2: TODO

```
{
  "type": "object",
  "properties": {
    "Cocktails": {
      "type": "array",
      "items": {
        "type": "object",
        "required": [
          "Name",
          "Zutaten",
          "Beschreibung",
          "fehlende Zutaten"
        ],
        "properties": {
          "Name": {
            "type": "string"
          },
          "Beschreibung": {
            "type": "string"
          },
          "Zutaten": {
            "type": "array",
            "items": {
              "type": "object",
              "required": [
                "Zutat",
                "Menge"
              ],
              "properties": {
                "Zutat": {
                  "type": "string"
                },
                "Menge": {
                  "type": "string"
                }
              }
            }
          },
          "fehlende Zutaten": {
            "type": "array",
            "items": {
              "type": "string"
            }
          }
        }
      }
    }
  }
}
```

Quellcode 3: TODO

```
const cocktails = $input.first().json.output.Cocktails.slice(0, 5); // max. 5
const htmlEntries = cocktails.map(cocktail => {
  const zutaten = cocktail.Zutaten.map(z => `${z.Menge} ${z.Zutat}`).join(", ");
  const fehlende = cocktail["fehlende Zutaten"].join(", ");

  return `
    <div class="entry">
        <h3>${cocktail.Name}</h3>
        <p>${cocktail.Beschreibung}</p>
        <h4>Zutaten:</h4>
        <p>${zutaten}</p>
        <h4>Fehlende Zutaten:</h4>
        <p>${fehlende}</p>
    </div>
  `;
}).join("\n");

return [
  {
    json: {
      html: htmlEntries
    }
  }
];
```

Quellcode 4: TODO

```
body {
  font-family: 'Open Sans', sans-serif;
  background-color: #f0f0f0;
  margin: 0;
  padding: 2rem;
}
h1 {
  margin-bottom: 2rem;
}
.header div {
  background: white;
  border-radius: 0.5rem;
  box-shadow: 0 2px 6px rgba(0,0,0,0.1);
  padding: 1rem;
  max-width: 500px;
  margin: 1rem auto 1rem auto;
}

div h4 {
  margin-top: 1rem;
}
```

Quellcode 5: TODO