Backlog: 14326870

Backlog: Relaunch Website

Datum	Zeichen	Ihre Referenz	Seite	Druck vom
01.10.2025	Rudolf Machherndl		1 / 11	15.01.2026

In dieser Aktivität werden die im Rahmen der Phase "Backlog" lokalisierten Anforderungen dokumentiert. Im Rahmen der laufenden Projektumsetzung wird diese um neue Posten ergänzt, wenn eine Anforderung nicht unmittelbar im Call/Meeting gelöst werden kann.

Der Status der Posten wird nach folgenden Regeln gesetzt:

• Konzept: Anforderung wurde nicht mit Kunde abstimmt
• Offen: Mit Kunde abgestimmt - Bearbeitung offen
• Erledigt: Posten wurde entweder direkt umgesetzt oder die Lösung in einem CR dokumentiert
• Evidenz: Umsetzung erfolgt in späteren Phase
• Storno: Posten wird nach Rücksprache mit Kunden nicht umgesetzt

01 CMA - Content Management Api

01.A Produktnavigation

Als User möchte ich einfach zu den verschiedenen Produkten navigieren können um eine gute Übersicht über die Produktpalette erhalten.

[Erfolgskriterien]

• Umsetzung laut Screendesign
• Es kann ein Link auf eine andere Seite hinterlegt werden
• Es können bis zu 3 Ebenen abgebildet werden
• Es können beliebig viele Einträge definiert werden
• Die hinterlegten Seiten werden direkt unterhlab vom Contentblock angezeigt
• Die URL soll wie folgt gebildet werden: example.com/prdoukte/. mit KLick auf eine Produktkategire wird auf der selben Seite die URL geladen z.B example.com/produkte/dickenmessgeräte
• Die gewünschte Seite wird direkt unterhlab von der Navigation angezeigt

Aktion: ☐ ARMA: Bearbeiten | OFFEN

01.A.A ProductNavigation CB

[Beschreibung]

Es wird eine neue Komponente (Contentblock) namens Produktbaum-Navigation erstellt. Diese Komponente kann in der Dynamic Zone jeder beliebigen Seite (z.B. einer ProduktSeite oder einer allgemeinen Seite) platziert werden, um die Produkt-Hierarchie darzustellen.

[Parameter]

Überschrift: (Text, optional) Eine optionale Überschrift für den Navigationsblock, z. B. "Unsere Produkte" oder "Weitere Themen".
Startpunkt: (Relation, One-to-One mit 'ProduktSeite') Definiert die Wurzelseite des anzuzeigenden Produktbaums. Alle untergeordneten Seiten dieses Startpunktes werden in der Navigation gerendert.

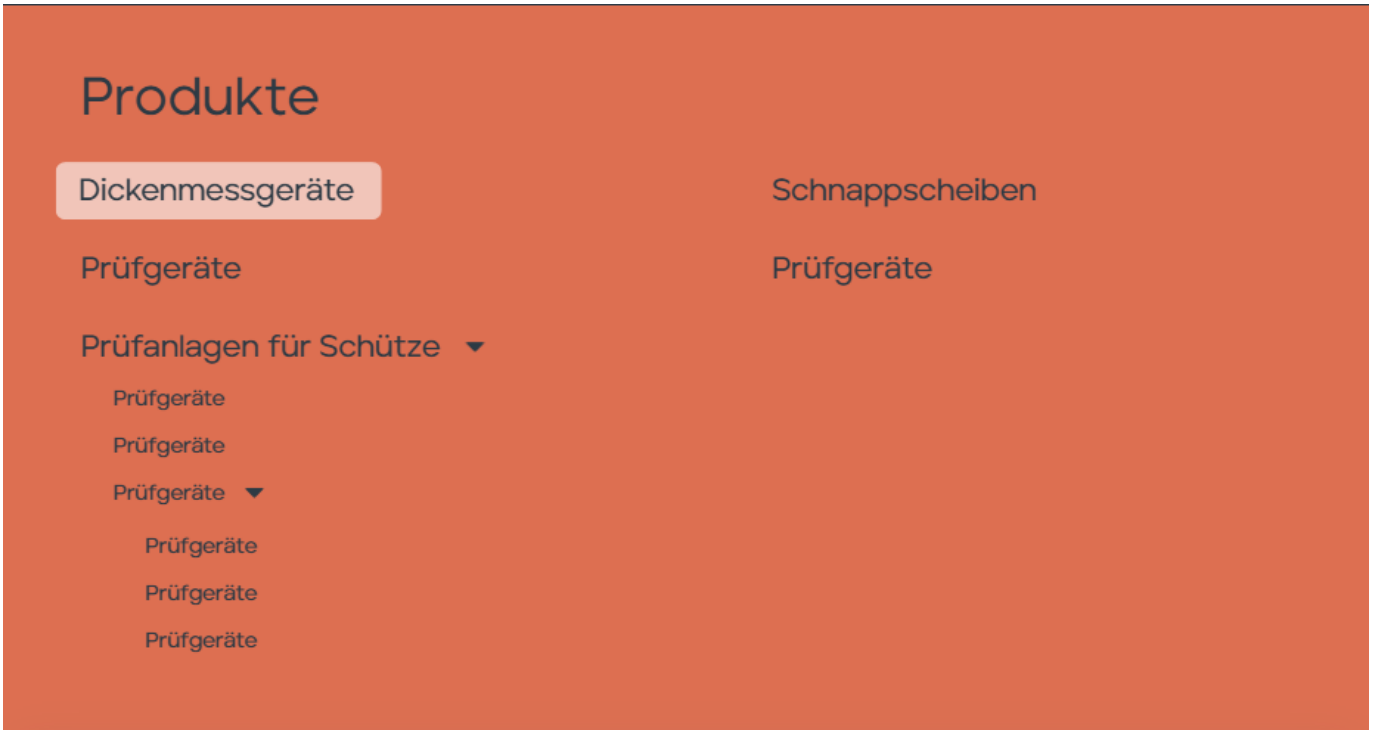
Anzeigetiefe: (Zahl, optional) Eine Zahl, die angibt, wie viele Ebenen des Baumes standardmäßig angezeigt werden sollen.

[Details]

• Wenn diese Komponente auf einer Seite platziert wird, rendert das Frontend eine Navigation, die bei der als Startpunkt ausgewählten ProduktSeite beginnt und alle ihre untergeordneten Seiten hierarchisch anzeigt.

[Akzeptanzkriterien]

• Die Komponente Produktbaum-Navigation kann in jeder Dynamic Zone in Strapi hinzugefügt werden.
• Ein Redakteur kann eine ProduktSeite als Startpunkt für die Navigation auswählen.
• Das Frontend zeigt die Navigationsstruktur korrekt als Baum oder verschachtelte Liste an, basierend auf der Parent-Kind-Beziehung der ProduktSeiten.
• Die Links in der Navigation verweisen auf die korrekten URLs der jeweiligen ProduktSeiten.
• Der Navigationsblock ist responsiv.



Aktion: ☐ ELRI: Bearbeiten | OFFEN

01.A.B CollectionType > ProduktPage

[Beschreibung]

Es wird ein CollectionType 'produktPage' eingerichtet.

[Details]

Der Collection-Type ProduktSeite besteht aus einer "Static Zone" für die Kerndaten und einer "Dynamic Zone" für flexible Inhaltsblöcke.

»»» Static Zone Felder

• Titel: (Text, erforderlich) Der Haupttitel der Seite, z. B. "Laptops" oder "Ultrabook XYZ".
• Slug: (UID, erforderlich) Der URL-freundliche Bezeichner, der aus dem Feld "Titel" generiert wird.
• ParentSeite: (Relation, One-to-One mit 'ProduktSeite') Dies ist eine selbstverweisende Beziehung. Sie legt die übergeordnete Seite im Produktbaum fest. Eine Seite ohne ParentSeite ist ein Hauptelement (Wurzel).
• MetaInformationen: (Component) Eine Komponente für SEO-relevante Daten wie Meta-Titel und Meta-Beschreibung.
• publishAtDate: (Datetime) Das Datum, ab dem der Inhalt öffentlich sichtbar ist.

»»» Dynamic Zone

• Diese Zone ermöglicht es Redakteuren, verschiedene vordefinierte Komponenten (Content-Blöcke) hinzuzufügen, um den Seiteninhalt flexibel zu gestalten. Hier wird später auch die Komponente 'Produktbaum-Navigation' eingefügt werden können.

[Aktzeptanzkriterien]

• Im Backend kann eine ProduktSeite erstellt und einer übergeordneten ProduktSeite zugewiesen werden, um eine Baumstruktur zu erzeugen.
• Die Seite ist über ihre URL (/produkt/{slug}) aufrufbar.
• Seiten werden erst ab dem publishAtDate veröffentlicht.
• Die Redaktion kann der Seite in der Dynamic Zone Inhaltsblöcke hinzufügen, entfernen und deren Reihenfolge ändern.
• Die Seite ist responsiv.

Aktion: ☐ ELRI: Bearbeiten | OFFEN

01.A.C Table CB

Es wird ein Contentblock "Table" eingerichet.

[Details]

»»» Parameter

• heading
• headingTag
• text (richtext)
• headingColor
• backgroundColor
• fontColour
• showOnSubpages Yes/No
• contentWidth
• tableAlternatingBackgroundColor
• tableAlternatingFontColor
• columnHeadingTag
• openNewTabOnLink (siteDefault/alwaysNewTab/alwaysCurrentTab)

Es können beliebig viele Spalten angelegt werden.

Eine Spalte besteht aus folgenden Parameter:

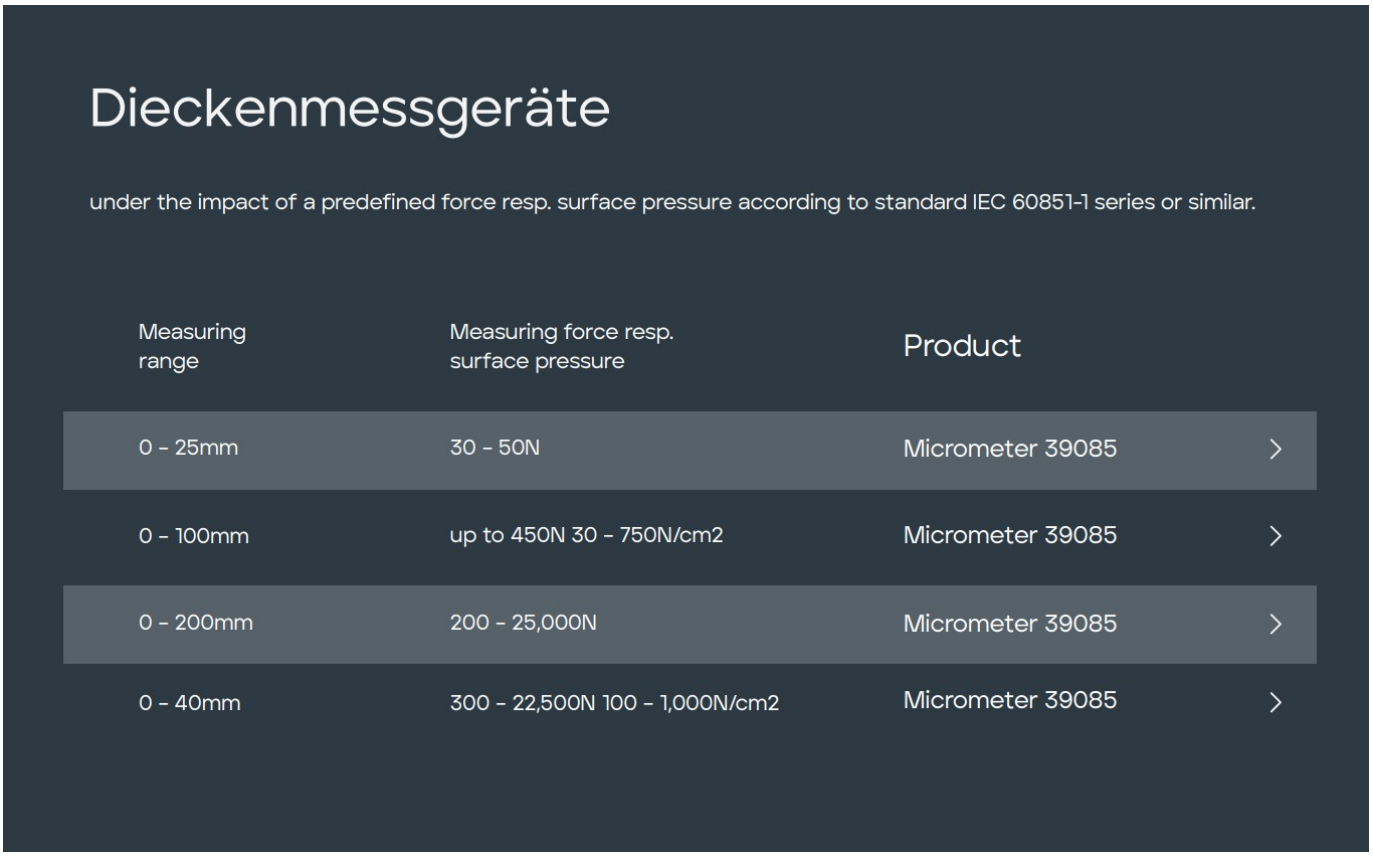
• Titel (Zwingend)
• FontStyling (Fett/Kursiv/normal usw)

Es können beliebig viele Zeilen angelegt werden. Eine Zeile hat folgende Parameter:

• Je Spalte kann ein Text erfasst werden (string)
• FontStyling (Fett/Kursiv/normal usw)
• Je Spalte kann eine URL/SLUG hinterlegt werden (optional)
• Je Zeile kann optional ein URL/SLUG hinterlegt werden

[Akzeptanzkriterien]

• Der Contentblock ist im Strapi vorhadnen und kann von der Redaktion gewartet werden
• Mit Klick auf einen Eintrag mit URL/SLUG wird die entsprechende Seite geladen
• Links für Spalten können nur mit direkte, Klick auf die entsprechende Spalte geladen werden
• Wenn man auf die Zeile klickt wird die URL/SLUG für die Zeile geladen
• openNewTabOnLink Yes/No



Aktion: ☐ ELRI: Bearbeiten | OFFEN

01.B Produktanfrage

[User Story]

Als Redakteur möchte ich in der Lage sein, auf jeder Seite vollständig benutzerdefinierte Formulare zu erstellen. Ich möchte die Struktur des Formulars, einschließlich der Gruppierungen, der einzelnen Felder (Typ, Name, Pflichtfeld) und der finalen JSON-Struktur für den API-Aufruf, direkt in Strapi definieren können.

[Erfolgskriterien]

• Umsetzung laut Screendesign

Aktion: ☐ ARMA: Bearbeiten | OFFEN

01.B.A ModularContactRequestCB

[Beschreibung]

Es wird eine neue, hochflexible Komponente (Contentblock) namens "ModularContactRequestCB" erstellt. Diese Komponente dient nicht dazu, ein festes Formular abzubilden, sondern gibt Redakteuren die Werkzeuge an die Hand, um beliebige Formulare direkt aus dem Backend zu konfigurieren. Das Frontend rendert dynamisch das Formular basierend auf dieser Konfiguration und generiert beim Absenden eine definierte JSON-Struktur.

»»» Aufbau der Komponenten (Strapi-Struktur)

Der Baukasten besteht aus einer Hauptkomponente, die wiederholbare Komponenten für Feldgruppen und Auswahlgruppen enthält.

»» Hauptebene: Komponente 'Formular-Baukasten'

Dies sind die globalen Einstellungen für das gesamte Formular.

• Interne Bezeichnung: (Text) Ein administrativer Name für dieses spezifische Formular (z.B. "Produktanfrage Laptop-Modell X").
• Formular-Überschrift: (Text, optional) Die Hauptüberschrift, die über dem Formular angezeigt wird (z.B. "Kontaktieren Sie uns").
• API-Endpunkt: (Text, erforderlich) Die vollständige URL, an die die Formulardaten per POST-Request gesendet werden sollen.
• Button-Text: (Text, erforderlich) Beschriftung für den Sende-Button (z.B. "Anfrage absenden").
• Datenschutzhinweis: (Richtext, erforderlich) Text, der angezeigt wird und einen Link zur Datenschutzerklärung enthalten muss.
• Feld-Gruppen: (Wiederholbare Komponente: `Feld-Gruppe`) Hier kann der Redakteur eine oder mehrere Gruppen von Eingabefeldern definieren.
• Auswahl-Gruppen: (Wiederholbare Komponente: `Auswahl-Gruppe`) Hier kann der Redakteur eine oder mehrere Gruppen von Checkboxen definieren (z.B. für Zubehör).

»» Ebene 2: Wiederholbare Komponente 'Feld-Gruppe'

Jede Feld-Gruppe repräsentiert einen logischen Abschnitt im Formular.

• Gruppen-Überschrift: (Text, optional) Eine optionale Überschrift für diesen Abschnitt (z.B. "Ihre Adressdaten").
• JSON-Object für Gruppe: (Text, erforderlich) Der Key, der im finalen JSON-Objekt für diese Gruppe verwendet wird (z.B. `addressDetails`). Muss eindeutig und ohne Sonderzeichen sein.
• Formularfelder: (Wiederholbare Komponente: `Formularfeld`) Eine Liste der Eingabefelder innerhalb dieser Gruppe.

»» Ebene 3: Wiederholbare Komponente 'Formularfeld' (innerhalb von 'Feld-Gruppe')

Dies ist die Definition eines einzelnen Eingabefeldes.

• Beschriftung (Label): (Text, erforderlich) Der Text, der dem Benutzer als Name des Feldes angezeigt wird (z.B. "Ihre E-Mail-Adresse").
• JSON-Subobject für Feld: (Text, erforderlich) Der Key für dieses Feld innerhalb des übergeordneten Gruppen-Objekts (z.B. `email`).

• Feldtyp: (Enumeration/Auswahl, erforderlich) Definiert die Art des Eingabefeldes. Optionen:
· Text (Standard-Input)
· E-Mail
· Telefon
· Zahl
· Postleitzahl
· URL
· Textbereich (Textarea)
· Datum
· Platzhalter: (Text, optional) Der Platzhaltertext im Eingabefeld.
· Pflichtfeld: (Boolean) Gibt an, ob dieses Feld ausgefüllt werden muss.

»» Ebene 2: Wiederholbare Komponente 'Auswahl-Gruppe' (für Checkboxen/Zubehör)

• Gruppen-Überschrift: (Text, erforderlich) Die Überschrift für den Checkbox-Bereich (z.B. "Wählen Sie Ihr Zubehör").
• API-Schlüssel für Gruppe: (Text, erforderlich) Der Key, der im finalen JSON-Objekt für diese Auswahl verwendet wird (z.B. `selectedAccessories`).
• Auswahloptionen: (Wiederholbare Komponente: `Auswahloption`) Eine Liste der verfügbaren Checkbox-Optionen.

»» Ebene 3: Wiederholbare Komponente 'Auswahloption' (innerhalb von 'Auswahl-Gruppe')

• Beschriftung (Label): (Text, erforderlich) Der Text neben der Checkbox (z.B. "Zusätzlicher Akku").
• Wert für API: (Text, erforderlich) Der Wert, der im API-Aufruf für diese Option gesendet wird (z.B. `extra_battery`).

»»» Funktionale Anforderungen (Frontend)

• Dynamisches Rendern: Das Frontend muss die von Strapi gelieferte Konfiguration interpretieren und das Formular dynamisch aufbauen, inklusive aller Gruppen, Felder (mit korrektem `type`-Attribut) und Checkboxen.
• Validierung: Das Frontend muss clientseitige Validierungen durchführen:
· Prüfung auf `Pflichtfeld`.
· Typ-spezifische Validierung (z.B. gültiges E-Mail-Format für den Feldtyp `E-Mail`).
• JSON-Generierung: Beim Absenden muss das Frontend ein JSON-Objekt dynamisch erstellen, das exakt der in Strapi definierten Struktur (`API-Schlüssel für Gruppe` und `API-Schlüssel für Feld`) entspricht. Beispiel-JSON-Struktur:

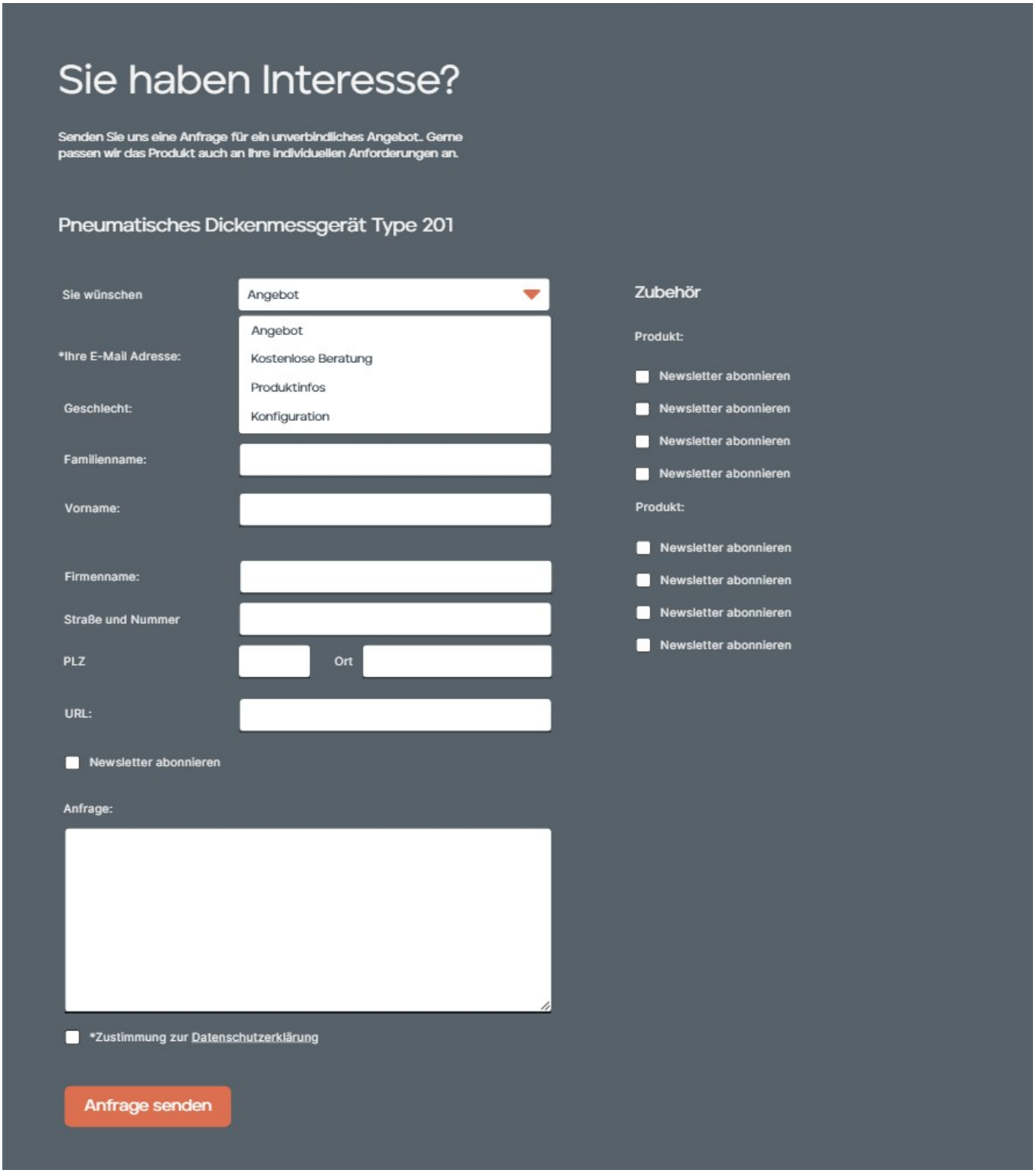
{: {
"addressDetails": {
"firstName": "Max",
"lastName": "Mustermann",
"email": "max@example.com"
},
"productInquiry": {
"productNumber": "XYZ-123",
"message": "Ich habe eine Frage."
},
"selectedAccessories": [
"extra_battery",
"protective_case"
]
} :}

4. Datenversand: Das generierte JSON-Objekt wird per `POST`-Request an den in Strapi definierten `API-Endpunkt` gesendet.
5. Feedback: Der Benutzer erhält eine klare Erfolgs- oder Fehlermeldung nach dem Absendeversuch.

[Akzeptanzkriterien]

• Ein Redakteur kann ein Formular mit mindestens zwei `Feld-Gruppen` und mehreren `Formularfeldern` unterschiedlichen Typs erstellen.
• Ein Redakteur kann eine `Auswahl-Gruppe` mit mehreren `Auswahloptionen` konfigurieren.
• Das im Frontend gerenderte Formular entspricht exakt der im Backend konfigurierten Struktur und den Bezeichnungen.
• Die clientseitige Validierung für Pflichtfelder und Feldtypen funktioniert wie erwartet und verhindert das Absenden bei Fehlern.

• Beim Absenden wird ein JSON-Objekt erzeugt, dessen Schlüssel exakt den definierten `API-Schlüsseln` entsprechen.
• Der `POST`-Request wird erfolgreich an den konfigurierten `API-Endpunkt` gesendet.
• Das gesamte Formular ist responsiv und auf mobilen Geräten bedienbar.



Aktion: Vorgetragen

01.B.B CDA > POST /cda/contactRequest

[Beschreibung]

Es wird ein API-Endpunkt erstellt, der die Daten des neuen "Formular-Baukasten"-Moduls entgegennimmt. Da das Frontend das Formular basierend auf der Strapi-Konfiguration dynamisch rendert, empfängt dieser Endpunkt ein JSON-Objekt, dessen Struktur durch die im CMS definierten Gruppenschlüssel bestimmt ist.

Dieser Endpunkt dient als generischer oder spezifischer Eingangskanal (je nach Konfiguration im CMS) für Kontakt- und Produktanfragen.

[Endpoint]

POST www.partner-success.at/cda/contactRequest

» Request Body

Der Body ist dynamisch, basierend auf der CMS-Konfiguration. Für diesen Task gilt die unten definierte Struktur (siehe Abschnitt "Request Payload") als Referenzimplementierung.

» Response

Content-Type: JSON
• Bei Erfolg: HTTP 202 Accepted: {"message": "Anfrage erfolgreich empfangen."}
• Bei Fehler: HTTP 400 Bad Request: "Ungültiges JSON Format."
• Bei Fehler: HTTP 422 Unprocessable Entity: "Validierung fehlgeschlagen (z.B. Pflichtfelder fehlen, falls serverseitig prüfbar)."
• Bei Fehler: HTTP 500 Internal Server Error.

[Details]

»»» Logik im API Handler

• Validierung: Prüfung, ob der Body ein gültiges JSON-Objekt ist und nicht leer ist.
• Verarbeitung:
· Es erfolgt keine direkte Speicherung in der PostgreSQL Datenbank durch den API-Handler selbst.
· Es wird eine REDIS-Nachricht erstellt, um die Anfrage asynchron weiterzuverarbeiten (z.B. E-Mail-Versand an Support, Speicherung im CRM).
• Antwort: Sofortige Rückmeldung an das Frontend (202 Accepted), sobald die Redis-Nachricht publiziert wurde.

»»» Redis Message Struktur

Die eingehende Payload wird in das payload-Feld des Redis-Events gewrappt.

DTO: contact_request

Action: create

[Akzeptanzkriterien]

• Der Endpunkt ist unter POST /cda/contactRequest erreichbar.
• Der Endpunkt akzeptiert die definierte JSON-Struktur (nested Objects & Arrays).
• Bei erfolgreichem Empfang wird HTTP 202 zurückgegeben.
• Eine REDIS-Nachricht wird korrekt laut untenstehender Struktur erstellt und versendet.
• Fehlerhafte JSON-Strukturen führen zu einem HTTP 400/422.
• Dokumentation ist im Swagger vorhanden.

Aktion: Vorgetragen

01.B.B.A CDA > POST /cda/contactRequest > Request

```
{: {
"addressDetails": {
"firstName": "string",        // Pflichtfeld laut CMS
"lastName": "string",         // Pflichtfeld laut CMS
"email": "string"             // Validierung auf E-Mail Format
},
"productInquiry": {
"productNumber": "string",
"message": "string"           // Textarea
},
"selectedAccessories": [        // Array aus der Auswahl-Gruppe
"extra_battery",
"protective_case"
],
"timestamp": "string"           // ISO 8601 (optional, falls vom Frontend gesendet)
} :}
```

01.B.B.B CDA > POST /cda/contactRequest > Request Validierung

[Übersicht]

Der Contact-Service implementiert eine umfassende Validierung eingehender Anfragen, um die Systemsicherheit und Performance zu gewährleisten. Die Validierungsparameter können flexibel über Umgebungsvariablen konfiguriert werden.

[Konfiguration über Umgebungsvariablen]

» CRV_MAX_FIELDS
• Standardwert: 50
• Beschreibung: Maximale Anzahl an Feldern pro Anfrage
• Schützt vor übermäßig großen Payloads

» CRV_MAX_ARRAYS
• Standardwert: 5
• Beschreibung: Maximale Anzahl an Arrays pro Anfrage
• Begrenzt die Komplexität der Datenstruktur

» CRV_MAX_DEPTH
• Standardwert: 2
• Beschreibung: Maximale Verschachtelungstiefe von Objekten
• Verhindert zu tiefe Objekthierarchien

» CRV_MAX_STRING_LEN
• Standardwert: 128
• Beschreibung: Maximale Länge für normale String-Felder
• Gilt für alle Felder außer Nachrichten-Felder

» CRV_MAX_MESSAGE_LEN
• Standardwert: 1024
• Beschreibung: Maximale Länge für Nachrichten-Felder
• Gilt für Felder, deren Name "message" enthält

» CRV_MAX_ARRAY_LEN
• Standardwert: 20
• Beschreibung: Maximale Anzahl an Elementen pro Array
• Begrenzt die Größe einzelner Arrays

[Validierungsregeln]

»» 1. Feldanzahl

Die Gesamtzahl aller Felder in der Anfrage darf CRV_MAX_FIELDS nicht überschreiten. Dies schützt vor übermäßig großen Payloads und verhindert Performance-Probleme.

»» 2. String-Länge

String-Felder werden unterschiedlich behandelt:

• Normale String-Felder sind auf CRV_MAX_STRING_LEN Zeichen begrenzt
• Felder, deren Name "message" enthält, dürfen bis zu CRV_MAX_MESSAGE_LEN Zeichen haben
• Die Erkennung von Nachrichten-Feldern erfolgt automatisch über den Feldnamen

»» 3. Arrays

Arrays unterliegen mehreren Beschränkungen:

• Maximal CRV_MAX_ARRAYS Arrays pro Anfrage erlaubt
• Jedes Array darf maximal CRV_MAX_ARRAY_LEN Elemente enthalten
• Nur String-Elemente in Arrays sind zulässig

»» 4. Verschachtelungstiefe

Objekte dürfen maximal CRV_MAX_DEPTH Ebenen tief verschachtelt sein. Dies verhindert zu komplexe Datenstrukturen und schützt vor Stack-Overflow-Problemen.

01.B.B.C CDA > POST /cda/contactRequest > REDIS

```
{: {
"id": "uuid-v4-string",       // Generierte UUID
"type": "sync",               // oder "communication", je nach Systemstandard
"timestamp": 1759960322,      // Aktueller Unix Timestamp
"payload": {
"dto": "contact_request",
"action": "create",
"payload": {
// Hier wird der komplette Inhalt des Request Body 1:1 durchgereicht
"addressDetails": {
"firstName": "Max",
"lastName": "Mustermann",
"email": "max@example.com"
},
"productInquiry": {
"productNumber": "XYZ-123",
"message": "Ich habe eine Frage."
},
"selectedAccessories": [
"extra_battery",
"protective_case"
]
}
}
} :}
```