CMDI Maker – Entwicklerhandbuch

# Paradigmen

Der CMDI Maker soll für den Benutzer so einfach wie nur irgendwie möglich zu bedienen sein. Er soll ihm dabei sinnvolle und nützliche Funktionen anbieten, die ihn schneller ans Ziel führen: Fertige Metadaten.

Das Benutzerinterface ist immer freundlich und einladend zu gestalten. Das Design ist flach, auf ablenkende 3D-Elemente werden verzichtet. Dafür werden kräftige Farben eingesetzt, um einschlägiger Software im Bereich der Korpus-Linguistik eine Alternative zu bieten.

# Formulare

Die Grunddatenstruktur, die beim CMDI Maker zum Einsatz kommt, sind Formulare (Forms). Forms beschreiben im JSON-Format, wie ein Formular für den Benutzer aufgebaut ist. Aus den Formular-Eingaben, die der Nutzer tätigt, ergibt sich auch die Speicherstruktur des Programms. Alle Daten werden gemäß der Struktur des Formulars gespeichert.

Beispiel:

title: "Contact",

name: "contact",

type: "column",

fields: [

{

heading: "Name",

name: "name",

type: "text"

},

{

heading: "Address",

name: "address",

type: "text"

},

{

heading: "Email",

name: "email",

type: "text"

}

Bei dieser Formularstruktur werden die Daten die der CMDI Maker speichert, entsprechend so formatiert sein:

"contact":   
{  
 "name": "Sebastian Zimmer",

"address: "Universitätsstr. 22",  
"email": "me@example.com"

}

Die APP bietet 4 Funktionen an, die für Formulare genutzt werden können:

1. **make (parent, field, element\_id\_prefix, element\_class\_prefix, session\_object)**

Diese Funktion erstellt ein Formular im DOM, unter dem Node „parent“. Die IDs aller Elemente im DOM bekommen den Präfix „element\_id\_prefix“, die Klassen den Präfix „element\_class\_prefix“. Optional kann mit „session\_object“ ein Datenobjekt übergeben werden, dessen Daten im Formular erscheinen sollen.

1. **fill (field, element\_id\_prefix, data\_object)**

Diese Funktion füllt ein bestehendes Formular, welches nach dem Template „field“ erstellt worden ist, mit Daten eines Datenobjekts „data\_object“. Damit die Funktion weiß, wo sich das Formular im DOM befindet, benötigt sie die Angabe des ID-Präfixes der DOM-Elemente, „element\_id\_prefix“.

1. **createEmptyObjectFromTemplate (field)**

Diese Funktion erstellt ein Datenobjekt nach Vorlage eines Form-Templates „field“ und gibt dieses zurück.

1. **fillObjectWithFormData (object, element\_id\_prefix, form\_element)**

Diese Funktion überprüft die Eingaben, die in einem Formular nach dem Template „form\_element“ mit dem ID-Präfix „element\_id\_prefix“ und füllt ein Objekt „object“ mit diesen Daten.

# Environments

Environments sind Arbeitsumgebungen, die sich dem Benutzer als Profiles darstellen, die er in den Einstellungen auswählen kann. Jedes Environment verfügt über einen eigenen Workflow mit eigenen Modulen, welche beim Laden des Environments in der Titelleiste dargestellt werden:



Jedes der Workflow-Module kann über eigene Funktionen verfügen, die in der Funktionsleiste angezeigt werden:



Jedes Modul verfügt über ein Identity-Objekt, welches von der App verwendet wird, um bspw. Den Workflow aufzubauen.

Bsp.:

my.identity = {

id: "actor",

title: "Actors",

icon: "user.png"

};

Das Identity-Objekt muss eine eindeutige ID, einen Titel und ein Icon haben.

Ein Environment kann weiterhin Module besitzen, welche nicht zum Workflow dazugehören.

Jedes Environment kann außerdem Settings exportieren, die in der View „Einstellungen“ bei geladenem Environment zu sehen sind. Die Settings sind im Hauptmodul des Environments definiert.

## Laden eines Environments

Der Ablauf beim Laden eines Environments wurde bewusst so festgelegt:

1. Die Settings des Environments werden initialisiert (zu finden im Hauptmodul des Environments mit dem Key „settings“)
2. Beim Erstellen des Workflows werden pro Modul zuerst seine Funktionen geladen, die das Modul im Key „functions“ definiert.
3. Dann wird das Modul initialisiert, d.h. falls es eine Funktion unter dem Key „init“ aufweist, wird diese ausgeführt. Dabei wird der Parameter „view“ übergeben, welcher das DOM-Element für die Ansicht („View“) des Moduls referenziert. Das Modul kann bei Bedarf mit seiner Init-Methode darauf zugreifen und Veränderungen an der View durchführen.

Da zuerst die Funktionen für ein Modul erstellt werden, kann sich das Modul bei der Call von module-init(view) darauf verlassen, dass diese zur Verfügung stehen und bei Bedarf nochmal angepasst werden.

# Save and Recall

Damit die Eingaben, die der Nutzer im CMDI Maker tätigt, nicht verloren gehen, nachdem das Browser-Fenster geschlossen wird, können diese in einer Browser-Datenbank gespeichert werden. Bei einem Speichervorgang speichert sowohl die App ihren aktuellen State (dazu gehören z. B. die zurzeit aktive View, das aktuell geladene Environment oder Einstellungen wie das Intervall für die automatische Speicherung) als auch das aktive Environment. Dazu fragt die App bei jedem Speichervorgang den State des Profils und all seiner Workflow-Module ab.

Um diese Funktion der Speicherung von Daten zu nutzen, benötigt jedes Workflow-Modul zwei Methoden:

1. **my.getSaveData()**

Diese Funktion wird von der App bei einem Speichervorgang aufgerufen und sollte den aktuellen State des Moduls zurückgeben.

1. **my.recall(data)**

Diese Funktion wird von der App bei einem Ladevorgang aufgerufen und als Parameter werden die Daten übergeben, die von diesem Modul gespeichert wurden. Das Modul ist dann dafür verantwortlich, seinen State auf der Grundlage dieser Daten wiederherzustellen.