

Automatische Generierung von Benutzeroberflächen

Bachelor-Thesis (P6)

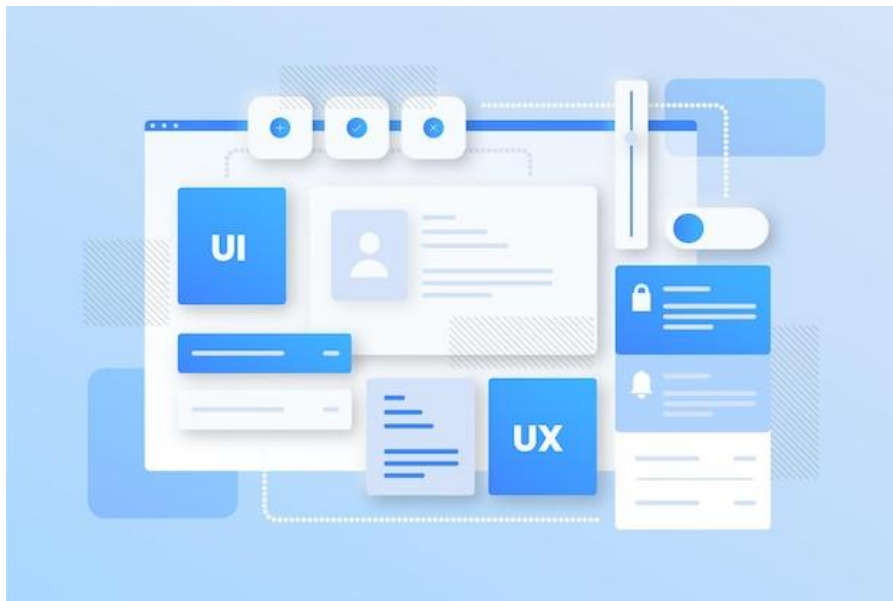


Abbildung 1: Visualisierung Benutzeroberflächengenerierung [1]

Stude

Fachbetreuer

*Prof. Dr. Norbert Seyff,
Dozent für Requirements Engineering
Institut für Interaktive Technologien FHNW*

*Dr. Nitish Patkar
Wissenschaftlicher Mitarbeiter
Institut für Interaktive Technologien FHNW*

Projektnummer

23FS_IIT03

Vereinbarung

vertraulich

Fachhochschule Nordwestschweiz, Hochschule für Technik

Windisch, August 2023

Abstract

Die vorliegende Arbeit behandelt die automatische Generierung von Benutzeroberflächen aus Beschreibungen im JSON-Format basierend auf dem integrativen Business System enablerr. Das System nutzt Plugins, um Arbeitsschritte zu automatisieren und Aufgaben, sogenannte Ereignisse, zu verarbeiten. Dieser Bericht stellt eine Lösung zur automatischen Generierung von Benutzeroberflächen auf Grundlage von den in enablerr definierten Ereignissen vor.

Die Projektvision strebt an, das bestehende enablerr-Projekt durch die ereignisbasierte Generierung von Benutzeroberflächen zu erweitern. Die Fragestellungen umfassen die Analyse des Forschungsstandes im Gebiet automatische Benutzeroberflächengenerierung, die eventuell nötigen Anpassungen der Ereignisstruktur und die Evaluierung der Lösungseffektivität.

Die Methodik beinhaltet eine ausführliche Recherche zum Thema automatische Benutzeroberflächengenerierung, die Konzepterarbeitung und Lösungsimplementierung. Das Projektmanagement folgt dem Water-Scrum-Fall-Modell, um die Konzepterarbeitung, Lösungsimplementierung und -evaluierung dynamischer zu gestalten.

Die Ergebnisse zeigen, dass die umgesetzte Lösung effizient ist. Die Lösung bietet eine solide Basis-Benutzeroberfläche und ist im Nachhinein noch erweiterbar. Limitationen umfassen fehlende Datenbeziehungen und einige Anpassungsmöglichkeiten. Nichtsdestotrotz hat die Lösung diverse Weiterentwicklungsmöglichkeiten.

Keywords:

Automatische Generierung, JSON-Verarbeitung, Benutzeroberflächengenerierung

Inhaltsverzeichnis

Abbildungsverzeichnis	v
1 Einleitung	1
1.1 Ausgangslage.....	1
1.2 Projektvision.....	1
1.3 Fragestellungen.....	1
1.4 Methodik.....	2
1.5 Aufbau des Projektberichts.....	2
2 Hintergrund	3
2.1 Business System enabler.....	3
2.1.1 Ereignisbasierte Systemarchitektur.....	3
2.1.2 Generierung der Benutzeroberfläche von Aufgaben.....	7
2.2 Adaptive Grid.....	10
3 Automatisierte Datenvisualisierung: State-of-the-art in der Transformationstechnologie	12
3.1 Wissenschaftliche Arbeiten zu «automatische Generierung von Benutzeroberflächen».....	12
3.2 Tools für die automatische Generierung von Benutzeroberflächen.....	17
3.3 Erkenntnisse aus der Recherche.....	17
4 Entwurf eines JSON-zu-HTML-Transformators	19
4.1 Prinzip.....	19
4.2 Anpassbarkeit und Erweiterbarkeit.....	22
4.3 Anforderungen an die Lösung.....	22
4.4 Integration des Adaptive Grids.....	22
5 Realisierung des JUIBuilder's	23
5.1 Programmiersprachen und Technologien.....	23
5.2 Architektur und Design.....	23
5.3 Datenstrukturen und Algorithmen.....	24
5.4 Funktionen und Module.....	25
5.5 Fehlerbehandlung.....	27
6 Evaluierung der Lösung «JUIBuilder»	27
6.1 Ablauf der Benutzertests.....	27
6.2 Verbesserungsvorschläge und Umsetzung des Feedbacks.....	28
6.2.1 Unklare Fehlermeldungen.....	28
6.2.2 undefinierte Elemente im generierten Code.....	28
6.2.3 Nachträgliche Anpassungen.....	28
6.2.4 Bessere Benutzerfreundlichkeit für Anfänger.....	28
6.2.5 Styling-Auslagerung.....	28
6.3 Finale Evaluierung der Lösung.....	29
6.4 Vergleich der generierten und der programmierten Benutzeroberfläche.....	29
7 Ergebnisse und Diskussion	31
7.1 Ergebnisse und Vergleich zum Forschungsstand.....	31
7.2 Limitationen und kritische Bewertung der Lösung.....	31

Inhaltsverzeichnis

7.3 Ausblick auf zukünftige Entwicklungsmöglichkeiten.....	31
Quellenverzeichnis.....	32
Ehrlichkeitserklärung.....	35
Anhang.....	36
A Aufgabenstellung im Originalwortlaut.....	36
B JSON-Schema Ereignis «Rechnung stellen».....	37
C Evaluierungsdokumentation für die Benutzertests.....	43

Abbildungsverzeichnis

Abbildung 1: Visualisierung Benutzeroberflächengenerierung [1]	1
Abbildung 2: Visualisierung eines Ereignisraums [2, S. 5]	3
Abbildung 3 Visualisierung Ereigniskreise [2, S. 6]	4
Abbildung 4 Visualisierung Ereignismodifikatoren [2, S. 9]	4
Abbildung 5 Visualisierung Ereignis [2, S. 12]	5
Abbildung 6 Visualisierung Ereignisfluss [2, S. 15]	6
Abbildung 7 Pluginschema in enablerr [5]	6
Abbildung 8 Liste des Aufgabentyps "Rechnung stellen" [5]	7
Abbildung 9 enablerr Startseite der Demo-Version [5]	7
Abbildung 10 Benutzeroberfläche des Aufgabentyps "Rechnung stellen" [5]	7
Abbildung 11 Berechnungsschema des Adaptive Grid [6, S. 21]	10
Abbildung 12 Darstellung des Empty Space im Adaptive Grid [6, S. 20]	11
Abbildung 13 HTML-Generierungsprozess	19
Abbildung 14 Beispiel eines Datenschemas im JSON-Format [29]	19
Abbildung 15 Beispiel eines Elementschemas	20
Abbildung 16 Beispiel von Daten definiert im JSON-Format [29]	20
Abbildung 17 Im Browser geöffneter generierter HTML-Code	21
Abbildung 18 Vom JUIBuilder generierter HTML-Code	21
Abbildung 19 Klassen des JUIBuilder's	24
Abbildung 20 Gegenüberstellung manuell programmierte und generierte Benutzeroberfläche	29

**Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here. Error!
Use the Home tab to apply Überschrift 1 to the text that you want to appear here.**

1 Einleitung

1.1 Ausgangslage

enablerr ist ein integratives Business System, welches die Vernetzung von Arbeitsschritten automatisiert. Das System basiert auf Plugins, welche Arbeitsschritte zusammenfassen und eine Wiederverwendung in anderen Kontexten ermöglichen. Die Plugins beschreiben Aufgaben, in enablerr Ereignisse genannt, welche in enablerr verarbeitet werden. Benutzer von enablerr legen dazu fest, was für Aufgaben ihr enablerr-System verarbeiten können soll. Die Arten an Aufgaben und Informationen, welche zur Erledigung der Aufgabe benötigt werden, und auch die Unternehmen sind vielfältig. Um all diese Aufgaben bearbeiten zu können, muss sich die Benutzeroberfläche an den Aufgabenkontext anpassen. Das heisst enablerr benötigt eine Vielzahl von unterschiedlichen Elementen, um die unterschiedlichen Aufgaben abzudecken.

1.2 Projektvision

Das bestehende Projekt soll erweitert werden um die automatische Generierung von Benutzeroberflächen basierend auf den in enablerr definierten Ereignissen. Zum Ende dieses Projekts soll eine Lösung ausgeliefert werden, welches in das System von enablerr eingebunden werden kann. Dabei ist das Ziel, dass sowohl das "Adaptive Grid" wie auch die Erweiterung der ereignisbasierten automatischen Generierung lauffähig sind.

1.3 Fragestellungen

- A. Analyse: Welche bestehenden Konzepte in wissenschaftlichen Veröffentlichungen, die sich auf die automatische Generierung von Benutzeroberflächen beziehen, könnten relevant sein für dieses Projekt?

Um diese Frage zu beantworten, werden aktuelle wissenschaftliche Arbeiten auf dem Themengebiet recherchiert und auf ihre Brauchbarkeit im Rahmen dieses Projekts analysiert.

- B. Wie kann die Benutzeroberfläche von enablerr automatisch generiert werden?
- Sind Anpassungen oder Erweiterungen an der Struktur von Ereignissen nötig?
 - Müssen noch andere Informationen aus dem System einbezogen werden?

Diese Frage wird während der Konzepterarbeitung geprüft und durch die Lösungsimplementierung beantwortet.

- C. Ist die implementierte Lösung eine effektive Methode zur Benutzeroberflächengenerierung?
- Inwiefern erfüllt die implementierte Methode die Anforderungen an Benutzeroberflächengenerierung?
 - Wie unterscheidet sich die implementierte Methode zur Benutzeroberflächengenerierung von traditionellen Ansätzen oder gängigen Praktiken?

Auf diese Fragen wird im Rahmen der Evaluierung und Diskussion eingegangen.

**Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here. Error!
Use the Home tab to apply Überschrift 1 to the text that you want to appear here.**

1.4 Methodik

Das Projekt beginnt mit einer ausführlichen Recherche zum momentanen Entwicklungsstand bei der automatischen Generierung von Benutzeroberflächen. Danach wird daraus ein Konzept für die ereignisbasierte automatische Generierung von Benutzeroberflächen erarbeitet. Sobald das Konzept steht, wird die Lösung entwickelt. Die Lösung wird fortlaufend getestet. Die Lösung wird daraufhin in einer Demo-Version von enablerr eingebunden und durch Benutzer evaluiert. Jeder Schritt wird entsprechend im Projektbericht dokumentiert. Das Projektmanagement erfolgt nach dem Water-Scrum-Fall-Modell. Dabei werden Recherche und Konzepterarbeitung, sowie Evaluierung nach Waterfall-Modell und die Entwicklung und die Validierung nach Scrum-Modell erledigt. Die Zielerreichung wird anhand der in der Konzept-Phase definierten Anforderungen an die Lösung gemessen.

1.5 Aufbau des Projektberichts

In dieser Arbeit wird ein Lösungskonzept für die automatisierte Generierung von Benutzeroberflächen ausgerichtet auf die ereignisbasierte Architektur vom Business system enablerr beschrieben. Kapitel 2 befasst sich mit dem Business System enablerr und dem Projekt Adaptive Grid als Hintergründe für dieses Projekt. Kapitel 3 geht auf die relevanten Grundlagen der automatisierten Generierung von Benutzeroberflächen ein. Kapitel 4 beschreibt das Lösungskonzept für das zu lösende Problem, indem es auf das Prinzip und die Anforderungen an die Lösung eingeht. In Kapitel 5 wird die Implementierung der Lösung anhand der Architektur und des Codes erläutert. Kapitel 6 behandelt die Evaluierung der Lösung mittels Vergleich der generierten Benutzeroberfläche mit der manuell erstellten, sowie der Benutzerauswertung. In Kapitel 7 werden die Ergebnisse und Diskussion präsentiert., indem die Ergebnisse dem Forschungsstand gegenübergestellt und kritisch bewertet werden und ein Ausblick auf zukünftige Entwicklungsmöglichkeiten vorgestellt wird.

Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here. Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here.

2 Hintergrund

Im vorliegenden Kapitel werden die Kernaspekte des Business Systems enablerr, dessen ereignisbasierte Architektur, sowie die Generierung der Benutzeroberfläche von enablerr beschrieben. Danach wird die Funktionsweise des Vorgängerprojekts Adaptive Grid aufgezeigt. Das Kapitel Hintergrund dient zur Vertiefung des Kontextes und der Motivation hinter den weiteren Untersuchungen zu dieser Arbeit.

2.1 Business System enabler

enablerr ist ein plugin-basiertes Business System, dessen Architektur eine flexible Anpassung des Systems zulässt. Es zeichnet sich vor allem durch vier Kernaspekte aus:

- Prozesse lassen sich in Arbeitsschritte und Aufgaben aufteilen, welche sich in anderen Kontexten wiederverwenden lassen.
- Arbeitsschritte verknüpfen sich selbstständig zu einem Workflow.
- Die Abbildung komplexer Geschäftstätigkeiten basiert auf operativen Arbeitsschritten.
- Offene Schnittstellen lassen eine Integration bestehender Infrastrukturen zu [1].

2.1.1 Ereignisbasierte Systemarchitektur

Die Systemarchitektur von enablerr basiert auf der Verarbeitung von sogenannten Ereignissen.

Diese Architektur besteht aus einem Ereignisraum, welcher zwei Ereigniskreise, zwei Ereignismodifikatoren und einen Ereignisfluss enthält. Diese haben unterschiedliche Funktionen:

Ereigniskreise generieren Ereignisse.

Ereignismodifikatoren verändern Ereignisse.

Der Ereignisfluss verbindet die Ereignisse dynamisch [3].

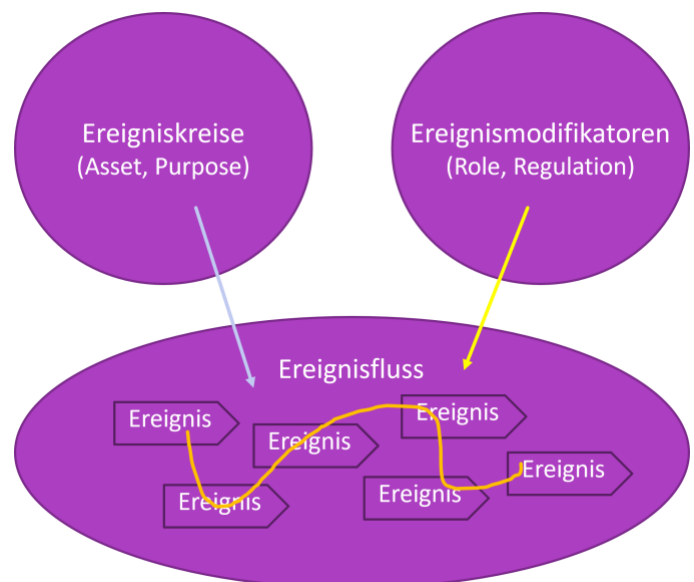


Abbildung 2: Visualisierung eines Ereignisraums [2, S. 5]

Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here. Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here.

Ereigniskreis

Ereigniskreise bestehen aus Domains, welche Geschäftsvermögen (Assets) und Geschäftstätigkeiten (Purposes) repräsentieren.

Assets umfassen das Geschäftsvermögen eines Unternehmens, daher dass, was ein Unternehmen für seine Geschäftstätigkeit benötigt.

Purposes umfassen Tätigkeiten, welche dem Unternehmen direkt oder indirekt Umsatz einbringen.

Ereignisse werden daher durch den Lebenszyklus von Objekten gesteuert und halten Werte wie Kosten und Umsätze fest [3].

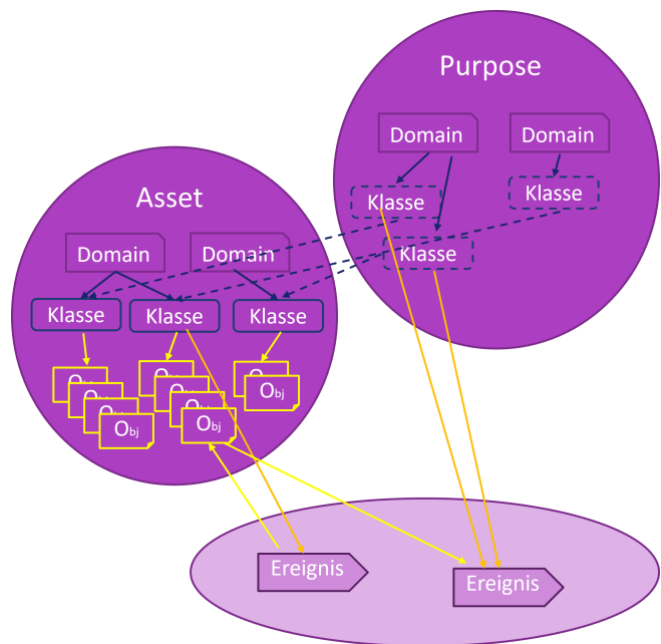


Abbildung 3 Visualisierung Ereigniskreise [2, S. 6]

Ereignismodifikator

Mit Ereignismodifikatoren kann man ausserhalb der Ereigniskreise auf Ereignisse zugreifen. Die Ereignismodifikatoren umfassen Roles und Regulations.

Roles definieren die Zugriffsrechte auf Ereignisse basierend auf den Ereignisdefinitionen. Einem Benutzer können mehrere Roles zugeordnet werden.

Regulations sind vom Unternehmen nicht steuerbare Flüsse von Assets und Purposes, wie z.B. staatliche Vorgaben wie Steuern und Versicherungen. Dabei generieren sie entweder neue Ereignisse oder sie verändern Werte eines Ereignisses [3].

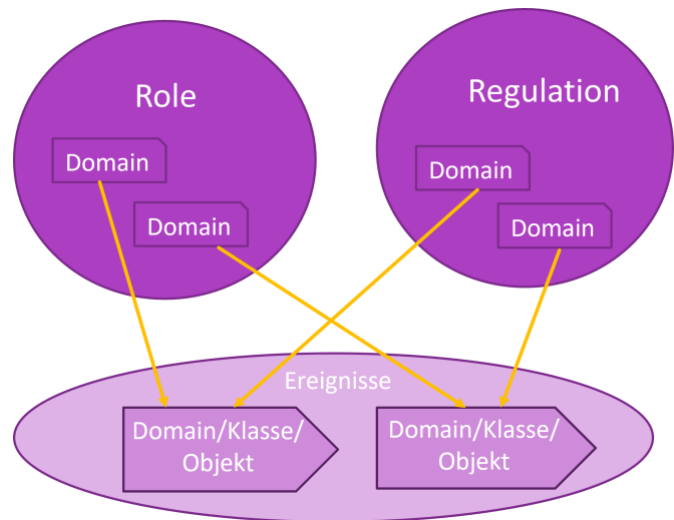


Abbildung 4 Visualisierung Ereignismodifikatoren [2, S. 9]

Ereignis

Ein Ereignis repräsentiert einen atomischen Teil eines Workflows und trägt zum Ziel eine Information, ein Produkt bzw. eine Dienstleistung oder eine Kompensation zwischen dem Unternehmen und einer juristischen Person zu transferieren [3].

Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here. Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here.

Ein Ereignis basiert auf einem Subjekt, also einer Person, sowie mindestens einem Objekt aus einem Ereigniskreis. Zu den Objekten kann es beliebig viele zusätzliche Informationen definieren.

Das Ereignis wird durch einen **Trigger** ausgelöst. Dieser kann manuell, in einer bestimmten Periodizität oder durch ein anderes Ereignis ausgelöst erfolgen.

Die Ausführung eines Ereignisses erzeugt Informationen als Datenklasse, welche in der Datenbank gespeichert werden. Eine Datenklasse kann durch ein oder mehrere Ereignisse aktualisiert werden. Eine Aktualisierung produziert eine neue nicht-modifizierbare Version der Datenklasse.

Das Ereignis wird im JSON-Format definiert. Ereignisdefinitionen sind jederzeit anpassbar. Ein angepasstes Ereignis ist eine neue Version des vorhergehenden Ereignisses, welcher Eigenschaften hinzugefügt oder entnommen wurden [3].

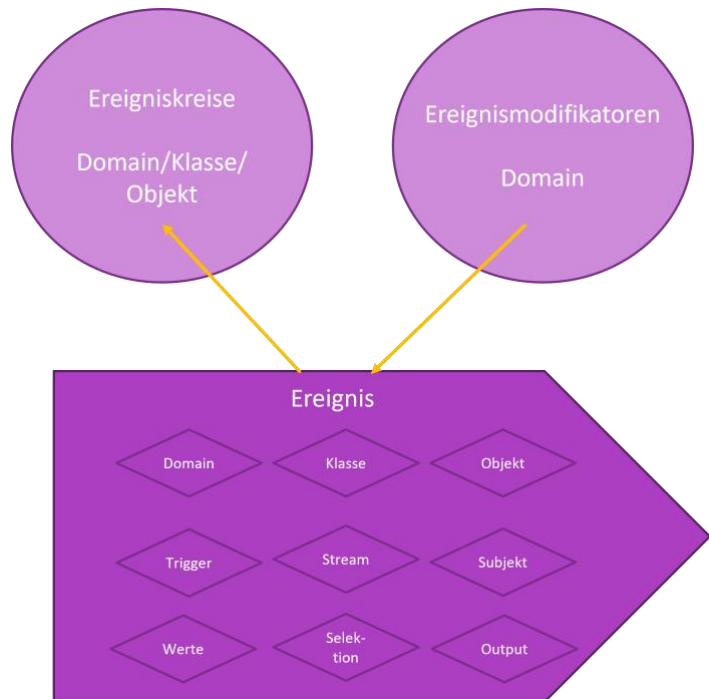


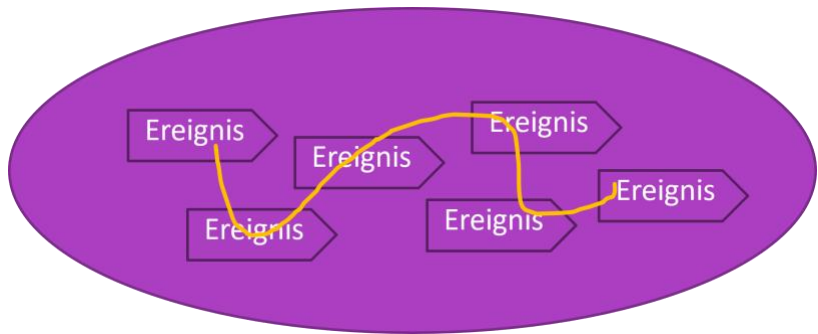
Abbildung 5 Visualisierung Ereignis [2, S. 12]

**Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here. Error!
Use the Home tab to apply Überschrift 1 to the text that you want to appear here.**

Ereignisfluss

Ein Ereignis kann mit Informationen aus anderen Ereignissen verknüpft werden.

Aufgaben haben keinen vorgegebenen Ablauf, sondern werden durch Ereignisse mittels bestimmter Werte innerhalb der Ereignisdefinitionen beeinflusst [3].



In enablerr gilt:

Abbildung 6 Visualisierung Ereignisfluss [2, S. 15]

Auf jedes Ereignis der Flow-Art Asset muss ein Ereignis der Flow-Art Kompensation folgen. Auf jedes Ereignis der Typ-Art Vorbereitung folgt ein Ereignis der Typ-Art Durchführung.

Der Ereignisfluss generiert nach Ausführung des Ereignisses Aufgaben des entsprechenden darauf folgenden Ereignisses [4].

Plugin

Ein Plugin ist eine Sammlung von Ereignissen im JSON-Format. Der einzige Weg, Ereignisse in das enablerr-System zu bringen, ist, Plugins über die enablerr-API heraufzuladen. Wird das Plugin durch eine bevollmächtigte Person validiert, werden die Ereignisse zum Aufgaben-Generator hinzugefügt [4].

```
{
  $schema : https://enablerr.dev/schema/enablerr_schema_event_V1.30.json
  event : [ 2 items ]
  0 : {
    identity : {
      name : openCustomer
      designation : Kunde eröffnen
      description : Erfassung und Pflege des Kundenstammes und Verknüpfung mit einer Firma oder Person.
      circle : Asset
      domain : Auftragsabwicklung
    }
    validity : { 0 props }
    trigger : { 0 props }
    input : { 0 props }
    output : {
      class : [ 1 item ]
      0 : {
        name : customer
        designation : Kunde
        display : [ 1 item ]
        item : [ 2 items ]
        list : [ 1 item ]
      }
    }
    state : {
      direction : Asset
      task : Information
      step : Execution
    }
  }
  1 : { 6 props }
}
```

Abbildung 7 Pluginschema in enablerr [5]

Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here. Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here.

2.1.2 Generierung der Benutzeroberfläche von Aufgaben

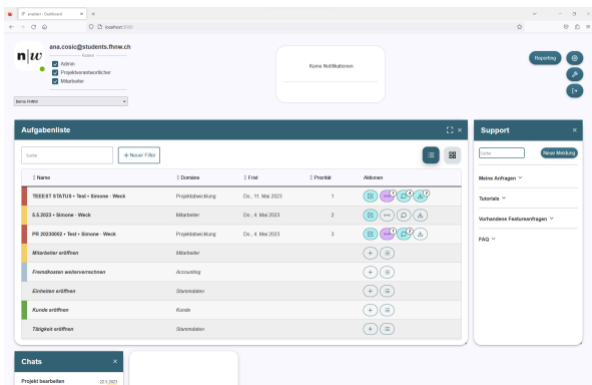


Abbildung 9 enablerr Startseite der Demo-Version [5]

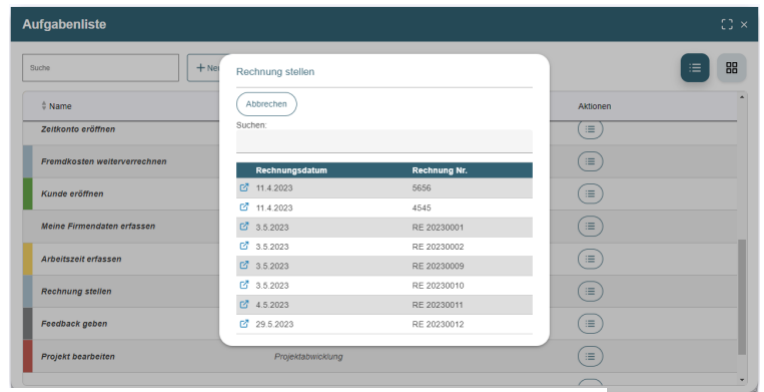


Abbildung 8 Liste des Aufgabentyps "Rechnung stellen" [5]

Die Benutzeroberfläche von enablerr besteht aus Modulen, welche in einem Svelte Grid platziert werden. Es gibt momentan vier Module: Aufgabenliste, Support, Chats und Neues Modul.

Die Aufgaben innerhalb des Moduls Aufgabenliste werden als Liste oder als Grid angezeigt. Die Benutzeroberfläche einzelner Module wird anhand der Informationen aus einem Ereignis zusammengestellt.

Die JSON-Beschreibung eines Ereignisses, beispielsweise um eine Rechnung zu erstellen, wird aufgerufen und daraus ein Schema-Objekt erstellt. Daraus wird die Benutzeroberfläche der Aufgabe zusammengestellt, wobei die Informationen aus dem Schema-Objekt als Inhalt von HTML-Elementen eingelesen werden.

Wird eine Aufgabe ausgewählt, erscheint zuerst eine Übersicht bereits erstellter Aufgaben dieses Aufgabentyps.

Aus der Liste kann dann eine Aufgabe des Typs «Rechnung stellen» ausgewählt werden, worauf sich die Aufgaben-Benutzeroberfläche öffnet.

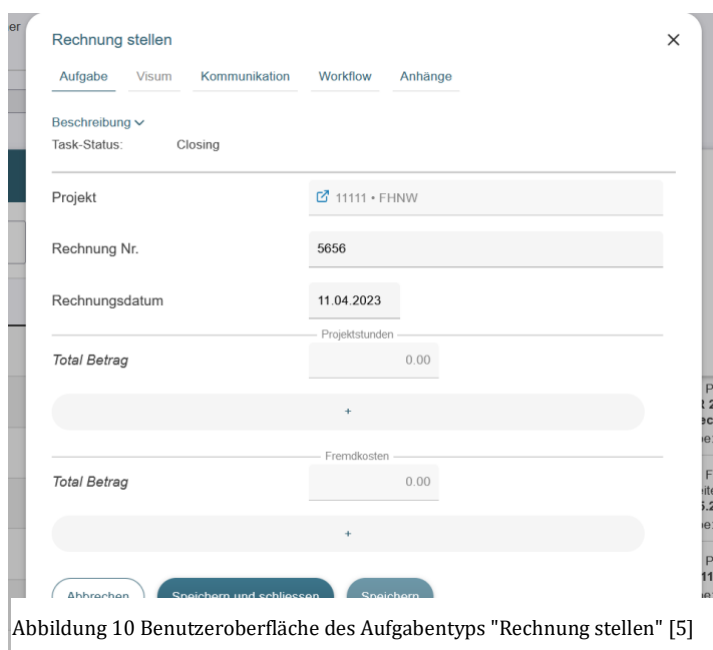


Abbildung 10 Benutzeroberfläche des Aufgabentyps "Rechnung stellen" [5]

**Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here. Error!
Use the Home tab to apply Überschrift 1 to the text that you want to appear here.**

**Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here. Error!
Use the Home tab to apply Überschrift 1 to the text that you want to appear here.**

Datenschema eines Ereignisses in enablerr

Eine Aufgabe in enablerr wird im JSON-Format beschrieben, welches nach einem vorgegebenen Muster aufgebaut ist. Die Aufgabenbeschreibung könnte dabei auf unterschiedliche Art erfolgen. Eine Anwendung benötigt spezifische, im Vorherein definierte Informationen, damit es eine Aufgabe verarbeiten kann. Dieses Muster wird durch das JSON-Schema vorgegeben. Ein JSON-Schema beschreibt die Struktur von Daten und dient wiederum als Grundlage, um Daten zu validieren.

Ein Ereignis im JSON-Format besitzt folgende Informationen verpackt als Objekte:

- a. «authorization» beschreibt, welche Benutzerrollen in enablerr welche Rechte am Ereignis haben.
- b. «identity» umfasst den Titel und die Beschreibung der Aufgabe, sowie welchem Ereigniskreis und welcher Domäne darin das Ereignis angehört.
- c. «input» beschreibt Daten aus Vorgänger-Ereignissen, welche in dieses Ereignis übernommen werden sollen.
- d. «output» legt fest, welche Daten wie auf der Benutzeroberfläche angezeigt oder erfasst werden sollen. Innerhalb von «output» wird noch weiter spezifiziert:
 - e. «display» beschreibt, welche Eigenschaften in der Übersicht der vorhandenen Aufgaben eines Aufgabentyps wie in Abbildung 9 als Spaltentitel dienen.
 - f. «item» definiert die Daten, welche einmalig für eine Aufgabe erfasst werden sollen, wie beispielsweise ein Rechnungsdatum. «item»-Eigenschaften können vom Typ Nummer, Zeichenkette, Wahrheitswert, Objekt, Datum oder Datum und Uhrzeit sein. Jede Eigenschaft kann weiter eingeschränkt und spezifiziert werden mithilfe von «properties»:
 - «required» definiert, ob das Feld ausgefüllt werden muss
 - «unique» definiert, ob der Wert einzigartig sein muss
 - «currency» bezeichnet einen Währungswert
 - «unit» bezeichnet eine Einheit
 - «decimals» bezeichnet wieviel Stellen nach dem Komma eines Dezimalwerts angezeigt werden sollen
 - «hidden» definiert, ob das Feld angezeigt werden soll
 - «readonly» definiert, ob der Wert bearbeitbar ist
 - «enum» definiert eine Auswahlliste an Werten, welche das Feld annehmen kann
 - «constant» bezeichnet einen immer gleich bleibenden Wert
 - «time» bezeichnet einen Zeitwert
 - «textarea» bezeichnet ein grösseres Texteingabefeld
 - «email» bezeichnet eine Zeichenfolge im E-Mail-Format
 - «tel» bezeichnet eine Zeichenfolge im Telefonnummer-Format
 - «url» bezeichnet eine Zeichenfolge im URL-Format
 - «value» definiert den Wert, welcher gesetzt werden soll, falls keine Benutzereingabe erfolgt. «refItemNext» nimmt den Wert des referenzierten Feldes, versetzt diesen um den angegebenen Wert, extrahiert den grössten Wert des Dokuments und zählt einen Schritt ab der Basisnummer herauf.
 - «refItem» bezeichnet den Wert eines referenzierten «item»-Objekts.

**Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here. Error!
Use the Home tab to apply Überschrift 1 to the text that you want to appear here.**

- «formula» bezeichnet Addition, Subtraktion, Multiplikation oder Division auf einem Wert.
 - «lookup» ist eine Referenz auf einen nicht in Input definierten Datensatz.
 - «currency», «unit» und «value» können die Werte «constant», «refItem», «formula», «variable», «userInput», «refItemNext» und «lookup» annehmen.
- g. «list» definiert die Daten, welche wiederholt für eine Aufgabe erfasst werden sollen, wie beispielsweise Projektkosten, welche monatlich erfasst werden. «list» besteht wiederum aus Objekten des Typs «item».
- h. «result» beschreibt das Ausgabeformat des Resultats des Ereignisses.
- i. «state» schildert die Aufgabenart und den Erledigungsgrad der Aufgabe
- j. «trigger» definiert, wann eine Aufgabe erstellt wird. Dabei gibt es die zwei Einstellungen «Manual» und «Event». «Manual» bezeichnet, dass die Aufgabe manuell durch den Benutzer angelegt wird. «Event» bezeichnet, dass die Aufgabe erstellt werden soll.
- k. «validity» bezeichnet den Gültigkeitszeitraum eines Ereignisses.

Für die Erstellung der Benutzeroberfläche der Aufgabe sind vor allem «identity», «input» und «output» relevant.

2.2 Adaptive Grid

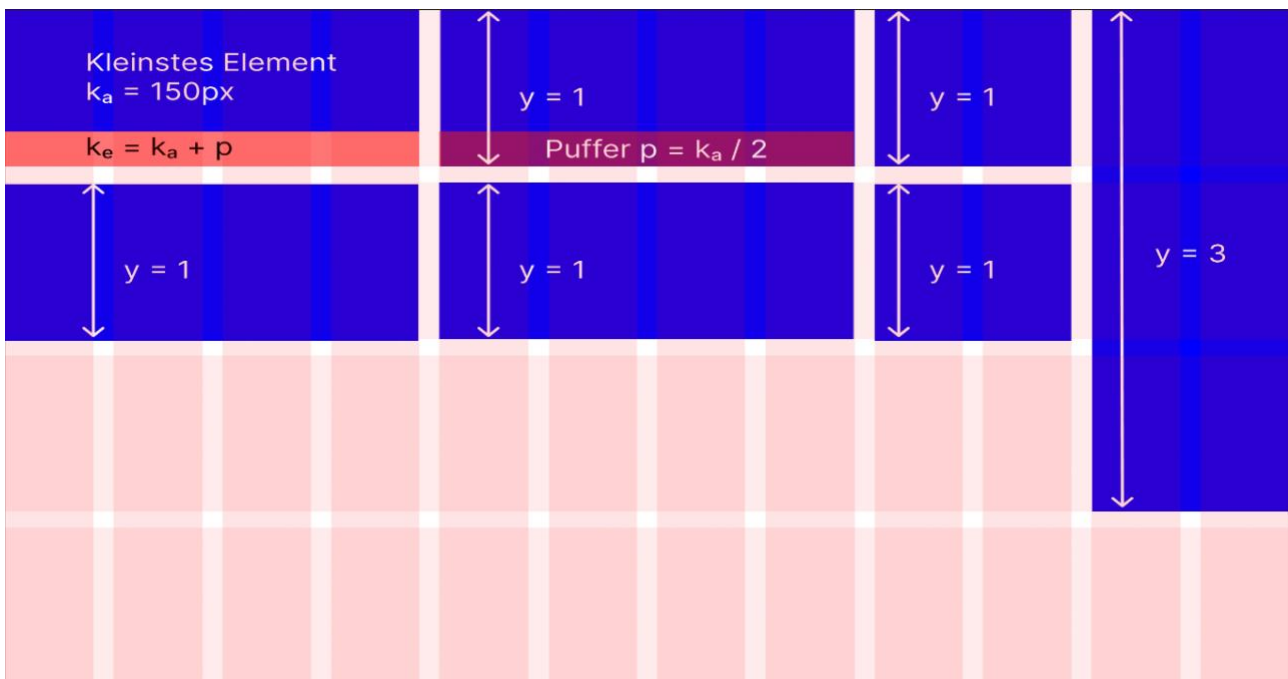


Abbildung 11 Berechnungsschema des Adaptive Grid [6, S. 21]

Das Adaptive Grid ist eine JavaScript-Bibliothek, welche dazu verwendet werden kann, Elemente auf einer Benutzeroberfläche beim Hinzufügen designtechnisch optimal zu platzieren. Die Bibliothek soll mit der in diesem Projekt entwickelten Lösung kompatibel sein.

**Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here. Error!
Use the Home tab to apply Überschrift 1 to the text that you want to appear here.**

Der Platzierungsprozess beginnt damit, dass die Grid-Grösse abhängig von der Fenstergrösse definiert wird. Das Grid enthält 12 Spalten. Daraufhin werden die direkten Kinderelemente ausgelesen und ihre Eigenschaften erfasst. Die Elementreihenfolge entspricht initial der DOM-Reihenfolge.

Für jedes direkte Kinderelement wird die Grösse definiert, um es korrekt im Grid platzieren zu können. Die verschiedenen Grössen werden über ihre Weite definiert. XS entspricht einer Spalte im Grid und XL entspricht 12 Spalten im Grid, wobei mehrere Werte dazwischen liegen.

Danach wird die Position des Elements im Grid mathematisch berechnet, wobei darauf geachtet wird, dass empty space vermieden wird.

Schlussendlich wird das Element mit der definierten Grösse an der definierten Position im Grid platziert [6, S. 18 - 19].

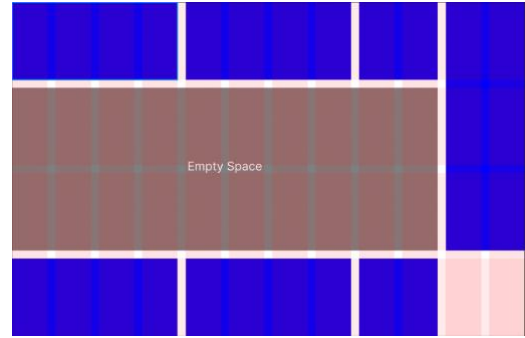


Abbildung 12 Darstellung des Empty Space im Adaptive Grid [6, S. 20]

**Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here. Error!
Use the Home tab to apply Überschrift 1 to the text that you want to appear here.**

3 Automatisierte Datenvisualisierung: State-of-the-art in der Transformationstechnologie

Es existieren bereits mehrere Ansätze für die automatisierte Generierung von Benutzeroberfläche, welche sich hauptsächlich in der Generierungsart unterscheiden. Dieser Abschnitt konzentriert sich auf bereits vorhandene wissenschaftliche Arbeiten und Technologien im Bereich automatische Generierung von Benutzeroberflächen und adaptive Benutzeroberflächen.

Da das Vorgängerprojekt bereits ausführlich auf das Thema «adaptive Benutzeroberflächen» eingegangen ist, wird das Thema in dieser Arbeit in Kombination mit der automatischen Generierung von Benutzeroberflächen behandelt.

3.1 Wissenschaftliche Arbeiten zu «automatische Generierung von Benutzeroberflächen»

Titel / Jahr	Zusammenfassung	Verwendete Technologien	Fazit
GUILGET: GUI Layout GEneration with Transformer / 2023 [7]	Die Arbeit befasst sich mit der automatischen Generierung von GUI-Layouts mit einem Transformer-Modell, welches Daten parallel statt sequentiell verarbeitet. Das Modell wird mit Hilfe eines Datensatzes an bestehenden Layouts trainiert.	Machine Learning	Konzentriert sich auf Machine Learning, was in dieser Arbeit nicht weiterverfolgt wird.
An Automatic GUI Generation Method Based on Generative Adversarial Network / 2022 [8]	Die Arbeit präsentiert die automatische Generierung von UI mittels eines GAN. Da GAN wird auf einem Datensatz von GUI Bildern trainiert. Der GAN-generator generiert Bilder, welche der GAN-discriminator mit den Bildern des Datensatzes vergleicht.	Deep Learning, Image Processing	Deep Learning wird in dieser Arbeit nicht verfolgt, die für das Training verwendeten Layouts sind zudem für enablerr uninteressant.
Inishell 2.0: semantically driven automatic GUI generation for scientific models / 2022 [9]	Die Arbeit umschreibt die Software Inishell, welches einen semantischen Ansatz zur automatischen Generierung von GUI verfolgt. Dabei spiegelt Inishell eine XML-Struktur wieder.	Modellbasierter Ansatz, XML, C++/Qt Software (Inishell)	XML hat ein ähnliches Abstraktionsniveau wie das in enablerr verwendete JSON-Format. Dieser Ansatz sollte näher angeschaut werden.

**Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here. Error!
Use the Home tab to apply Überschrift 1 to the text that you want to appear here.**

Automatic generation of user-sensitive application-sensitive self-adapted UI system for smartphone applications / 2022 [10]	Die Arbeit befasst sich mit adaptiven UI Systemen, welche Benutzer- und Applikations-Kontext in die Layout-Generierung miteinbeziehen. Das System sammelt Daten zu Benutzerverhalten, generiert Bewegungsregeln basierend auf den Daten und generiert dann das UI.		Konzentriert sich eher auf das Thema "Adaptivität".
UI Generation for Business Data-Based Apps from Task, Domain and User Models / 2021 [11]	Es wird ein modell-basierter Ansatz für die automatische Generierung von UI für Business datenbasierte Applikationen vorgestellt. Die hinzugezogenen task, domain und user Modelle beziehen sich auf einander und werden in dieser Arbeit miteinander kombiniert im Framework DB-Use.	Modellbasierter Ansatz, Template-basierte Codegenerierung	Die Art, wie und worauf basierend die verschiedenen Modelle verlinkt werden, könnte relevant sein für diese Arbeit und ist näher anzuschauen.
GUIGAN: Learning to Generate GUI Designs Using Generative Adversarial Networks / 2021 [12]	Das Deep Learning Modell GUIGAN generiert GUI Designs mit Hilfe von Generative Adversarial Networks basierend auf existierenden Komponenten aus mobilen Applikationen, welche neu zusammengesetzt werden.	Deep Learning, Image Processing	Deep Learning wird in dieser Arbeit nicht verfolgt.
Generation of Adaptive Mobile Applications Based On Design Patterns for User Interfaces / 2019 [13]	Die Arbeit untersucht den Einsatz von Design Patterns für die Generierung adaptiver Benutzeroberflächen. Dafür werden Benutzerprofile und Umgebungsbedingungen beigezogen.	Ontologiebasierter Ansatz	Die Arbeit beschäftigt sich mit der adaptiven Benutzeroberfläche.
Evaluating user interface generation approaches: model-based versus model-driven development / 2019 [14]	Der modell-basierte und der modell-getriebene Ansatz werden miteinander anhand festgelegter Kriterien auf Effizienz bei der automatischen Generierung von	Modellbasierter und modellgetriebener Ansatz	Die gesammelten Kriterien können relevant sein für die Evaluierung der Lösung.

**Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here. Error!
 Use the Home tab to apply Überschrift 1 to the text that you want to appear here.**

	Benutzeroberflächen verglichen.		
A Framework for Adaptive User Interface Generation based on User Behavioural Patterns / 2019 [15]	Es wird ein Ansatz zur Generierung von UI basierend auf Benutzerverhalten untersucht. Dabei wird festgehalten, welche UI-Komponenten besonders oft verwendet werden und das UI danach gerichtet adaptiert.	Machine Learning	Da enablerr noch nicht produktiv läuft, ist eine Auswertung des Verhaltens in einem genügend grossen Masse nicht möglich.
Json-GUI-A module for the dynamic generation of form-based web interfaces / 2019 [16]	JSON-GUI generiert auf Basis von JSON-Dateien dynamisch form-basierte Webinterfaces. Der Ansatz basiert auf der Definition von JSON-Schemas, welche die Struktur des Formulars definieren und generiert durch JavaScript Formularelemente und -aktionen.	JSON, Bootstrap, HTML, CSS, JavaScript, Angular	Die Umsetzung des UI mittels JavaScript und JSON ist interessant für diese Arbeit, da Ereignisse in enablerr ebenfalls mittels JSON definiert werden.
Automatic UI Generation for Aggregated Linked Data Applications by Using Sharable Application Ontologies / 2018 [17]	Die Arbeit beschäftigt sich mit der automatischen Generierung von Benutzeroberflächen für Anwendungen, welche auf aggregierten Linked Data basieren. Dies wird durch eine einzelne, deklarative, datenzentrierte Applikationsbeschreibung, welche strukturelle und verhaltensrelevante Informationen enthält, erledigt.	Ontologie-basierter, modellgetriebener Ansatz	Eventuell relevant, um sich ein Bild davon zu machen, welche strukturellen und verhaltensrelevanten Informationen für die Generierung gesammelt wurden.
Automatische Generierung Barrierefreier Grafischer Benutzeroberflächen / 2018 [18]	Die Arbeit untersucht die Barrierefreiheit von Applikationen anhand eines modell-getriebenen Frameworks zur automatischen Generierung von GUI.	UCP, HTML, JavaScript, CSS	Es geht hier eher darum, wie bestehende Applikationen angepasst werden können, damit sie barrierefrei sind.

**Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here. Error!
Use the Home tab to apply Überschrift 1 to the text that you want to appear here.**

<p>A Model Driven Approach for generating Graphical User Interface for MVC Rich Internet Application / 2016 [19]</p>	<p>Der beschriebene Ansatz umfasst die Modellierung des Anwendungskontexts, daraus die Generierung des GUI-Modells anhand Beschreibung von Struktur und Verhalten der Komponenten und danach die Implementierung der GUI-Komponenten.</p>	<p>Modellgetriebener Ansatz, QVT, JavaFX, Acceleo</p>	<p>Die Umsetzung mit Acceleo ist nicht vorgesehen in dieser Arbeit.</p>
<p>Generating User Interface from Conceptual, Presentation and User models with JMermaid in a learning approach / 2015 [20]</p>	<p>In dieser Arbeit wird die automatische Generierung von Benutzeroberflächen basierend auf der Verwendung von Präsentations-, Konzept- und Benutzermodellen vorgestellt. Dabei generiert das System das UI basierend auf einer Kombination von Elementen aus den Modellen.</p>	<p>Modelbasierter Ansatz, JMermaid, MERODE, Java, XML, AUI</p>	<p>In dieser Arbeit hat es keine Informationen, welche relevant sein könnten für das Projekt.</p>
<p>Automatic Generation of Tailored Accessible User Interfaces for Ubiquitous Services / 2015 [21]</p>	<p>Die Arbeit beschreibt die automatische Generierung barrierefreier adaptiver Benutzeroberflächen für ubiquitäre Dienste. Dafür wird ein Modell genutzt, welches Nutzer, Kontext und die Benutzeroberfläche mittels UIML beschreibt. Das Modell wird mittels Interaktionen zwischen dem Nutzer und dem System aktualisiert.</p>	<p>Modelbasierter Ansatz, UIML, Ontologie</p>	<p>UIML könnte interessant für die Umsetzung des UI sein.</p>
<p>GUI generation based on user interface guidelines / 2013 [22]</p>	<p>Die Arbeit befasst sich mit der Generierung von UI basierend auf den Richtlinien für Benutzeroberflächengestaltung. Dazu werden die UI-Richtlinien analysiert und entsprechendes UI generiert.</p>	<p>JavaCC</p>	<p>Bei dieser Arbeit geht es eher darum, bestehendes GUI anzupassen an UI-Richtlinien.</p>
<p>A Grey-box Approach for Automated GUI-</p>	<p>In dieser Arbeit wird ein «Grey-Box»-Ansatz zur automatischen Generierung von UI verwendet,</p>	<p>Java, Android, Machine Learning</p>	<p>Der Ansatz Machine Learning wird in</p>

**Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here. Error!
Use the Home tab to apply Überschrift 1 to the text that you want to appear here.**

Model Generation of Mobile Applications / 2013 [23]	wobei statische und dynamische Analysen durchgeführt werden, um Informationen zur Applikation und den Benutzerinteraktionen zu sammeln. Die Analyse wird als Grundlage für das GUI-Modell verwendet, worauf dann das UI basiert.		dieser Arbeit nicht weiterverfolgt.
---	--	--	-------------------------------------

**Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here. Error!
 Use the Home tab to apply Überschrift 1 to the text that you want to appear here.**

3.2 Tools für die automatische Generierung von Benutzeroberflächen

Name	Funktionsweise	Verwendete Technologien	Fazit
Autogui [24]	Autogui ist eine Bibliothek für automatische Generierung von Java/Swing Applikationen aus POJOs (Plain Old Java Objekten). Dabei werden Swing-basierte Komponenten aus den Klassendefinitionen der vordefinierten Objekte erstellt.	Modell-basierter Ansatz, Java, Swing, jshell	Der Umgang mit Eingabeparametern für die UI-Generierung kann näher angeschaut werden.
Openxava [25]	OpenXava ist ein Webframework zur automatischen UI-Generierung in Webanwendungen mittels Deklaration von POJOs für Businessanwendungen.	Modell-basierter Ansatz, Java	Der Bezug zu Businessanwendungen könnte für dieses Projekt interessant sein.
EntityUI [26]	EntityUI ist ein Tool zur automatischen Erstellung von UI, indem es die geschriebenen Entitätsklassen analysiert und daraus das UI erstellt.	Modell-basierter Ansatz, C#	Da keine Datenbank für die Umsetzung benötigt wird, ist dieser Ansatz mit Entitätsklassen nicht relevant.
RacketUI [27]	RacketUI ist eine Bibliothek für einfache Generierung von Benutzeroberflächen mittels Racket. Die Benutzeroberfläche wird auf Grundlage einer annotierten Spezifikation der von der zugrundeliegenden Funktion produzierten Datentypen erstellt.	Modell-basierter Ansatz, Racket	Der verwendete Ansatz scheint für das Projekt keine relevanten Inputs zu haben.
UI-Schema [28]	JSON-Schema ist ein Tool zur Beschreibung von JSON-Datenstrukturen worauf basierend Benutzeroberflächen generiert werden.	JSON, React	Hier werden die Daten und das UI-Schema im JSON-Format festgelegt, was für dieses Projekt relevant ist.

3.3 Erkenntnisse aus der Recherche

Die Recherche zum Thema «automatische Generierung von Benutzeroberflächen» hat ergeben, dass es keine vorhandene Lösung gibt, welche genau auf das Business System enablerr passt. Es gibt jedoch

**Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here. Error!
Use the Home tab to apply Überschrift 1 to the text that you want to appear here.**

einige vorhandene Ansätze, welche bei der Entwicklung des Lösungskonzepts von Nutzen sein könnten.

Die meisten der vorgestellten Ansätze und Tools sind modellbasiert und fokussieren sich auf den Benutzerkontext für die UI-Generierung. Ein solcher Ansatz ist für diese Lösung aufwändig, da man die vorhandenen Datenstrukturen von enablerr erst umwandeln müsste in ein passendes Format für die Weiterverarbeitung. UI-Schema kann als Grundlage für das Lösungskonzept hinzugezogen werden. Es verarbeitet JSON-Daten anhand eines JSON-Schemas und generiert daraus eine Benutzeroberfläche. Eine ähnliche Lösung für enablerr wäre sinnvoll, da enablerr bereits ein JSON-Schema für die Datenvalidierung zur Verfügung stellt und die Ereignisse ebenfalls im JSON-Format vorhanden sind.

Noch ein weiterer interessanter Punkt ist das Informationssammelschema, um die relevanten Informationen für die Generierung der UI-Komponenten zu sammeln und zu sortieren. Beispielsweise teilt eine Arbeit die Informationen in strukturelle und verhaltenstechnische Informationen [29].

Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here. Error!
Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here.

4 Entwurf eines JSON-zu-HTML-Transformators

In diesem Kapitel wird das Lösungskonzept für die Umsetzung einer adaptiven Benutzeroberfläche vorgestellt. Es beschreibt konzeptuell eine Lösung, um aus JSON-Daten HTML-Elemente zu generieren. Zuerst wird auf die Grundidee, inklusive Analyse der vorhandenen JSON-Struktur und Lösungsstruktur eingegangen. Danach werden die Anforderungen an die Lösung und deren Akzeptanzkriterien vorgestellt. Abschliessend wird die Integration des Adaptive Grid festgelegt.

4.1 Prinzip

Die Lösung soll eine Aufgabenbeschreibung entgegennehmen und daraus eine Benutzeroberfläche erstellen, welche weiterverwendet werden kann. Dabei soll der Benutzer die Gestaltung der Benutzeroberfläche mitbestimmen können.

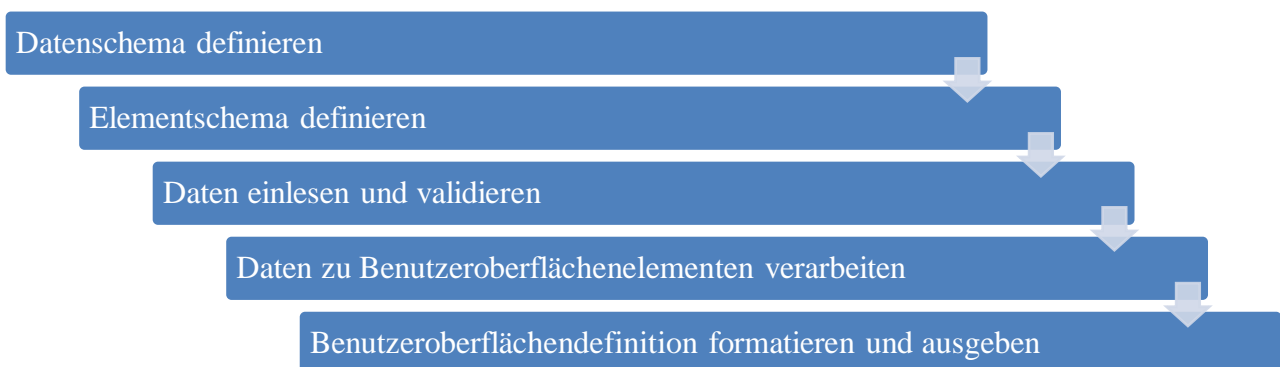


Abbildung 13 HTML-Generierungsprozess

In Abbildung 13 wird der Prozess in fünf Schritten beschrieben. Im Folgenden wird detailliert auf die Schritte eingegangen:

Datenschema definieren

Bevor die Beschreibung einer Aufgabe verarbeitet werden kann, muss definiert sein, wie eine Aufgabe aufgebaut sein muss und was für Informationen in welchem Format sie enthalten kann. Wie in Kapitel 2.1.2 beschrieben ist ein solches Schema notwendig, um sicherzustellen, dass die Lösung mit den eingelesenen Daten umgehen kann. Den eine Lösung benötigt eine definierte Datenstruktur, damit sie weiss, welche Daten wie zu verarbeiten sind. Dies kann der Benutzer anhand eines Datenschemas festlegen. Das Schema soll klar aufzeigen, welchem Muster eine Beschreibung folgen muss, welche Eigenschaften darin enthalten sein sollen und welche Werte diese Eigenschaften annehmen können.

```
{
  "$id": "https://example.com/arrays.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "description": "A representation of a person, company, organization, or place",
  "type": "object",
  "properties": {
    "fruits": {
      "type": "array",
      "items": {
        "type": "string"
      }
    },
    "vegetables": {
      "type": "array",
      "items": { "$ref": "#/$defs/veggie" }
    }
  },
  "$defs": {
    "veggie": {
      "type": "object",
      "required": [ "veggieName", "veggieLike" ],
      "properties": {
        "veggieName": {
          "type": "string",
          "description": "The name of the vegetable."
        },
        "veggieLike": {
          "type": "boolean",
          "description": "Do I like this vegetable?"
        }
      }
    }
  }
}
```

Abbildung 14 Beispiel eines Datenschemas im JSON-Format [29]

Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here. Error!
Use the Home tab to apply Überschrift 1 to the text that you want to appear here.

Elementschema definieren

Neben dem Datenschema ist wichtig, dass der Benutzer ein Elementschema definiert. Dieses legt fest, wie die im Datenschema erwähnten Eigenschaften auf der Benutzeroberfläche dargestellt werden sollen. Das Elementschema enthält Anweisungen dazu, wie eine Eigenschaft auszuschauen hat. Beispielsweise legt der Benutzer fest, dass eine Eigenschaft als Überschrift mit der Schriftgröße 16 dargestellt werden soll. Pro Eigenschaft kann festgelegt werden, was für ein HTML-Element daraus werden soll, wie dieses Element mittels CSS gestylt werden soll und was für einen Klassennamen es erhalten soll. Zudem kann das Aussehen von verschachtelten Eigenschaften weiter spezifiziert werden, indem ein subLevel definiert wird. Eigenschaften, welche nicht im Elementschema definiert sind, werden nicht verarbeitet.

```
{
  fruits: {
    tag: 'div',
    class: 'fruits-class',
    sublevel: {
      items: {
        tag: 'p',
        style: 'font-weight: bold;'
      }
    }
  },
  vegetables: {
    tag: 'div',
    class: 'veggies-class',
    sublevel: {
      veggieName: {
        tag: 'h3',
        style: 'font-style: italic;'
      }
    }
  }
}
```

Abbildung 15 Beispiel eines Elementschemas

Elementschema validieren

Bevor das Elementschema eingelesen wird, ist sicherzustellen, dass die im Elementschema enthaltenen Informationen valide sind. Dabei überprüft die Lösung, ob die Angaben im Elementschema ein valides Format aufweisen und ob auf deren Basis Elemente auf der Benutzeroberfläche generiert werden können.

Daten einlesen und validieren

Nachdem festgelegt wurde was für Eigenschaften wie visualisiert werden sollen, kann der Benutzer konkrete Daten einlesen.

Die Daten werden eingelesen und es wird nun geprüft, ob die Daten dem definierten Datenschema entsprechen. Dabei wird für jede Eigenschaft geprüft, ob sie im Datenschema vorhanden ist und ob das Format des mitgegebenen Wertes stimmt.

```
{
  "fruits": [ "apple", "orange", "pear" ],
  "vegetables": [
    {
      "veggieName": "potato",
      "veggieLike": true
    },
    {
      "veggieName": "broccoli",
      "veggieLike": false
    }
  ]
}
```

Abbildung 16 Beispiel von Daten definiert im JSON-Format [29]

Daten zu Benutzeroberflächenelementen verarbeiten

Sind die Daten valide beginnt die Verarbeitung der Daten Eigenschaft für Eigenschaft. Jede Eigenschaft durchläuft folgende Schritte:

1. Es wird geprüft, ob das Aussehen der Eigenschaft im Elementschema festgelegt wurde.
 - a. Wenn ja, wird das Element in Schritt 2 weiterverarbeitet.
 - b. Wenn nein, wird die nächste Eigenschaft in Schritt 1 weiterverarbeitet.
2. Zuerst wird der Wert der Eigenschaft verarbeitet.
 - a. Handelt es sich dabei um ein oder mehrere Objekte, durchlaufen diese rekursiv nochmals diesen Prozess und werden in das ursprüngliche Element eingefügt.
 - b. Handelt es sich um eine Zeichenkette oder eine Nummer, wird der Wert zurückgegeben.

**Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here. Error!
Use the Home tab to apply Überschrift 1 to the text that you want to appear here.**

3. Danach wird geprüft, ob im Elementschema für verschachtelte Werte innerhalb der Eigenschaft weitere Angaben zur Formatierung auf der Benutzeroberfläche gemacht wurden.
 - a. Wenn ja, werden die Werte innerhalb der Eigenschaft wiederum rekursiv verarbeitet, indem sie Schritt 2 durchlaufen.
 - b. Wenn nein, wird das Element für die Benutzeroberfläche zusammengestellt.
4. Nun wird geprüft, ob die Eigenschaft im Elementschema als Beschriftung bezeichnet wird.
 - a. Wenn ja, wird ein Eingabefeld und weiterverarbeitet.
 - b. Wenn nein, wird aus den Angaben zum Aussehen der Eigenschaft aus dem Elementschema ein Benutzeroberflächen-Element erstellt, welches den Wert aus Schritt 2 zum Inhalt hat.
5. Es wird geprüft, ob Einschränkungen oder Vorgaben für das Eingabefeld aufgeführt sind.
 - a. Wenn ja, werden die Einschränkungen oder Wertvorgaben verarbeitet und es wird ein Benutzeroberflächen-Element erstellt. Die Einschränkungen oder Wertvorgaben werden dabei in das Element eingebaut.
 - b. Wenn nein, wird ein Eingabefeld ohne Einschränkungen und Vorgaben erstellt.
6. Wurden alle inneren Informationen einer Eigenschaft verarbeitet, wird das im Generierungsprozess entstandene Element in das Gesamtdokument eingebaut.

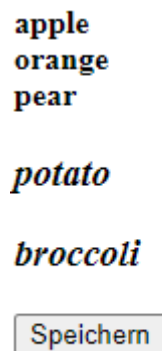
Benutzeroberflächendefinition formatieren und ausgeben

Wurden alle Daten verarbeitet, wird die generierte Benutzeroberflächendefinition überprüft auf leere oder undefinierte Elemente, welche dann entfernt werden. Danach wird die Benutzeroberflächendefinition so formatiert, dass die Struktur des Gesamtdokuments ersichtlich ist. Die formatierte Definition wird zuletzt an den Benutzer ausgegeben in Textform.

```
<form id="htmlForm">
  <div style="" class="fruits-class">
    <div>
      <div>
        <div style="font-weight: bold;">apple</div>
      </div>
      <div>
        <div style="font-weight: bold;">orange</div>
      </div>
      <div>
        <div style="font-weight: bold;">pear</div>
      </div>
    </div>
  </div>
  <div style="" class="veggies-class">
    <div>
      <div>
        <h3 style="font-style: italic;">potato</h3>
      </div>
      <div>
        <h3 style="font-style: italic;">broccoli</h3>
      </div>
    </div>
  </div>

  <input id="submitFormInput" type="submit" value="Speichern">
</form>
```

Abbildung 18 Vom JUIBuilder generierter HTML-Code



apple
orange
pear

potato

broccoli

Speichern

Abbildung 17 Im Browser geöffneter generierter HTML-Code

**Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here. Error!
Use the Home tab to apply Überschrift 1 to the text that you want to appear here.**

4.2 Anpassbarkeit und Erweiterbarkeit

Benutzer sollen die Möglichkeit haben das Styling der Benutzeroberflächenelemente bereits vorgängig in der Datenabbildungskonfiguration einfügen zu können. Zusätzlich sollen die Elemente im Nachhinein anpassbar sein, indem der Benutzer den Elementen einen Klassennamen gibt. Die Elemente können nach ihrer Generierung über den Klassennamen aufgerufen und angepasst werden.

4.3 Anforderungen an die Lösung

Als Benutzer möchte ich benutzerdefinierte Inhalte über JSON in meine Website einbinden können, um die Darstellung dynamisch anzupassen.

Akzeptanzkriterien:

- Ich kann JSON-Daten gemäss vorgegebenem Schema eingeben und die Lösung generiert daraus das entsprechende HTML.
- Die Lösung stellt sicher, dass das generierte HTML korrekt und fehlerfrei angezeigt wird.

Als Benutzer möchte ich das Erscheinungsbild der aus dem JSON generierten HTML-Elemente anpassen können.

Akzeptanzkriterien:

- Ich kann CSS-Stile auf das generierte HTML anwenden, um das visuelle Erscheinungsbild anzupassen.
- Die Lösung ermöglicht mir, Klassen zu den HTML-Elementen hinzuzufügen, um sie gezielt zu stylen.

4.4 Integration des Adaptive Grids

Da das Konzept des Adaptive Grids und dieses Konzept unterschiedliche Ziele verfolgen, macht eine Integration des Adaptive Grid in diese Lösung keinen Sinn. Diese Lösung soll eine Benutzeroberfläche generieren, welche vom Benutzer der Lösung weiterverarbeitet wird. Dabei kann der Benutzer die Gestaltung der Daten auf der Benutzeroberfläche selbst bestimmen. Dabei kann nicht davon ausgegangen werden, dass der Benutzer die Daten auf einem Grid dargestellt haben will.

**Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here. Error!
Use the Home tab to apply Überschrift 1 to the text that you want to appear here.**

5 Realisierung des JUIBuilder's

In diesem Kapitel wird beschrieben, wie aus der Idee aus einer Aufgabenbeschreibung die Benutzeroberfläche automatisiert generieren zu lassen der JUIBuilder umgesetzt wurde. Der Fokus liegt auf den Technologien und der Architektur der Lösung. Es werden die Module 'JUIBuilder', 'DataProcessor', 'HtmlUtils' und 'SchemaUtils' sowie deren Funktionalität vorgestellt. Zuletzt werden die Limitationen und Fehlerbehandlungsstrategien vorgetragen.

5.1 Programmiersprachen und Technologien

Für die Umsetzung des JUIBuilder wurde hauptsächlich JavaScript verwendet. Die Lösung mittels JavaScript zu realisieren macht Sinn, da es plattformunabhängig ist und zudem mit HTML und CSS, welche ebenfalls verwendet wurden in dieser Lösung, nahtlos integrierbar ist.

Da enablerr für die Aufgabenbeschreibung das JSON-Format verwendet wurde die Lösung danach ausgerichtet. Sowohl die verarbeiteten Daten, als auch das Schema, wie auch die Datenabbildungskonfiguration orientieren sich am JSON-Format. Die Ereignisbeschreibungen können im gleichen Format, wie es enablerr benötigt, belassen werden. Es sind weder Ergänzungen an den Beschreibungen nötig, damit diese von der Lösung verarbeitet werden können, noch müssen ausser dem Datenschema andere Informationen aus dem System hinzugezogen werden.

Da HTML und CSS in der Webentwicklung weit verbreitet sind und die Erstellung der Benutzeroberfläche möglichst wenig neuen Lernaufwand für Benutzer der Lösung darstellen sollte wurden sie für die Definition der Datenabbildungskonfiguration festgelegt und für die Generierung der Benutzeroberfläche benutzt.

Für die Datenvalidierung und die Formatierung des generierten HTML Codes wurden JavaScript-Bibliotheken eingesetzt.

Für die Versionskontrolle der Lösung wurde GitHub verwendet.

5.2 Architektur und Design

Die Lösung ist in die Klassen JUIBuilder, DataProcessor, HtmlUtils und SchemaUtils unterteilt, welche verschiedene Zuständigkeiten haben.

JUIBuilder stellt den Eintrittspunkt in die Lösung dar und ist die Hauptklasse, welche die Daten zur Generierung entgegennimmt und im gewünschten Format zurückgibt. Sie verwendet die Utility Klassen DataProcessor, HtmlUtils und SchemaUtils, um dies zu erreichen.

HtmlUtils führt die Validierung der Datenabbildungskonfiguration durch. Die Datenabbildungskonfiguration wird auf Korrektheit der HTML Elemente und deren Styling geprüft. Zudem formatiert HtmlUtils den auszugebenden HTML-Code.

SchemaUtils behandelt die einkommenden Daten und das Datenschema. Das Datenschema wird auf syntaktische und semantische Korrektheit geprüft und die einkommenden Daten, darauf, ob sie in das angegebene Datenschema passen.

DataProcessor ist zuständig für das Verarbeiten der einkommenden Daten. Es durchläuft diese und generiert daraus HTML Elemente.

**Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here. Error!
 Use the Home tab to apply Überschrift 1 to the text that you want to appear here.**

DataProcessor
listCount: number elementConfigMap: string
processArgument(key, argumentValue): any string string undefined processSublevelObject(elementConfig, value): string processValue(value, key): string handleValueUnitCurrency(value): string processInput(properties): string string string processList(listData): void processObject(objectData, elementConfigMap: null): string processItemProperties(itemProperties): string string processArray(arrayData, key): string addButtonEventListener(): string

JUIBuilder
constructor(elementConfigMap: null):
validateHtml(configMap): boolean boolean validateSchema(data, schema: null): boolean boolean processEventData(jsonData, jsonSchema: null): string any processObject(objectData, elementConfigMap): string

HtmlUtils
validateHtml(configMap): boolean boolean validateHtmlTag(tag): boolean undefined validateStyling(style): boolean undefined removeUndefinedOrEmptyElements(htmlMarkup): any processHtmlMarkup(htmlMarkup): any

SchemaUtils
validateSchema(data, schema, jsonSchema, elementConfigMap): boolean boolean prepareSchema(schema, jsonSchema, elementConfigMap): any undefined disallowAdditionalProperties(schema): void validateData(data, jsonSchema): boolean boolean

Abbildung 19 Klassen des JUIBuilder's

5.3 Datenstrukturen und Algorithmen

Die Lösung fokussiert sich vor allem auf das Verarbeiten und Transformieren von Daten. Dabei werden verschiedene Datenstrukturen verarbeitet. Die Lösung nimmt Daten eines JSON-Objekts entgegen und verarbeitet daraus primitive Datentypen, wie ganze Zahlen, Wahrheitswerte und Kommazahlen. Ebenfalls werden Arrays, welche wiederum JSON-Objekte enthalten, sowie Listen, behandelt.

In der Lösung werden algorithmische Ansätze verwendet. Das entgegengenommene JSON-Objekt und dessen verschachtelten Objekte und Arrays werden rekursiv verarbeitet, indem überprüft wird, ob sich ein verschachteltes Objekt oder ein Array im Datensatz befindet. Ist dies der Fall, wird die Verarbeitungsmethode nochmals aufgerufen und beendet zuerst die Verarbeitung des verschachtelten Objekts, bevor das übergeordnete Objekt weiterverarbeitet wird.

**Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here. Error!
 Use the Home tab to apply Überschrift 1 to the text that you want to appear here.**

5.4 Funktionen und Module

Die Lösung hat ein modulares Design, wobei jedes Modul seine eigene Zuständigkeit hat. Dies verbessert die Skalierbarkeit und Wartbarkeit der Lösung und minimiert Wiederholungen derselben Funktionalität innerhalb der Lösung.

Modul	Methode	Funktionalität
JUIBuilder	constructor	Zur Kreierung einer Instanz des JUIBuilder's. Der Instanz kann eine 'elementConfigMap' (auch Datenabbildungskonfiguration) mitgegeben werden, an welcher sich die Lösung für die Generierung der Benutzeroberflächenelemente orientiert.
	processEventData(jsonData, jsonSchema)	Hauptmethode der Lösung, welche die Validierungsmethoden der Utility Klassen, sowie die Datenverarbeitungsmethoden aufruft und schlussendlich den formatierten HTML-Code zurückgibt. Sie nimmt die zu verarbeitenden Daten und optional ein eigenes Datenschema entgegen.
HtmlUtils	validateHtml(configMap)	Validiert die HTML Konfiguration in der 'elementConfigMap'
	validateHtmlTag(tag)	Validiert die HTML tags des Elements
	validateStyling(style)	Validiert den CSS Style des HTML Elements
	removeUndefinedOrEmptyElements(htmlMarkup)	Entfernt undefinierte oder leere HTML Elemente aus dem HTML Code

**Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here. Error!
 Use the Home tab to apply Überschrift 1 to the text that you want to appear here.**

	processHtmlMarkup(htmlMarkup)	Formatiert den HTML Code
SchemaUtils	validateSchema(data, schema, jsonSchema, elementConfigMap)	Validiert die zu verarbeitenden Daten mit dem Datenschema
	prepareSchema(schema, jsonSchema, elementConfigMap)	Verbietet zusätzliche Eigenschaften, falls ein eigenes Datenschema mitgegeben wurde
	disallowAdditionalProperties(schema)	Ändert das Datenschema, so dass keine zusätzlichen Eigenschaften hinzugefügt werden können
DataProcessor	processObject(objectData, elementConfigMap)	Verarbeitet ein JSON-Objekt und generiert daraus ein HTML Element
	processArray(arrayData, key)	Verarbeitet ein JSON-Array innerhalb des JSON-Objekts und ruft rekursiv Methoden zur Weiterverarbeitung der Array-Elemente auf
	processList(listData)	Generiert einen Index für die verschiedenen Listen innerhalb der mitgegebenen Daten
	processElement(elementConfig, value, keys, currentIndex, objectData)	Verarbeitet eine JSON-Eigenschaft und generiert daraus ein HTML-Element
	processValue(value, key)	Überprüft, ob der Inhalt eines JSON-Objekts als Inhalt des HTML-Elements eingefügt oder weiterverarbeitet wird
	processSublevelObject(elementConfig, value)	Verarbeitet Daten, deren Aussehen innerhalb eines Sublevels in der 'elementConfigMap' definiert wurden

**Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here. Error!
 Use the Home tab to apply Überschrift 1 to the text that you want to appear here.**

	<code>processInput(properties)</code>	Verarbeitet HTML Input Elemente
	<code>processItemProperties(itemProperties)</code>	Generiert HTML Input Elemente basierend auf den mitgegebenen Dateneigenschaften
	<code>handleValueUnitCurrency(value)</code>	Verarbeitet vorgegebene Datenwerte der Input Elemente
	<code>processArgument(key, argumentValue)</code>	Verarbeitet weitere Eigenschaften innerhalb der Werteinschränkungen der Input Elemente
	<code>addButtonEventListttener()</code>	Gibt den Knöpfen für die Listenelemente Funktionalität zur Erstellung weiterer Objekte

5.5 Fehlerbehandlung

Die Fehlerbehandlung während der Validierung wird durch Logging und try-catch-Blöcke umgesetzt. Fehler werden in der Konsole angezeigt. Daten, deren Aussehen nicht in der 'elementConfigMap' definiert wurde, werden nicht verarbeitet.

6 Evaluierung der Lösung «JUIBuilder»

In diesem Kapitel stehen die Evaluierung und Optimierung der umgesetzten Lösung im Mittelpunkt. Im ersten Abschnitt wird auf die durchgeführten Benutzertests eingegangen. Danach werden die gesammelten Verbesserungsvorschläge analysiert und die Umsetzung des Feedbacks angeschaut. Abschliessend wird im Rahmen der finalen Evaluierung eine Benutzerbewertung angeschaut und das Resultat der fertigen Lösung mit der aktuellen Benutzeroberfläche verglichen.

6.1 Ablauf der Benutzertests

Um die umgesetzte Lösung auf Erfüllung der Anforderungen aus Kapitel 4.3 zu prüfen, wurden vier Personen mit Kenntnissen in der Entwicklung von Benutzeroberflächen Aufgaben gestellt, welche sie mit Hilfe des JUIBuilder's lösen sollten. Danach haben die Personen Fragen zur Qualität der Lösung beantwortet.

Die Benutzer erhielten Zugriff auf die Lösung und haben diese in einer eigenen Entwicklungsumgebung geöffnet. Die Benutzer erhielten die Anweisung zuerst das README zu lesen und auf Grundlage dessen den JUIBuilder auszutesten. Um die Lösung testen zu können, wurden sechs verschiedene

**Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here. Error!
Use the Home tab to apply Überschrift 1 to the text that you want to appear here.**

Datenbeispiele zur Verfügung gestellt, wobei drei davon dem Datenschema von enablerr entsprachen. Als Leitfaden wurden drei Testfälle vorbereitet, wovon einer ein Beispiel für enablerr abdeckte.

Die Fragen überprüften anschliessend die Funktionalität, Benutzerfreundlichkeit und Leistung der Lösung.

6.2 Verbesserungsvorschläge und Umsetzung des Feedbacks

Insgesamt wurde das Konzept und die Umsetzung vor allem für den Testfall mit einem Datenbeispiel aus enablerr als gut empfunden, jedoch gab es auch einige Kritikpunkte.

6.2.1 Unklare Fehlermeldungen

Als die Lösung evaluiert wurde gab es zwei Fehlermeldungen. Eine Fehlermeldung kam auf, wenn das HTML im Elementschema nicht korrekt war und die zweite, wenn die eingelesenen Daten nicht mit dem Datenschema übereinstimmten. Dies empfanden die Tester als zu wenig genau, um den Fehler gezielt zu finden.

Im Nachgang wurden beide Fehlermeldungen so spezifiziert, dass die genaue Stelle, an welchem der Fehler auftritt ausgegeben wird.

6.2.2 undefinierte Elemente im generierten Code

Sowohl Elemente, wie auch Elementinhalte, wiesen zum Teil den Wert "undefined" auf.

In der fertig umgesetzten Lösung wird im Falle das ein Elementinhalt nicht erkannt wird statt "undefined" ein Platzhalter mit dem Wert gesetzt. Elemente mit dem Tag "undefined" oder leere Elemente werden vor Ausgabe des generierten Codes gelöscht.

6.2.3 Nachträgliche Anpassungen

Die Lösung bot die Möglichkeit, im Elementschema den HTML-Tag und ein CSS-Styling mitzugeben. Jedoch konnte die spezifische Eigenschaft nicht nachträglich angepasst werden.

Durch Hinzufügen von Klassen im Elementschema, welche die Benutzer setzen können, lassen sich die Elemente nun im Nachhinein aufrufen und anpassen.

6.2.4 Bessere Benutzerfreundlichkeit für Anfänger

Das Konzept mit dem Elementschema, Datenschema und Datenbeispiel wurde trotz Anwendungsbeispiel im README als etwas schwierig empfunden.

Im README wurde die Anwendung noch etwas genauer beschrieben.

6.2.5 Styling-Auslagerung

Um die Wiederholung von Code zu minimieren wurde empfohlen, das Styling in eine separate Datei auszulagern.

Aus zeitlichen Gründen wurde dieser Vorschlag nicht umgesetzt, da die Anwendungslogik stark hätte angepasst werden müssen.

Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here. Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here.

6.3 Finale Evaluierung der Lösung

Die optimierte Lösung wurde an die Hauptentwicklerin der Benutzeroberfläche von enablerr für eine finale Evaluierung übergeben. Dabei wurden Effizienz, Lernkurve, Fehlerbehebung, Dokumentation und Gesamterfahrung der aktuellen Benutzeroberflächengenerierungsmethode mit der Generierung durch den JUIBuilder verglichen.

Grosse Aufgabenbeschreibungen, wie diese in enablerr vorkommen, werden wesentlich schneller generiert, als diese manuell programmiert werden können.

Die Lernkurve für die Verwendung des JUIBuilder's ist gering. Für Modifikationen am generierten HTML oder für die Anwendung eines eigenen Stylings ist ein gewisses Vorwissen notwendig.

Fehlermeldungen wurden als leicht verständlich wahrgenommen und machten die Fehlerbehebung schnell und einfach.

Die Dokumentation ist gut strukturiert und enthält alle wichtigen Informationen.

Insgesamt wurde die Lösung auf einer Skala von 1 für sehr schlecht bis 10 für sehr gut mit einer 8 positiv bewertet. Das Generierungskonzept wurde vielversprechend angesehen und könnte mit einigen Erweiterungen produktiv eingesetzt werden.

6.4 Vergleich der generierten und der programmierten Benutzeroberfläche

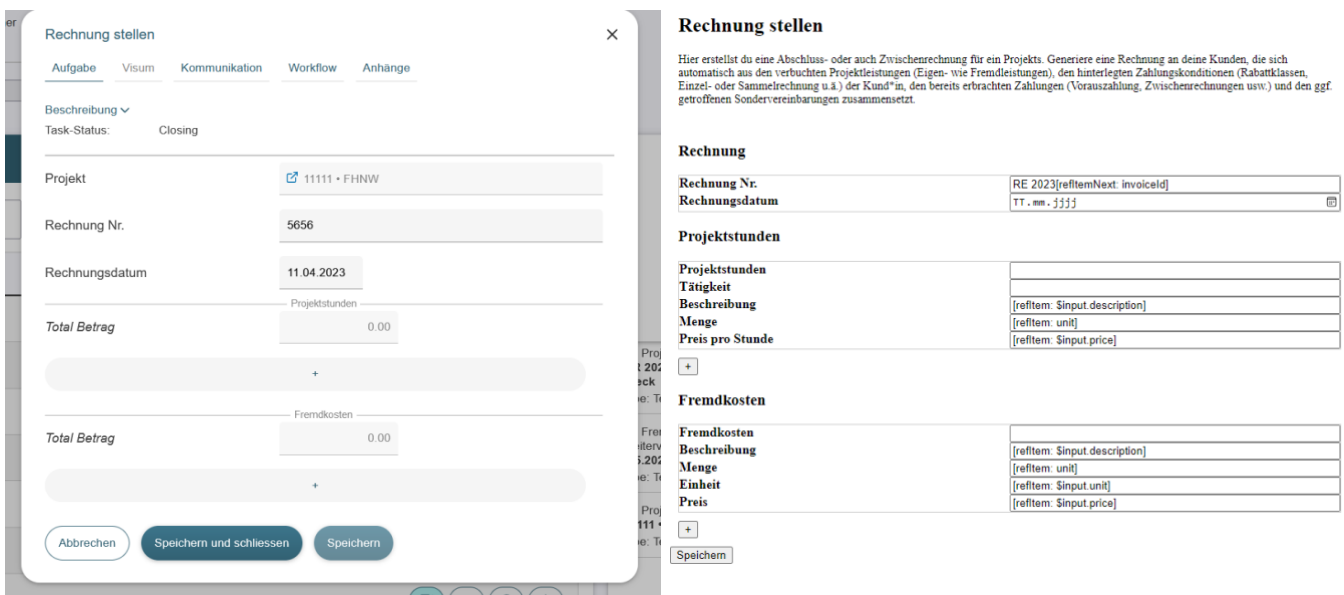


Abbildung 20 Gegenüberstellung manuell programmierte und generierte Benutzeroberfläche

Im direkten Vergleich fallen einige Unterschiede auf. Die Benutzeroberfläche auf der linken Seite enthält einige Informationen, welche nicht der Aufgabenbeschreibung im JSON-Format entnommen wurden, sondern nachträglich ergänzt, wie beispielsweise "Task-Status". Ebenfalls verfügt die Benutzeroberfläche links über weitere Unterseiten, welche nicht in der Aufgabenbeschreibung ersichtlich sind. Was zudem auffällt ist, dass links Datenbeziehungen, wie beispielsweise die Referenz im Feld "Projekt" verlinkt werden können.

**Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here. Error!
Use the Home tab to apply Überschrift 1 to the text that you want to appear here.**

Die relevanten Informationen aus der Aufgabenbeschreibung sind in beiden Beispielen sichtbar. Durch die Möglichkeit eigenes Styling anzuwenden, kann die generierte Benutzeroberfläche visuell ebenfalls optimiert werden.

**Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here. Error!
Use the Home tab to apply Überschrift 1 to the text that you want to appear here.**

7 Ergebnisse und Diskussion

Im vorliegenden Kapitel werden die Ergebnisse der entwickelten Lösung für die Generierung von HTML aus JSON vorgestellt und im Kontext des aktuellen Forschungsstandes diskutiert. Dazu gehört die Leistungsfähigkeit der Lösung im Vergleich mit etablierten Ansätzen sowie die kritische Reflexion der Stärken und Schwächen der Lösung. Mit einem Ausblick in zukünftige Entwicklungsmöglichkeiten wird dieser Bericht abgeschlossen.

7.1 Ergebnisse und Vergleich zum Forschungsstand

Während der Recherche wurde festgestellt, dass die meisten Arbeiten und Tools einen modell-basierten Ansatz verfolgen. Da enablerr grösstenteils mit JSON-Strukturen arbeitet, wurde diese Lösung darauf aufgebaut, damit möglichst wenig an den Datenstrukturen verändert werden muss. Mit dem JUIBuilder wurde ein performantes und effizientes Tool geschaffen, um vor allem grosse Aufgabenbeschreibungen, wie diese in enablerr vorzufinden sind, zu Benutzeroberflächen zu verarbeiten. Der vom JUIBuilder generierte Code bietet eine solide Basis-Benutzeroberfläche, welche einfach erweiterbar ist.

7.2 Limitationen und kritische Bewertung der Lösung

Obwohl die Evaluierung der Lösung positive Ergebnisse erbrachte, gibt es einige Limitationen, welche bei der Bewertung dieser Lösung einbezogen werden sollten.

Die umgesetzte Lösung behandelt keine Datenbeziehungen. Handelt es sich bei dem Wert eines Elements um eine Referenz wird eine Platzhalter eingeblendet, welcher durch den Benutzer weiter verarbeitet werden kann. Ebenfalls durch den Benutzer anzupassen ist das Verhalten des Buttons "Speichern" am Ende des Formulars, um die gewünschte Aktion umzusetzen.

Obwohl die Lösung bereits Funktionalitäten bietet, gibt es noch Optimierungspotenzial. So könnten noch weitere Anpassungsmöglichkeiten angeboten werden, damit noch mehr Gestaltungsoptionen offenstehen.

7.3 Ausblick auf zukünftige Entwicklungsmöglichkeiten

Die entwickelte Lösung bildet die Grundlage für Erweiterungen im Bereich dynamischer, automatischer Benutzeroberflächengenerierung. Eine mögliche Erweiterung wäre die Behandlung von Datenbeziehungen und die Einbindung von Funktionen.

Eine weitere mögliche Weiterentwicklung wäre die automatische Generierung eines Datenschemas anhand einer Aufgabenbeschreibung. In dieselbe Richtung würde auch die automatische Generierung eines Elementschemas anhand einer Aufgabenbeschreibung gehen.

Ebenfalls interessant wäre aus einer HTML-Benutzeroberfläche automatisch die Aufgabenbeschreibung im JSON-Format zu generieren.

Insgesamt hat die umgesetzte Lösung gezeigt, dass die automatische Generierung von Benutzeroberflächen für enablerr und auch andere Anwendungen, welche ihre Daten im JSON-Format verarbeitet, möglich ist.

Quellenverzeichnis

- [1] "Gradient ui/ux background," freepik.com. https://www.freepik.com/free-vector/gradient-ui-ux-background_16683348.htm [Zugegriffen: 09-Mär 2023].
- [2] J. Schmidig, "Ereignis Design," enablerr, Dez. 2019.
- [3] "Das integrative Business System," enablerr.com. <https://www.enablerr.ch/> [Zugegriffen: 15-Apr 2023].
- [4] J. Schmidig, "The Workflow," enablerr, Oct. 2022.
- [5] pier4all AG, "enablerr," unpubliziert.
- [6] A. Gervalla, J. Frey, "Adaptive Grid," unpubliziert.
- [7] A. Sobolevsky, G.-A. Bilodeau, J. Cheng, und J. L.C. Guo, "GUILGET: GUI Layout GEneration with Transformer," in *The 36th Canadian Conference on Artificial Intelligence*, 2023, doi: 10.21428/594757db.08fe0a25.
- [8] X. Yao, M.H. Yap, Y. Zhang, "An Automatic GUI Generation Method Based on Generative Adversarial Network, " in *Proceedings of Seventh International Congress on Information and Communication Technology*, 2022, pp. 641–653, doi: 10.1007/978-981-19-2394-4_59.
- [9] M. Bavay, M. Reisecker, T. Egger und D. Korhammer, "Inishell 2.0: Semantically Driven Automatic GUI Generation for Scientific Models," *Geoscientific Model Development*, vol. 15, no. 2, 2022, pp. 365-378, doi: 10.5194/gmd-15-365-2022.
- [10] Z. Ruan, Y. Fukazawa und J. Shirogane, "Automatic generation of user-sensitive and application-sensitive self-adapted UI system for smartphone applications," *2022 International Congress on Human-Computer Interaction*, 2022, pp. 1-9, Jun. doi: 10.1109/HORA55278.2022.9800000.
- [11] V. Tran, M. Kolp, Y. Wautelet und S. Heng, "UI Generation for Business Data-Based Apps from Task, Domain and User Models," in *Handbook of Human Computer Interaction*. (2021). doi: 10.1007/978-3-319-27648-9_59-1.
- [12] T. Zhao, C. Chen, Y. Liu und X. Zhu, "Guigan: Learning to Generate GUI Designs Using Generative Adversarial Networks," in *Proceedings of the 43rd International Conference on Software Engineering (ICSE '21)*, 2021, pp. 748–760, doi: 10.1109/ICSE43902.2021.00074.
- [13] A. Braham, F. Buendía, M. Khemaja und F. Gargouri, "Generation of Adaptive Mobile Applications Based on Design Patterns for User Interfaces," *13th International Conference on Ubiquitous Computing and Ambient Intelligence UCAmI 2019*, Nov. 2019, doi: 10.3390/proceedings2019031019.

Quellenverzeichnis

- [14] J. Ruiz, E. Serral und M. Snoeck, "Evaluating user interface generation approaches: model-based versus model-driven development," *Software and Systems Modelling*, vol. 18, no. 10, 2019, pp. 2753-2776, doi: 10.1007/s10270-018-0698-x.
- [15] N. Rathnayake, D. Meedeniya, I. Perera und A. Welivita, "A Framework for Adaptive User Interface Generation based on User Behavioural Patterns," *2019 Moratuwa Engineering Research Conference (MERCon)*, 2019, pp. 698-703, doi: 10.1109/MERCon.2019.8818825.
- [16] A. Galizia, G. Zereik, L. Roverelli, E. Danovaro, A. Clematis und D. D'Agostino, "Json-GUI—A module for the dynamic generation of form-based web interfaces," *SoftwareX*, vol. 9, 2019, pp. 28-34, doi: 10.1016/j.softx.2018.11.007.
- [17] M. Hitz, T. Kessel und D. Pfisterer, "Automatic UI Generation for Aggregated Linked Data Applications by Using Sharable Application Ontologies," in *Model-Driven Engineering and Software Development. MODELSWARD 2017*, L. Pires, S. Hammoudi, and B. Selic (eds.), vol. 880 of *Communications in Computer and Information Science*, Springer, Cham, 2018, doi: 10.1007/978-3-319-94764-8_14.
- [18] J. K. Thöner, "Automatische Generierung Barrierefreier Grafischer Benutzeroberflächen," Diploma Thesis, Technische Universität Wien, 2018, doi: 10.34726/hss.2018.39880.
- [19] S. Roubi, M. Erramdani und S. Mbarki, "Modeling and generating graphical user interface for MVC Rich Internet Application using a model driven approach," *2016 International Conference on Information Technology for Organizations Development (IT4OD)*, 2016, pp. 1-6, doi: 10.1109/IT4OD.2016.7479249.
- [20] J. Ruiz, G. Sedrakyan und M. Snoeck, "Generating User Interface from Conceptual, Presentation and User Models with JMermaid in a Learning Approach," in *Proceedings of the XVI International Conference on Human Computer Interaction (Interacción '15)*, 2015, pp. 1-8. doi: 10.1145/2829875.2829893.
- [21] B. Gamecho et al., "Automatic Generation of Tailored Accessible User Interfaces for Ubiquitous Services," in *IEEE Transactions on Human-Machine Systems*, vol. 45, no. 5, 2015, pp. 612-623, doi: 10.1109/THMS.2014.2384452.
- [22] K. Sugiuchi, J. Shirogane, H. Iwata und Y. Fukazawa, "GUI Generation Based on User Interface Guidelines," in *Proceedings of the IADIS International Conference on Information Systems*, 2013, [Online]. Verfügbar: https://www.researchgate.net/publication/293090206_GUI_generation_based_on_user_interface_guidelines
- [23] W. Yang, M.R. Prasad und T. Xie, "A Grey-Box Approach for Automated GUI-Model Generation of Mobile Applications," in *Fundamental Approaches to Software Engineering, FASE 2013*, V. Cortellessa and D. Varró, Eds., *Lecture Notes in Computer Science*, vol. 7793, Springer, Berlin, Heidelberg, 2013, doi: 10.1007/978-3-642-37057-1_19.

Quellenverzeichnis

- [24] *AutoGUI*. [Online]. Verfügbar: <https://www.autogui.org/>
- [25] *OpenXava*. [Online]. Verfügbar: <https://www.openxava.org/ate/java-automatic-ui>
- [26] *EntityUI Automatic UI Generator*. [Online]. Verfügbar: <https://www.codeproject.com/Articles/743114/EntityUI-Automatic-UI-Generator>
- [27] *RacketUI*. [Online]. Verfügbar: <https://docs.racket-lang.org/racketui/>
- [28] *UI-Schema*. [Online]. Verfügbar: <https://ui-schema.bemit.codes/examples>
- [29] "Miscellaneous Examples," [json-schema.org](https://json-schema.org/learn/miscellaneous-examples.html). <https://json-schema.org/learn/miscellaneous-examples.html> [Zugegriffen: 14-Aug 2023].

Ehrlichkeitserklärung

Ehrlichkeitserklärung

«Hiermit erkläre ich, die vorliegende Bachelor-Thesis selbständig und nur unter Benutzung der angegebenen Quellen verfasst zu haben. Die wörtlich oder inhaltlich aus den aufgeführten Quellen entnommenen Stellen sind in der Arbeit als Zitat bzw. Paraphrase kenntlich gemacht. Diese Bachelor-Thesis ist noch nicht veröffentlicht worden. Sie ist somit weder anderen Interessierten zugänglich gemacht noch einer anderen Prüfungsbehörde vorgelegt worden.»

Windisch, 18.08.2023

Name: Ana Cosic

Unterschrift:

Anhang

A Aufgabenstellung im Originalwortlaut

23FS_IIT03: Automatic Generation of User Interfaces

Advisor: [Norbert Seyff](#)
[Nitish Palkar](#)

Work scope:	Priority 1 P6 (360h pro Student)	Priority 2 ---
Team size:	2er Team	---

Languages: German or English
Study course: Computer Science



Initial position

The term "adaptive user interface" is often used to refer to a UI that adapts itself based on the user's current context (e.g., current location, screen size). It is also often used interchangeably (and wrongly) with "responsive design." There are no solutions that allow UI to be adapted at run time. There are also special cases when user interfaces are needed to be generated automatically and adapted to the current context.

Objective

We have developed a framework prototype in a previous project, which we call "Adaptive Grid." It is an intelligent grid system that adapts UI elements at run time. The goal of the project is to extend and adapt the current solution to support the generation and adaptation of user interfaces within the enablerr platform. enablerr is a novel business software we are developing within an Innosuisse project. A particular extension we are envisioning is the generation of user interfaces based on enablerr task descriptions (e.g., issuing a bill for a customer) and the generation of visualization of data relevant to the task.

Problem statement

Key project challenges include:

- Investigate existing work (literature and tools/frameworks) in the area of UI generation and adaptation.
- Understand the current "adaptive grid" solution and the enablerr business software.
- Create a conceptual solution regarding the generation and adaptation of enablerr user interfaces.
- Adaption and extension of the existing Adaptive Grid solution and integration within the enablerr solution.
- Validation based on different task descriptions within enablerr, e.g., in the context of an industrial use case.

Technologies/Technical emphasis/References

- Adaptive Grid is based on CSS, HTML, and JS
- Enablerr is based on Svelte

B JSON-Schema Ereignis «Rechnung stellen»

```
{
  "authorization": [
    {
      "role": "Projektverantwortlicher",
      "activity": "edit"
    }
  ],
  "identity": {
    "name": "createInvoice",
    "designation": "Rechnung stellen",
    "description": "Hier erstellst du eine Abschluss- oder auch Zwischenrechnung für ein Projekts.
\nGeneriere eine Rechnung an deine Kunden, die sich automatisch aus den verbuchten Projektleistungen
(Eigen- wie Fremdleistungen), den hinterlegten Zahlungskonditionen (Rabattklassen, Einzel- oder
Sammelrechnung u.ä.) der Kund*in, den bereits erbrachten Zahlungen (Vorauszahlung, Zwischenrechnungen
usw.) und den ggf. getroffenen Sondervereinbarungen zusammensetzt.",
    "circle": "Purpose",
    "domain": "Accounting"
  },
  "input": {
    "class": [
      {
        "name": "project",
        "selection": "project"
      },
      {
        "name": "projectHours",
        "destination": "project.projectHours"
      },
      {
        "name": "thirdPartyCost",
        "destination": "project.thirdPartyCost"
      }
    ]
  },
  "output": {
    "class": [
      {
        "name": "invoice",
        "designation": "Rechnung",
        "display": [
          {
```

B JSON-Schema Ereignis «Rechnung stellen»

```
        "type": "list",
        "refItem": [
            "date",
            "invoiceId"
        ]
    },
],
"item": [
    {
        "name": "invoiceId",
        "designation": "Rechnung Nr.",
        "type": "string",
        "properties": {
            "required": true,
            "unique": true,
            "value": {
                "concatenation": [
                    {
                        "constant": "RE"
                    },
                    {
                        "constant": " "
                    },
                    {
                        "variable": "year"
                    },
                    {
                        "refItemNext": {
                            "refItem": "invoiceId",
                            "offset": "7,4",
                            "type": "table",
                            "step": 1,
                            "base": 0
                        }
                    }
                ]
            }
        }
    },
    {
        "name": "date",
```

B JSON-Schema Ereignis «Rechnung stellen»

```
        "designation": "Rechnungsdatum",
        "type": "date"
    }
],
"list": [
    {
        "name": "projectHours",
        "designation": "Projektstunden",
        "type": "array",
        "item": [
            {
                "name": "projectHours",
                "designation": "Projektstunden",
                "type": "object",
                "refClass": "projectHours",
                "properties": {
                    "unique": true
                }
            },
            {
                "name": "activity",
                "designation": "Tätigkeit",
                "type": "object",
                "refClass": "activity"
            },
            {
                "name": "description",
                "designation": "Beschreibung",
                "type": "string",
                "refItem": "activity",
                "properties": {
                    "value": {
                        "refItem": "$input.description"
                    },
                    "format": "textarea"
                }
            },
            {
                "name": "quantity",
                "designation": "Menge",
                "type": "number",
```

B JSON-Schema Ereignis «Rechnung stellen»

```
        "properties": {
            "decimals": 1,
            "unit": {
                "refItem": "unit"
            }
        }
    },
    {
        "name": "price",
        "designation": "Preis pro Stunde",
        "type": "number",
        "properties": {
            "decimals": 2,
            "currency": {},
            "value": {
                "refItem": "$input.price"
            }
        }
    }
]
},
{
    "name": "thirdPartyCost",
    "designation": "Fremdkosten",
    "type": "array",
    "item": [
        {
            "name": "thirdPartyCost",
            "designation": "Fremdkosten",
            "type": "object",
            "refClass": "thirdPartyCost",
            "properties": {
                "unique": true
            }
        }
    ],
    {
        "name": "description",
        "designation": "Beschreibung",
        "type": "string",
        "refItem": "thirdPartyCost",
        "properties": {
```

B JSON-Schema Ereignis «Rechnung stellen»

```
        "value": {
            "refItem": "$input.description"
        },
        "format": "textarea"
    }
},
{
    "name": "quantity",
    "designation": "Menge",
    "type": "number",
    "properties": {
        "decimals": 1,
        "unit": {
            "refItem": "unit"
        }
    }
},
{
    "name": "unit",
    "designation": "Einheit",
    "type": "object",
    "refClass": "unit",
    "properties": {
        "value": {
            "refItem": "$input.unit"
        }
    }
},
{
    "name": "price",
    "designation": "Preis",
    "type": "number",
    "properties": {
        "decimals": 2,
        "currency": {},
        "value": {
            "refItem": "$input.price"
        }
    }
}
]
```

B JSON-Schema Ereignis «Rechnung stellen»

```
    }
  ]
}
],
"result": {
  "resultDef": [
    {
      "template": "Rechnung",
      "service": "pdf",
      "transport": {
        "destinationItem": "invoice.customer.contact.email",
        "type": "email"
      }
    }
  ]
},
"state": {
  "direction": "Purpose",
  "task": "Compensation",
  "step": "Preparation"
},
"trigger": {
  "startOn": [
    {
      "type": "Manual"
    }
  ]
},
"validity": {}
}
```

C Evaluierungsdokumentation für die Benutzertests

Eval JUIBuilder

Vielen Dank für deine Teilnahme an der Evaluierung des JUIBuilder.

«JUIBuilder» ist eine Lösung, um die Umwandlung von JSON-Daten in HTML-Code zu automatisieren und zu vereinfachen.

Vorgehen Testing

«JUIBuilder» ist für das Businesssystem «enablerr» entwickelt worden, soll jedoch allgemein als Lösung für die Umwandlung von JSON in HTML anwendbar sein. Die hinterlegte elementConfigMap und das JSON-Schema, welche standardmässig von der Lösung verwendet werden, sind auf die enablerr-Struktur ausgelegt.

Vorgehen:

1. Github-Repository klonen und in eigener Entwicklungsumgebung öffnen.
2. README lesen.
3. HTML Viewer <https://html.onlineviewer.net/> öffnen.
4. Testfälle durchgehen.
5. Fragen beantworten.
6. Schlussbesprechung um 16:45 Uhr.

Testfälle

Die Lösung läuft in der Testing/Eval-Phase noch über Main.js. Dort sind sechs Beispiele und noch eine elementConfigMap und ein jsonSchema hinterlegt.

1. Testfall:
Generiere HTML-Code für enablerr mit dem enablerr JSON-Schema und der enablerr elementConfigMap (beides default hinterlegt.)
2. Testfall:
Generiere HTML-Code mit der elementConfigMap und dem JSON-Schema aus Main.js und teste die JSON-Daten exampleOne und exampleTwo.
3. Testfall:
Generiere HTML-Code mit einer leeren elementConfigMap und dem JSON-Schema aus Main.js.

Fragen

Wie bewertest du die Funktionalität der JSON-zu-HTML-Lösung insgesamt?

Gab es für dich Schwierigkeiten bei der Verwendung der Lösung?

Hat die Lösung die JSON-Daten deiner Meinung nach korrekt in den erwarteten HTML-Code umgewandelt?

Wurden alle notwendigen Daten aus dem JSON-Datensatz deiner Meinung nach korrekt im generierten HTML dargestellt?

Gab es Situationen, in denen der generierte HTML-Code nicht deinen Erwartungen entsprach?

Gab es für dich alle erforderlichen Optionen und Steuerelemente, um die Konvertierung anzupassen?

Gab es Funktionen oder Eigenschaften, die deiner Meinung nach fehlten oder verbessert werden könnten?

Wie reaktionsschnell war die Lösung beim Verarbeiten der JSON-Datensätze?

Gab es Leistungsprobleme oder Verzögerungen während des Umwandlungsprozesses?

Hat die Lösung dir hilfreiche Informationen zurückgegeben, wenn Fehler oder Probleme auftraten?

Welche Aspekte der Lösung haben dir am meisten gefallen? Welche Aspekte könnten deiner Meinung nach verbessert werden?