Abschlussprüfung Sommer 2024

Fachinformatiker für Anwendungsentwicklung

Dokumentation zur betrieblichen Projektarbeit

Contentful Validitäts-Plugin

Entwicklung eines Contentful Plugins zur Überprüfung der Validität von Einträgen von Contentfeldern

Abgabedatum: Neuburg an der Donau, den 14.05.2024

Prüfungsbewerber:

Kacper Patryk Sobczak

Amalienstraße 22 (1/3)

86633 Neuburg an der Donau

Ausbildungsbetreib:

LUSINI Service GmbH

Hettlinger Straße 9

86637 Wertigen

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Formelverzeichnis

Abkürzungsverzeichnis

CMS: Content Management System

SDK: Software Development Kit

API: Application Programming Interface

CMA: Content Management API

CDN: Content Delivery Network

UI: User Interface

UX: User Experience

CI: Continuous Integration

CD: Continuous Deployment

JSON: JavaScript Object Notation

HTML: Hypertext Markup Language

CSS: Cascading Style Sheets bzw. gestufte Stilvorlagen

Plugin: Software-Erweiterung oder Zusatzmodul

Deployment: Softwareverteilung

Content: Inhalt

SCRUM: Agiles Vorgehensmodel

Netilfy: Plattform für Web-Entwicklung und Hosting

Open-Source: Freier Zugang zum Quellcode eines externe Software Modules

E-Commerce: elektronischer Handel

Framework: Rahmenstruktur bzw. Programmiergerüst in Software-Kontext

TS: TypeScript

JS: JavaScript

React: JavaScript-Programmbibliothek

JSON: JavaScript Object Notation

Snippets/Schnipsel: Eine kurze Zusammenfassung des Inhalts einer Website e.g. Text

Dashboard: Überblick über relevanten Daten

Accordion: interaktives grafisches Benutzeroberflächenelement

Forma 36: Open-Source Gestaltungssystem von Contentful

VSC: Visual Studio Code, Entwicklungsumgebung

Jest: JavaScript testing framework

Code Review: Maßnahme zur Qualitätssicherung bei der Softwareentwicklung

1 Einleitung

Die folgende Projektdokumentation schildert den Ablauf des IHK-Abschlussprojektes, welcher der Autor im Rahmen ihrer Ausbildung zur Fachinformatiker Fachrichtung Anwendungsentwicklung durchgeführt hat. Ausbildungsbetrieb ist die LUSINI Service GmbH, ein Omnichannel-Anbieter für Hotellerie- und Gastronomiebedarf im Non-Food-Bereich in Europa mit Standort in Wertingen. Derzeit beschäftigt die Lusini ca. 700 Mitarbeiter.[1]

1.1 Projektbeschreibung

Das Projekt zielt darauf ab, eine Lösung für die wachsende Herausforderung zu entwickeln, die sich aus dem massiven Anstieg an selbst erstelltem Content sowie der Zunahme von Übersetzbaren Texten und notwendigen Inhalten ergibt, der für den Shop in Contentful gepflegt werden muss. Der derzeitige Arbeitsablauf ist geprägt von Schwierigkeiten beim Deployment-Prozess, da alle erforderlichen und kritischen Inhalte zwangsläufig übersetzt werden müssen, um einen fehlerfreien Shop in allen Sprachen sicherzustellen.

Diese Anforderung führt zu einem zunehmenden Aufwand für das Content-Team, da es schwierig ist, den Überblick darüber zu behalten, welche Inhalte bereits übersetzt wurden und welche nicht. Diese Unklarheit resultiert oft in fehlerhaften Deployments, die abgebrochen werden müssen. Die Identifizierung und Behebung dieser Probleme erfordert einen erheblichen Aufwand seitens der Entwickler, was langfristig nicht tragbar ist.

In diesem Projekt soll der gerade beschriebene Prozess verwendet werden, um nicht übersetzbare Texte automatisch einzusehen.

1.2 Projektziel

Das angestrebte Ergebnis besteht in der Entwicklung eines Plugins, das jedes einzelne Element, das in Contentful gepflegt wird, erfasst, und den Contentpflegern ermöglicht, fehlende Übersetzungen direkt einzusehen und darauf zuzugreifen. Dies entlastet die Contentpfleger und ermöglicht es den Entwicklern, sich auf wichtigere Themen zu konzentrieren.

[1] Kennzahl zum Stichtag 06.05.2024, Vgl. (Lusini, 2024)

1.3 Projektumfeld

Der Auftraggeber dieses Projekts ist die Fachabteilung für Contentmanagement bei LUSINI. Diese Abteilung ist verantwortlich für die Verwaltung und Pflege sämtlicher Inhalte, die auf der Online-Plattform präsentiert werden, einschließlich Produktbeschreibungen, Bilder und andere relevante Informationen. Sie müssen sicherstellen, dass alle Inhalte korrekt und aktuell sind, um ein reibungsloses Funktionieren des Online-Shops auf anderen Sprachen zu gewährleisten. Die Mitarbeiter dieser Abteilung stehen vor der Herausforderung, den Überblick über den massiven Content und die Übersetzungen zu behalten. Daher ist die Entwicklung eines Plugins, das den Prozess der Übersetzungsvalidierung unterstützt, von großer Bedeutung.

Um den Anforderungen der Abteilung gerecht zu werden, ist eine intensive und regelmäßige Kommunikation und Rücksprache mit der Fachabteilung erforderlich, sowie die technischen Hintergründe und die Dokumentation des Plugins von Contentful.[2]

1.4 Projektbegründung

Die steigende Komplexität und Menge an Inhalten auf der LUSINI-Plattform erfordert eine effiziente Lösung, um die Übersetzungsvalidierung zu gewährleisten. Das Contentmanagement-Team steht vor der Herausforderung, den Überblick über den wachsenden Content und die erforderlichen Übersetzungen zu behalten. Die derzeitigen Prozesse führen zu fehlerhaften Deployments und erfordern einen erheblichen Aufwand seitens der Entwickler. Die Entwicklung eines Plugins zur Überprüfung der Validität von Einträgen von Contentfeldern bietet eine Lösung, die die Arbeitsbelastung des Contentmanagement-Teams reduziert und die Effizienz des Deployment-Prozesses verbessert.

Durch das Plugin können Contentpfleger fehlende Übersetzungen direkt identifizieren und beheben, was zu einer höheren Genauigkeit und Zeitersparnis führt. Die direkte Integration in den Contentful-Arbeitsablauf erleichtert die Nutzung und Akzeptanz des Plugins. Darüber hinaus entlastet das Plugin die Entwickler, da sie sich nicht mehr mit Problemen aufgrund fehlender Übersetzungen befassen müssen, und ermöglicht es ihnen, sich auf wichtigere Aufgaben zu konzentrieren.

Die Entwicklung dieses Plugins ist daher von entscheidender Bedeutung, um die Effizienz und Genauigkeit des Contentmanagements bei LUSINI zu verbessern, die Kundenerfahrung zu optimieren und die Wettbewerbsfähigkeit des Unternehmens auf dem internationalen für Hotellerie- und Gastronomieausstattung zu stärken.

[2] Vgl. https://www.contentful.com/developers/docs/extensibility/app-framework/

1.5 Projektschnittstellen

Das Projekt umfasst verschiedene Schnittstellen, die eine nahtlose Integration gewährleisten sollen. Die LUSINI-Website dient als zentraler Anlaufpunkt für Kunden und präsentiert sämtliche Inhalte, einschließlich Produktinformationen und Dienstleistungen. Die Daten für diese Inhalte stammen aus Contentful, einem Headless-CMS, das es ermöglicht, Inhalte unabhängig von ihrer Präsentation zu verwalten und bereitzustellen.[3]

Die Interaktion mit Contentful Logik erfolgt über das Contentful App Framework, das benutzerdefinierte Apps für Contentful ermöglicht. Dadurch erhält das entwickelte Plugin direkten Zugriff auf die in Contentful gepflegten Daten, ohne zusätzliche Schnittstellen oder externe Dienste zu benötigen. Dies ermöglicht eine effiziente Kommunikation und Datenaustausch, um den Arbeitsablauf des Contentmanagements bei LUSINI zu optimieren.[4]

1.6 Projektabgrenzung

Da der Projektumfang beschränkt ist, soll das Bereitstellen von CI/CD Pipeline über GitHub Action und Netlify nicht Bestandteil des Abschlussprojektes sein.

2 Projektplanung

2.1 Projektphasen

Für die Umsetzung des Projektes standen der Autor 80 Stunden zur Verfügung. Diese wurden vor Projektbeginn auf verschiedene Phasen verteilt, die während der Softwareentwicklung durchlaufen werden. Eine grobe Zeitplanung sowie die Hauptphasen lassen sich der *Tabelle 1: Grobe Zeitplanung* entnehmen. Außerdem können die einzelnen Hauptphasen noch in kleinere Unterpunkte zerlegt werden. Eine detaillierte Übersicht dieser Phasen befindet sich im Anhang *Tabelle 4: Detaillierte Zeitplanung* auf Seite iv[5]

[3] Vgl. https://www.contentful.com/headless-cms/
[4] Vgl. https://www.contentful.com/blog/build-custom-apps-with-the-contentful-app-framework/
[5] Vgl. https://www.lucidchart.com/blog/de/4-lebenszyklus-projektmanagement-phasen

Projektphase	Geplante Zeit
Informationsbeschaffung	5 h
Planung	18 h
Durchführung	50 h
Kontrolle	7 h
Gesamt	**80 h**

Tabelle 1: Grobe Zeitplanung

2.2 Ressourcenplanung

Die Übersicht auf Seite *A.2 Verwendete Ressourcen* listet alle Ressourcen auf, die für das Projekt verwendet wurden. Hierbei sind sowohl Hardware- und Software-Ressourcen als auch das Personal inkludiert. Bei der Auswahl der Software wurde besonderes Augenmerk daraufgelegt, dass sie entweder kostenfrei verfügbar ist (z.B. als Open Source) oder dass LUSINI bereits Lizenzen dafür besitzt. Dieser Ansatz zielt darauf ab, die Projektkosten möglichst niedrig zu halten.

2.3 Entwicklungsprozess

Vor Beginn der Projektrealisierung stand die Entscheidung für einen passenden Entwicklungsprozess an. Dieser sollte festlegen, wie das Projekt umgesetzt wird. Für das Abschlussprojekt wurde ein agiler Ansatz gewählt, der sich an dem Scrum-Vorgehensmodell orientiert. Merkmale wie das iterative Durchlaufen der Phasen und regelmäßige Abstimmungen mit den Stakeholdern prägen diesen agilen Entwicklungsprozess.[6] Die kurzen Iterationszyklen ermöglichen eine flexible Umsetzung der Anforderungen, wodurch dem Fachbereich relativ zeitnah Ergebnisse präsentiert werden können. Aufgrund dieser Flexibilität wurde in der Projektplanung *2.1 Projektphasen* vergleichsweise wenig Zeit für die Planungsphase eingeplant, da sich Teile dieser Phase erst im Verlauf der Softwareentwicklung ergeben.[7][8]

In diesem Projekt wurde auf eine agile Entwicklung mit einem Ticketsystem gesetzt. Zuerst wurde ausführlich geplant und die Grundfunktionen entstanden. Im folgenden Schritt wurde mit Tickets gearbeitet, um die einzelnen Aufwände zu buchen und damit zu protokolieren. In der Projektplanung wurde bewusst viel Zeit für die Durchführungsphase

[6] Vgl. https://www.lucidchart.com/blog/de/phasen-des-lebenszykluses-einer-agilen-softwareentwicklung
[7] Vgl. https://digitaleneuordnung.de/blog/agiler-prozess/
[8] Vgl. https://de.smartsheet.com/best-practices-guide-agile-planning-project-managers

eingeplant, um das Prinzip der Agilen Entwicklung und das damit verbundene intensive Testen vollständig umsetzen zu können.

3 Analysephase

3.1 Ist-Analyse

Wie bereits im Abschnitt *1.1 Projektbeschreibung* erwähnt wurde, ist die Contentpflegeabteilung des E-Commerce-Teams für die Pflege, Übersetzung und Anpassung der Text zuständig. Die erforderlichen Daten werden derzeit ohne Validierung auf der Contentful-Plattform gepflegt. Es ist derzeit erforderlich, diese Texte zu übersetzen, da andernfalls bei einem wöchentlichen Webseiten-Release aufgrund fehlender Texte in der CI/CD-Pipeline das Deployment Prozess nicht abgeschlossen werden kann. In solchen Fällen müssen die Entwickler die von den Deployment-Services bereitgestellten Fehlerprotokolle analysieren und dem E-Commerce-Bereich die Stellen melden, die noch übersetzt werden müssen. Erst nachdem alle Text-Felder übersetzt sind und keine weiteren Fehler in der Deployment-Pipeline auftreten, kann ein neues Seiten-Deployment in Ländern erfolgen, in denen der Datensatz vollständig übersetzt ist, bei beispielsweise neuem Teaser oder Webkomponenten der Seite, die beim Aufrufen der Seite mit entsprechendem Sprachtext angezeigt werden in Abhängigkeit welches Sprach gewählte würde von dem Benutzer.

Der Zwang, alle erforderlichen Inhalte zu übersetzen, um einen fehlerfreien Shop in allen Sprachen zu gewährleisten, erschwert es dem Content-Team, den Überblick zu behalten. Dies führt zu fehlerhaften Shop-Deployments, die von den Entwicklern identifiziert werden müssen, was langfristig nicht tragbar ist.

Um eine visuelle Übersicht zu gewinnen, wurde auch ein Aktivitätsdiagramm zum IST-Zustand erstellt. Dieses ist im *Anhang Abbildung 1: Aktivitätsdiagramm des Ist-Zustandes* auf Seite vi zu finden.

3.2 Wirtschaftlichkeit

Angesichts der Probleme im aktuellen Übertragungswertverarbeitungsprozess, wie sie in den Abschnitten *1.4 Projektbegründung* und *3.1 Ist-Analyse* beschrieben und erläutert wurden, ist die Umsetzung des Projektes erforderlich. Es stellt sich jedoch die Frage, ob die Realisierung auch aus wirtschaftlicher Sicht gerechtfertigt ist. Diese Frage soll in den folgenden Abschnitten geklärt werden.

3.2.1 „Make or Buy“-Entscheidung

Da das Projekt sehr spezifische Anforderungen von LUSINI umfasst und verschiedene Komponenten wie Contentful und die Programmiersprachen TypeScript (TS) für die Browseranwendung miteinander interagieren müssen, und diese auch nur nach intern definierten Datenschemas bearbeitet werden müssen, besteht keine große Möglichkeit Outsourcing auf dem Markt zu finden. Zusätzlich ist es auch entscheidend, dass das Wissen über die Einführung der SDK-Möglichkeiten von Contentful intern bleibt, um auch zukünftige Anforderungen zu bewältigen und den Wissensaustausch innerhalb des Unternehmens zu fördern. Aus diesem Grund wird das Projekt intern entwickelt.

3.2.3 Projektkosten

Die Projektkosten während der Entwicklung werden im Folgenden kalkuliert. Neben den Personalkosten, die durch die Realisierung des Projekts entstehen, werden auch die Ausgaben für Ressourcen (Hard- und Software, Büroarbeitsplatz usw.) berücksichtigt. Die genauen Personalkosten können aus datenschutzrechtlichen Gründen nicht veröffentlicht werden, daher werden Stundensätze verwendet, die von der Personalabteilung festgelegt wurden. Der Stundensatz eines Auszubildenden beträgt 10 €, der eines Mitarbeiters 30 €.

Vorgang	Mitarbeiter	Zeit	Stundensatz	Gesamt
Entwicklungskosten	1 x Auszubildener	80 h	10 €	800,00 €
Fachgespräch	2 x Mitarbeiter	5 h	30 €	150,00 €
Code-Review	1 x Mitarbeiter	1 h	30 €	30,00 €
Abnahme	1 x Mitarbeiter	0,5 h	30 €	15,00 €
Projektkosten gesamt				**995,00 €**

Tabelle 2: Kostenaufstellung

3.2.3 Amortisationsdauer

In diesem Abschnitt wird untersucht, wann die Entwicklung der Anwendung sich amortisiert hat. Dieser Wert ist entscheidend für die wirtschaftliche Sinnhaftigkeit des Projekts und zeigt, ob langfristige Kostenvorteile entstehen. Die Amortisationsdauer wird durch die Division der Anschaffungskosten durch die laufenden Einsparungen berechnet, die durch das neue Produkt erzielt werden. In diesem Fall hat es im Durchschnitt den Entwicklern ca. 2 Stunden pro Woche gekostet, nicht übersetzbare Texte zu identifizieren und dem Contentpfleger zu melden. Die Übersetzung dieser fehlenden Texte durch den Contentpfleger hat im Durchschnitt 1 Stunde gedauert. Somit ergibt sich insgesamt eine Einsparung von 3 Stunden pro Woche. Bei einem Stundenlohn von 30 Euro bedeutet das eine Einsparung von 360 Euro pro Monat ergibt.

Die Amortisationsdauer wird durch die Division der Anschaffungskosten A durch die laufenden Einsparungen E berechnet.[9]

$$Amortisationsdauer = A/E \quad (1)$$

$$180\ \frac{min}{Woche} \times 4 \frac{Wochen}{Monat} = 720 \frac{min}{Monat} = 12 \frac{std}{Monat} \quad (2)$$

$$12 \frac{std}{Monat} \times 30\ € = 360 \frac{€}{Monat} \quad (3)$$

$$\frac{995{,}00€}{360 \frac{€}{Monat}} = 2{,}76\ Monate \approx 2\ Monate\ 3\ Wochen\ 2\ Tage \quad (4)$$

Formel 1: Berechnung der Amortisationsdauer

Die Amortisation wurde auch grafisch dargestellt, um die Berechnung der Amortisationsdauer visuell darzustellen. Das Diagramm enthält die Kosten (pro Jahr) sowohl der alten als auch der neuen Lösung. Zusätzlich wurden die Entwicklungskosten der neuen Lösung berücksichtigt. Die entsprechende Grafik befindet sich im Anhang *Abbildung 2: Graphische Darstellung der Amortisation* auf Seite vii.

Basierend auf der Amortisationsrechnung beträgt die Amortisationsdauer des Projekts 2 Monate, 3 Wochen und ca. 2 Tage. Dies ist der Zeitraum, über den die neue Anwendung mindestens genutzt werden muss, damit sich die Anschaffungskosten und die Kosteneinsparungen ausgleichen. Da das Unternehmen beabsichtigt, die neue Anwendung langfristig zu nutzen, lässt sich das Projekt auch aus wirtschaftlicher Sicht als sinnvoll einstufen.

3.3 Nicht-monetäre Vorteile

Da die Ergebnisse der Wirtschaftlichkeitsanalyse bereits die Realisierung des Projekts ausreichend rechtfertigen, wird an dieser Stelle auf eine detaillierte Analyse nicht-monetärer Vorteile verzichtet. Dennoch sollten nicht-monetäre Vorteile der neuen Anwendung erwähnt werden, wie etwa die verbesserte Übersichtlichkeit durch den Contentful-Plugin gewährgeleistet ist.

3.4 Anwendungsfälle

Im Verlauf der Analysephase wurde ein Use-Case-Diagramm erstellt, um eine grobe Übersicht über die Anwendungsfälle zu erhalten, die vom zu entwickelnden Programm abgedeckt werden sollen. Dieses Diagramm ist im Anhang *Abbildung 3: Use-Case-*

[9] Vgl. https://de.smartsheet.com/best-practices-guide-agile-planning-project-managers

Diagramm auf Seite vii zu finden und umfasst alle Funktionen, die aus Sicht der Endanwender benötigt werden.[10] Das Diagramm zeigt auch, dass die Entwickler die Fehlermeldung nicht mehr manuell an die E-Commerce-Abteilung übermitteln müssen, da dies automatisch durch das Plugin geschieht.

4 Durchführungsphase

4.1 Zielplattform

Die Auswahl der Zielplattform für das Projekt basierte auf mehreren Kriterien. TypeScript in Verbindung mit React wurde als die geeignete Programmiersprache gewählt, da diese Technologien als robust und effizient für die Entwicklung von Webanwendungen gelten und das Team bereits über langjährige Erfahrung und Vertrautheit damit verfügt. Die Entscheidung, auf andere Software oder Programmiersprachen zu verzichten, wurde aufgrund der bestehenden Vertrautheit mit React und Contentful getroffen. Da das Team bereits über Erfahrung und Know-how in der Entwicklung mit React verfügte und Contentful als Plattform für die Datenverwaltung bereits etabliert war, wurde darauf verzichtet, alternative Technologien zu evaluieren und eine Nutzwertanalyse nicht in Betracht gezogen. Dies ermöglichte eine reibungslose Integration des Projekts in die bestehende Infrastruktur und reduzierte den Schulungsbedarf für neue Technologien und Archtekturdesign als auch die zusätzliche Planungszeit, bei Erstellen der Software

4.2 Architekturdesign

Die gewählte Anwendungsarchitektur basiert auf einer clientseitigen Entwicklungsumgebung, die durch mehrere Merkmale unterstützt wird:

Clientseitige Anwendungsentwicklung: Die gesamte Logik, einschließlich Datenverarbeitung und Zustandsmanagement, wird im Frontend, also im Browser des Benutzers, ausgeführt. React und der useEffect-Hook werden verwendet, um die Datenabfrage und -verarbeitung durchzuführen.

Komponentenbasierte Architektur: Die Anwendung ist in wiederverwendbare Komponenten strukturiert, die jeweils spezifische Funktionen erfüllen. Beispiele hierfür sind *ConfigScreen*, *Dialog*, *EntryEditor*, *EntryField*, *EntrySidebar* und andere Komponenten, die im Code verwenden werden und auch von Contenful erwartet werden, in Anhang *Abbildung 6: Dialog App Locations* auf Seite x werde diese auch visuell aufgezeichnet.[11]

[10] Vgl. https://www.lucidchart.com/pages/uml-use-case-diagram
[11] Vgl. https://appmaster.io/blog/react-component-based-architecture

Asynchrone Datenverarbeitung: Die Anwendung behandelt asynchrone Operationen wie das Abrufen von Daten von einer externen Quelle (Contentful) und das Verarbeiten dieser Daten, um fehlende Texten für verschiedene Lokalisierungen zu identifizieren.[12]

Zustandsmanagement: Der Zustand der Anwendung wird mithilfe von React Hooks wie useState und useEffect verwaltet. Beispielsweise wird der Zustand der fehlenden Texte mit setAllMissingSnippets und setDataFetched aktualisiert.[13][14]

Im Hinblick auf Frameworks wird React als Hauptframework für die Benutzeroberfläche verwendet. React zeichnet sich durch Effizienz, eine klare Komponentenstruktur und die einfache Handhabung von Zuständen und Lifecycle-Methoden aus.[15] Die Verwendung von Contentful für das Datenmanagement zeigt eine effektive Integration eines externen Content-Management-Systems in die Anwendung. Contentful bietet eine flexible API und ermöglicht eine einfache Verwaltung von Inhalten, was es zur idealen Wahl für solche Anwendungen macht.[16]

Insgesamt ermöglicht die gewählte Architektur eine effiziente und wartbare Entwicklung von Anwendungen mit einer klaren Trennung von Anliegen und einer einfachen Integration externer Ressourcen wie Contentful.

4.3 Implementierung der Datenstrukturen

Die Datenstrukturen für die Anwendung wurden mithilfe von Contentful definiert, einem Content-Management-System, das die Verwaltung und Bereitstellung von Inhalten ermöglicht. Die Hauptdatenbank besteht aus verschiedenen Content-Typen, die die Struktur der Daten festlegen. Hierbei handelt es sich um JSON-Schemas, die die Struktur der Datensätze definieren. Dieses Schema ist in Anhand *Abbildung 4: Klassendiagramm* auf Seite viii zu sehen.[17]

Für das vorliegende Projekt wurden spezifische Content-Typen wie snippets erstellt, um die benötigten Daten zu speichern. Diese Content-Typen enthalten Felder, die verschiedene Informationen speichern, wie z. B. den Namen des Schnipsels, die ID des Schnipsels und andere relevante Daten. Durch die Definition dieser Content-Typen wird die Struktur der Datenbank festgelegt, was eine einheitliche und geordnete Speicherung der Informationen ermöglicht. Die Implementierung der Datenstrukturen wurde in Form von JSON-Schemas durchgeführt, die in Contentful definiert und verwaltet werden. Diese

[12] Vgl. https://contentful.github.io/contentful.net-docs/articles/async-programming.html
[13] Vgl. https://medium.com/@dev.umigo/how-to-contentful-and-react-7a2bb88fccf8
[14] Vgl. https://blog.logrocket.com/modern-guide-react-state-patterns/
[15] Vgl. https://www.codecademy.com/learn/react-101/modules/react-102-lifecycle-methods-u/cheatsheet
[16] Vgl. https://www.contentful.com/developers/docs/references/content-delivery-api/
[17] Vgl. https://www.contentful.com/seo-guide/schema-seo/

Schemas stellen sicher, dass die Daten konsistent und gemäß den Anforderungen der Anwendung strukturiert sind.

Die genaue Beschreibung der angelegten Datenbank und der einzelnen Content-Typen sowie deren Felder und Beziehungen kann im Contentful-Dashboard eingesehen werden. Dort sind die JSON-Schemas für jeden Content-Typ detailliert dokumentiert und können bei Bedarf angepasst werden, um die Anforderungen der Anwendung zu erfüllen. Die Auszüge sind zu finden in Anhang *Abbildung 5:* auf die Seite ix

4.4 Implementierung der Geschäftslogik

Die Implementierung der Geschäftslogik erfolgte in mehreren Schritten gemäß dem Entwurf der Anwendung. Zunächst wurden die Anforderungen und Funktionen der Anwendung analysiert, um die erforderlichen Algorithmen und Datenstrukturen zu identifizieren. Anschließend wurde das Vorgehen bei der Umsetzung geplant und die Programmierung begonnen. Bei der Umsetzung wurden verschiedene Entwurfsmuster und bewährte Praktiken verwendet, um eine saubere und strukturierte Codebasis zu gewährleisten. Die Funktionen und Algorithmen, die in der Anwendung implementiert wurden, umfassen:

1. Datenabfrage und -verarbeitung: Die Anwendung verwendet Contentful, um Daten abzurufen und zu verarbeiten. Dazu wurden Funktionen wie fetchData() implementiert, die asynchron Daten von Contentful abrufen und verarbeiten.

2. Fehlerbehandlung: Um mit fehlenden Schnipseln umzugehen, wurden entsprechende Logiken implementiert, die fehlende Schnipsel identifizieren und entsprechend verarbeiten, die detaillierte Beschreibung der Funktionsweise von Quellencode, kann wie folgt beschreiben werden: Die Komponente Dialog in der Anwendung wurde implementiert, um mit fehlenden Schnipseln umzugehen. Beim Laden der Daten werden alle Schnipsel von Contentful abgerufen und nach fehlenden Übersetzungen in den verschiedenen Lokalisierungen durchsucht. Wenn ein fehlender Schnipsel gefunden wird, wird er zur Liste der fehlenden Schnipsel allMissingSnippets hinzugefügt, die anschließend dem Benutzer angezeigt wird in Benutzeroberfläche. Auf Benutzeroberfläche, wird detailliert in Abschnitt *4.5 Implementierung der Benutzeroberfläche* eingegangen. Die Implementierung verwendet verschiedene Mechanismen, um sicherzustellen, dass fehlende Schnipsel bzw. Snippets effizient identifiziert und verarbeitet werden. Dazu gehört die Verwendung von React Hooks wie useState und useEffect zur Verwaltung des Anwendungszustands sowie die Verwendung der Accordion-Komponente von Forma 36 zur Darstellung der fehlenden Schnipsel bzw. Snippets in einer übersichtlichen Weise, die in Benutzeroberfläche zu sehen ist in den

Anhang *Abbildung 20: Benutzeroberfläche von MissingSnippets (geöffnet)* auf Seite xxvii.[18]

3. Zustandsmanagement: Die Anwendung verwendet React Hooks wie useState und useEffect, um den Zustand der Anwendung (e.g. allMissingSnippets) zu verwalten und auf Änderungen zu reagieren. Ein Beispiel für Quelltextausschnitte, der die Implementierung der Geschäftslogik zeigt, ist die fetchData()-Funktion, die in React.useEffect verwendet wird, um Daten von Contentful abzurufen und zu verarbeiten, diese ist zu finden in Anhand *Abbildung 12: Dialog.tsx Komponente (Quellencode)* auf Seite xvi. Diese Funktion demonstriert das Abrufen von Daten von Contentful und deren anschließende Verarbeitung. Durch die Verwendung von asynchronen Funktionen wird sichergestellt, dass die Anwendung reaktiv bleibt und nicht blockiert wird. Ein ausführlicher Code ist in Anhand Kapitel *A.9 Quellencode* auf Seite xv bis xxiv zu sehen, um detaillierte Arbeitsweise der einzelnen Komponente zu entnehmen.

Die Implementierung der Geschäftslogik wurde sorgfältig dokumentiert und kommentiert, um ihre Funktionsweise und ihren Zweck klar zu erklären. Dies stellt sicher, dass Entwickler, die den Code lesen oder warten müssen, ihn leicht verstehen und weiterentwickeln können. Zusätzlich wurden mehrere Verweise auf die offizielle Dokumentation von React und des Contentful App Frameworks in den Kommentaren des Codes eingefügt. Diese Verweise, wie sie beispielsweise im Anhang *Abbildung 22: Auszug aus README.md für Entwickler* zu sehen sind, ermöglichen es Entwicklern, bei Bedarf weiterführende Informationen zu bestimmten Funktionen oder Konzepten zu konsultieren, was zur Verbesserung der Codeverständlichkeit und -qualität beiträgt.

4.5 Implementierung der Benutzeroberfläche

Die Implementierung der Benutzeroberfläche wurde mithilfe des Forma 36-Designsystems umgesetzt, das eine Open-Source-Bibliothek von React-Komponenten bereitstellt. Forma 36 wurde entwickelt, um den Entwicklungsprozess zu beschleunigen und eine konsistente Benutzererfahrung zu gewährleisten. Die UI-Bibliothek enthält eine Vielzahl von vorgefertigten Komponenten, die nahtlos in die Contentful-Umgebung integriert werden können.[19]

Das Corporate Design wurde gemäß den Vorgaben des Forma 36-Designsystems umgesetzt, um eine einheitliche visuelle Identität zu gewährleisten. Dies umfasst unter

[18] Vgl. https://f36.contentful.com/components/accordion
[19] Vgl. https://www.contentful.com/developers/docs/extensibility/app-framework/forma-36/

anderem die Verwendung von definierten Farbpaletten, Typografie und Layout-Strukturen.[20]

Die Benutzeroberfläche der Anwendung ist intuitiv gestaltet und bietet eine benutzerfreundliche Navigation. Screenshots der Anwendung werden im Anhang Kapitel *A.10 Benutzeroberfläche (UI)* bereitgestellt, um einen visuellen Einblick in das Erscheinungsbild der Benutzeroberfläche zu geben. Diese Screenshots zeigen verschiedene Ansichten der Anwendung, einschließlich der verwendeten CSS-Stile und UI-Komponenten aus dem Forma 36-Designsystem.

4.6 Maßnahmen zur Qualitätssicherung

Im Abschnitt *2.3 Entwicklungsprozess* wurde bereits betont, dass ein agiles Vorgehen angestrebt wird. Durch die agile Entwicklungsmethodik werden Teams befähigt, sich flexibel an Änderungen anzupassen und kontinuierlich auf Internen Feedback zu reagieren. Das Prinzip der iterativen Entwicklung ermöglicht es, frühzeitig Prototypen zu erstellen und sie mit eigentlicher Anwendung zu validieren. Durch diese iterative Vorgehensweise können fachliche Fehler von Anfang an vermieden werden, da die Anforderungen und Lösungsansätze kontinuierlich überprüft und angepasst werden.[21]

Des Weiteren können während des agilen Entwicklungsprozesses regelmäßige Reviews und Demos durchgeführt werden bei Bedarf, um die Funktionalität zu kontrollieren und sicherzustellen, dass das Produkt den Erwartungen entspricht. Diese kontinuierliche Überprüfung und Anpassung des Produkts trägt dazu bei, dass es den Anforderungen und Bedürfnissen der Benutzer gerecht wird. [22] Zur Qualitätssicherung werden mehrere Maßnahmen ergriffen:

Automatisierte Unit-Tests: Automatisierte Unit-Tests werden mithilfe der Jest-Erweiterung in Visual Studio Code (VSC) durchgeführt. Diese Tests zielen darauf ab, sicherzustellen, dass einzelne Teile der Anwendung korrekt funktionieren. Dabei wird jede Komponente isoliert getestet, um sicherzustellen, dass sie die erwarteten Ausgaben liefert und die erwarteten Zustände korrekt handhabt. Jest bietet eine benutzerfreundliche Syntax für das Schreiben von Tests und ermöglicht eine effiziente Ausführung und Überwachung der Testergebnisse direkt in der Entwicklungsumgebung. Die Verwendung von Jest in VSC erleichtert die Entwicklung und Wartung von Unit-Tests, was zu einer höheren Codequalität und Robustheit der Anwendung beiträgt.[23][24] Diese sind zu finden

[20] Vgl. https://f36.contentful.com/introduction/getting-started
[21] Vgl. https://www.accelq.com/blog/quality-assurance-in-agile-methodology/
[22] Vgl. https://www.accelq.com/blog/quality-assurance-in-agile-methodology/
[23] Vgl. https://www.computerweekly.com/de/definition/Unit-Test
[24] Vgl. https://www.computerix.info/skripten/unit-tests.pdf

in Anhang *Abbildung 15: Unittest von Dialog.tsx (Quellencode)* auf die Seite xx und *Abbildung 17: Unittest von ConfigScreen.tsx (Quellencode)* auf die Seite xxiv

Typsicherheit durch TypeScript: Die gesamte Anwendung wird mit TypeScript geschrieben, um eine höhere Typsicherheit zu gewährleisten. Dies ermöglicht eine frühzeitige Erkennung von Fehlern während der Entwicklung und verbessert die Wartbarkeit der Anwendung.

Manuelle Anwendertests: Neben den automatisierten Unit Tests werden auch manuelle Anwendertests durchgeführt, um sicherzustellen, dass die Benutzeroberfläche benutzerfreundlich ist und alle Funktionen wie erwartet funktionieren. Dabei werden verschiedene Szenarien durchgespielt, um sicherzustellen, dass die Anwendung unter realen Bedingungen einwandfrei funktioniert. Durch diese Maßnahmen wird sichergestellt, dass das Projektergebnis von hoher Qualität ist, zuverlässig funktioniert und den Anforderungen der Benutzer entspricht.[25]

5 Kontrolle und Abnahme

5.1 Abnahme durch Fachbereich

Nach Fertigstellung der gesamten Anwendung wurde sie dem Fachbereich zur Endabnahme präsentiert. Dank der agilen Softwareentwicklungsmethode erhielten die Fachbereiche nach jeder Iteration eine Vorführung der aktuellen Anwendungsversion. Dadurch waren sie bei der Endabnahme bereits mit der Benutzeroberfläche und dem Funktionsumfang der Anwendung vertraut. Anregungen und Kritik der Fachbereiche konnten durch die kontinuierlichen Rücksprachen frühzeitig während der Entwicklungsphase berücksichtigt werden. Dies führte dazu, dass es bei der Endabnahme keine Probleme oder Hindernisse mehr gab, und der Einführung der Anwendung nichts mehr im Wege stand. Vor dem Deployment wurde zusätzlich zur Abnahme durch den Fachbereich ein Code-Review von einem anderen Entwickler durchgeführt, um die Qualität der Anwendung sicherzustellen.[26][27]

5.2 Deployment und Einführung

Um den Zugriff auf die Anwendung für die Fachbereiche möglichst einfach zu gestalten, wurde die Anwendung zentral über das Contentful Framework für Applikations-Plugins integriert. Dadurch war sie auch über die Benutzeroberfläche von Contentful zugänglich,

[25] Vgl. https://www.zaptest.com/de/was-ist-das-testen-von-ui-software-tiefes-eintauchen-in-die-typen-verfahren-werkzeuge-und-umsetzung
[26] Vgl. https://www.nan-labs.com/blog/code-review-agile/
[27] Vgl. https://smartbear.com/learn/code-review/agile-code-review-process/

wo ein Button für jeden Content-Editor bereitstand, um die Anwendung problemlos zu nutzen. Die Integration erfolgte automatisch über die vorhandene Erweiterungsfunktion in den Einstellungen von Contentful, was die Implementierung erleichterte.[28] [29]

Wie bereits im vorherigen Abschnitt erwähnt wurde, waren die Fachbereiche durch die kontinuierliche Rückmeldung bereits mit der Anwendung vertraut. Daher waren zusätzliche Benutzerschulungen nicht erforderlich.

6 Dokumentation

Die Übertragungswertverwaltungsdokumentation besteht aus zwei Teilen: Projektdokumentation beschreibt die Umsetzungsphasen, und die Entwicklerdokumentation. Ein Benutzerhandbuch wurde aufgrund der intuitiven Benutzeroberfläche nicht erstellt.

Bei der Entwicklerdokumentation handelt es sich um eine übliche README-Datei im Markdown-Format. Im Grunde enthält die README-Datei eine allgemeine Beschreibung des Projekts sowie Anleitungen zur Installation, Konfiguration und Verwendung der Anwendung. Sie bietet eine Übersicht über verfügbare Skripte in der Befehlszeile und deren Funktionen, wie zum Beispiel das Starten der Anwendung im Entwicklungsmodus, das Erstellen eines Produktionsbuilds auf lokalen Rechner und das Hochladen der Anwendung in Contentful. Ein Ausschnitt aus der Entwicklerdokumentation befindet sich im Anhang *Abbildung 22: Auszug aus README.md für Entwickler* auf S. xxiv.[30]

7 Fazit

7.1 Soll- /Ist-Vergleich

In einer retrospektiven Analyse des IHK-Abschlussprojekts lässt sich feststellen, dass sämtliche vorab definierten Anforderungen gemäß dem Pflichtenheft erfolgreich erfüllt wurden. Der initial erstellte Projektplan im Abschnitt *2.1 Projektphasen* wurde präzise eingehalten. In der *Tabelle 3: Soll-/Ist-Vergleich*, welche einen Soll-/Ist-Vergleich präsentiert, wird die tatsächlich benötigte Zeit für die verschiedenen Phasen mit der ursprünglich geplanten Zeit gegenübergestellt. Hier zeigt sich lediglich eine marginale Abweichung von der Zeitplanung. Diese Unterschiede konnten miteinander ausgeglichen

[28] Vgl. https://www.contentful.com/developers/docs/extensibility/app-framework/create-contentful-app/
[29] Vgl. https://www.contentful.com/developers/docs/extensibility/app-framework/sdk/
[30] Vgl. https://www.ibm.com/docs/en/SSYKAV?topic=train-how-do-use-markdown

werden, wodurch das Projekt erfolgreich innerhalb des von der IHK vorgegebenen Zeitrahmens von 80 Stunden realisiert werden konnte.[31]

Projektphase	Soll	Ist	Differenz
Informationen Anschaffung	5 h	6 h	+ 1 h
Planung	18 h	19 h	+ 1 h
Durchführung	50 h	50 h	0 h
Kontroller	7 h	5 h	- 2 h
Gesamt	**80 h**	**80 h**	**0 h**

Tabelle 3: Soll-/Ist-Vergleich

7.2 Autors Erfahrung

Im Verlauf dieses Projekts konnte der Autor wertvolle Erkenntnisse über die Planung und Durchführung von Projekten sammeln. Insbesondere wurde deutlich, wie entscheidend kontinuierliche Kommunikation und Abstimmungen mit den Fachbereichen für eine erfolgreiche Projektdurchführung sind. Darüber hinaus wurden neue Einsichten in Bezug auf die Integration und Nutzung von Frameworks gewonnen. Abschließend lässt sich festhalten, dass die Umsetzung des Projekts nicht nur einen Mehrwert für die beteiligten Fachbereiche bietet, sondern auch für die Autor persönlich eine bedeutende Bereicherung darstellt.

7.3 Ausblick auf die Zukunft

Obwohl sämtliche festgelegten Anforderungen erfolgreich umgesetzt wurden, besteht dennoch Raum für die Definition neuer Anforderungen oder die Entwicklung von Erweiterungsvorschlägen in Zukunft. Ein Beispiel hierfür ist die Anfrage aus dem E-Commerce-Abteilung, ob es möglich wäre, nicht übersetzbare Texte mithilfe generativer Künstlicher Intelligenz wie beispielsweise ChatGPT zu übersetzen, um die fehlenden Felder automatisch zu ergänzen. Des Weiteren wird erwogen, die Anwendung künftig auch im Open-Source Bibliothek einzusetzen.

Dank der robusten Architektur Struktur, wie sie im Abschnitt *4.2 Architekturdesign* beschrieben ist, können solche Anpassungen oder Erweiterungen durchgeführt werden. Die Modularität der Anwendung ermöglicht somit eine effiziente Wartung und Erweiterung des Plugins.

[31] Vgl. https://www.business-wissen.de/artikel/soll-ist-vergleich-erstellen-formeln-tools-zusammenhaenge/

Literatur-/Quellenverzeichnis

AppMaster. (2023, 07 31). *React's Component-Based Architecture: A Case Study* . Retrieved from React's Component-Based Architecture: A Case Study : https://appmaster.io/blog/react-component-based-architecture

Chernyak, A. Z. (2024, 05 10). *Was ist das Testen von UI-Software? Tiefes Eintauchen in die Typen, Verfahren, Werkzeuge und Umsetzung*. Retrieved from Was ist das Testen von UI-Software? Tiefes Eintauchen in die Typen, Verfahren, Werkzeuge und Umsetzung: https://www.zaptest.com/de/was-ist-das-testen-von-ui-software-tiefes-eintauchen-in-die-typen-verfahren-werkzeuge-und-umsetzung

Codecademy. (2024, 05 10). *Lifecycle Methods*. Retrieved from Lifecycle Methods: https://www.codecademy.com/learn/react-101/modules/react-102-lifecycle-methods-u/cheatsheet

Contenful. (2016, 12 31). *Async programming*. Retrieved from Async programming: https://contentful.github.io/contentful.net-docs/articles/async-programming.html

Contenful. (2024, 05 10). *Build custom apps with the Contentful App Framework*. Retrieved from Build custom apps with the Contentful App Framework: https://www.contentful.com/blog/build-custom-apps-with-the-contentful-app-framework/

Contenful. (2024, 05 10). *Contenful Headless CMS explained in 1 minute*. Retrieved from Contenful Headless CMS explained in 1 minute: https://www.contentful.com/headless-cms/

Contenful. (2024, 05 10). *Content Delivery API*. Retrieved from Content Delivery API: https://www.contentful.com/developers/docs/references/content-delivery-api/

Contenful. (2024, 05 10). *Contentful App Framework*. Retrieved from Contentful App Framework: https://www.contentful.com/developers/docs/extensibility/app-framework/

Contenful. (2024, 05 10). *Create Contentful App*. Retrieved from Create Contentful App: https://www.contentful.com/developers/docs/extensibility/app-framework/create-contentful-app/

Contenful. (2024, 05 10). *Forma 36*. Retrieved from Forma 36: https://www.contentful.com/developers/docs/extensibility/app-framework/forma-36/

Contentful. (2024, 05 10). *App SDK Reference*. Retrieved from App SDK Reference: https://www.contentful.com/developers/docs/extensibility/app-framework/sdk/

contentful. (2024, 05 10). *Getting started*. Retrieved from Getting started: https://f36.contentful.com/introduction/getting-started

Diehl, A. (2024, 05 10). *Was ist ein agiler Prozess?* Retrieved from Was ist ein agiler Prozess?: https://digitaleneuordnung.de/blog/agiler-prozess/

Duran, M. E. (2022, 08 23). *Why You Should Definitely Be Adding Code Review to Your Agile Processes*. Retrieved from Why You Should Definitely Be Adding Code Review to Your Agile Processes: https://www.nan-labs.com/blog/code-review-agile/

Feldhaus, G. (2014, 12 03). *it-berufe-podcast.de.* Retrieved from it-berufe-podcast.de: https://s3.eu-central-1.amazonaws.com/fiae/Beispiele/ProjektdokumentationFachinformatikerAnwendungsentwicklung2015GerdaFeldhaus.pdf

Forma 36. (2024, 05 10). *Accordion*. Retrieved from Accordion: https://f36.contentful.com/components/accordion

IBM Watson Career Coach Trial. (2021, 03 05). *How do I use Markdown?* Retrieved from How do I use Markdown?: https://www.ibm.com/docs/en/SSYKAV?topic=train-how-do-use-markdown

Joshua Lohr, R. R. (2023, 12 06). *Schema SEO & structured data*. Retrieved from Schema SEO & structured data: https://www.contentful.com/seo-guide/schema-seo/

Kokane, K. (2023, 07 31). *The modern guide to React state patterns*. Retrieved from The modern guide to React state patterns: https://blog.logrocket.com/modern-guide-react-state-patterns/

Kusche, K. (2012). *Qualitätssicherung von Software am Beispiel von Unit Testing.* Keine Angabe Möglich: Keine Angabe Möglich. Retrieved from Qualitätssicherung von Software am Beispiel von Unit Testing.

Loncarevic, S. (2024, 01 01). *Soll-Ist-Vergleich erstellen (mit Excel)*. Retrieved from Soll-Ist-Vergleich erstellen (mit Excel): https://www.business-wissen.de/artikel/soll-ist-vergleich-erstellen-formeln-tools-zusammenhaenge/

Lucidchart. (2024, 05 10). *Die 4 Phasen des Projektlebenszykluses*. Retrieved from Die 4 Phasen des Projektlebenszykluses: https://www.lucidchart.com/blog/de/4-lebenszyklus-projektmanagement-phasen

Lucidchart. (2024, 05 10). *Die Phasen des Lebenszykluses einer agilen Softwareentwicklung*. Retrieved from Die Phasen des Lebenszykluses einer agilen Softwareentwicklung: https://www.lucidchart.com/blog/de/phasen-des-lebenszykluses-einer-agilen-softwareentwicklung

Lucidchart. (2024, 05 10). *UML Use Case Diagram Tutorial*. Retrieved from UML Use Case Diagram Tutorial: https://www.lucidchart.com/pages/uml-use-case-diagram

Lusini. (2024, 05 06). *Lusini Corporate*. Retrieved from Lusnin Corporate: https://www.lusini-corporate.com/de/

Macke, S. (2020, 01 01). *https://dieperfekteprojektdokumentation.de/*. Retrieved from https://dieperfekteprojektdokumentation.de/: https://dieperfekteprojektdokumentation.de/MicrosoftWord_Vorlage_Projektdokumentation_ITBerufe.pdf

Modu learn. (2024, 05 10). *Wie funktioniert Amortisationsrechnung?* Retrieved from Wie funktioniert Amortisationsrechnung?: https://www.modu-learn.de/verstehen/investition-finanzierung/amortisationsrechnung/

Raghammudi, V. R. (2024, 03 13). *Quality Assurance In Agile Methodology*. Retrieved from Quality Assurance In Agile Methodology: https://www.accelq.com/blog/quality-assurance-in-agile-methodology/

Smartbear. (2024, 05 10). *Code Review: An Agile Process* . Retrieved from Code Review: An Agile Process : https://smartbear.com/learn/code-review/agile-code-review-process/

Smartsheet. (2023, 12 31). *Ein Best Practices-Leitfaden für agile Planung für Projektmanager*. Retrieved from Ein Best Practices-Leitfaden für agile Planung für Projektmanager: https://de.smartsheet.com/best-practices-guide-agile-planning-project-managers

Umigo. (2019, 06 10). *HOW TO: Contentful and React*. Retrieved from HOW TO: Contentful and React: https://medium.com/@dev.umigo/how-to-contentful-and-react-7a2bb88fccf8

A Anhang

A.1 Detaillierte Zeitplanung

Informationsbeschaffung Phase		**5 h**
IST-Analyse durchführen (Fachgespräch, UML-Diagramme, etc.)	4 h	
Wirtschaftlichkeitsanalyse durchführen	1 h	
Planungsphase		**18 h**
Analyse bestehende Integration	2,5 h	
Interne Abstimmung	1,5 h	
Einlese in Contentful SDK Dokumentation	7 h	
Einarbeitung in Contentful SDK Framework	7 h	
Durchführungsphase		**50 h**
Anlage GitHub Repository / Branch	1 h	
Konfiguration Contentful Plugins	2 h	
Implementierung Contentful Plugins SDK	30 h	
Projektdokumentation schreiben	17 h	
Kontrolle Phase		**7h**
Funktionstest	4 h	
Unittest schreiben	2 h	
Manuels testen	1 h	
Gesamt		**80 h**

Tabelle 4: Detaillierte Zeitplanung

A.2 Verwendete Ressourcen

Hardware

- Unternehmens Laptop für Homeoffice

Software

- Linux Debian 20.04 - Betriebssystem
- Visual Studio Code – Entwicklungsumgebung TypeScript
- Contentful – App Framework
- Contentful – Content Management System
- Git – Verteilte Versionsverwaltung
- Jest - Framework zur Durchführung von Unit-Tests
- React – JavaScript/TypeScript Framework
- Jupyter Notebook – Diagrammdarstellung Tool
- PlantUML – Programm zum Erstellen von Diagrammen

Personal

- Mitarbeiter des E-Commerce Abteilung
- Entwickler/Auszubildener – Umsetzung des Projektes
- Anwendungsentwickler – Review des Codes und Fachgespräch

A.3 Aktivitätsdiagramm des Ist-Zustandes (UML)

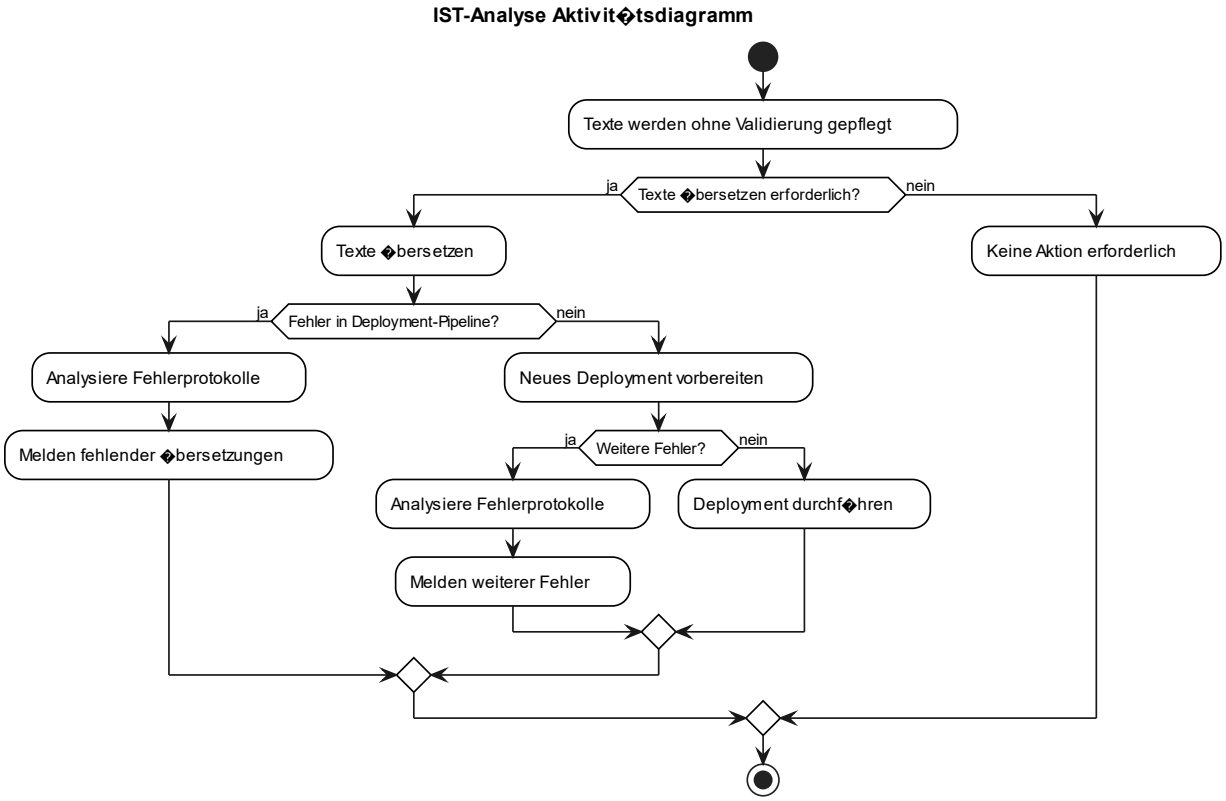


Abbildung 1: Aktivitätsdiagramm des Ist-Zustandes

A.4 Amortisation

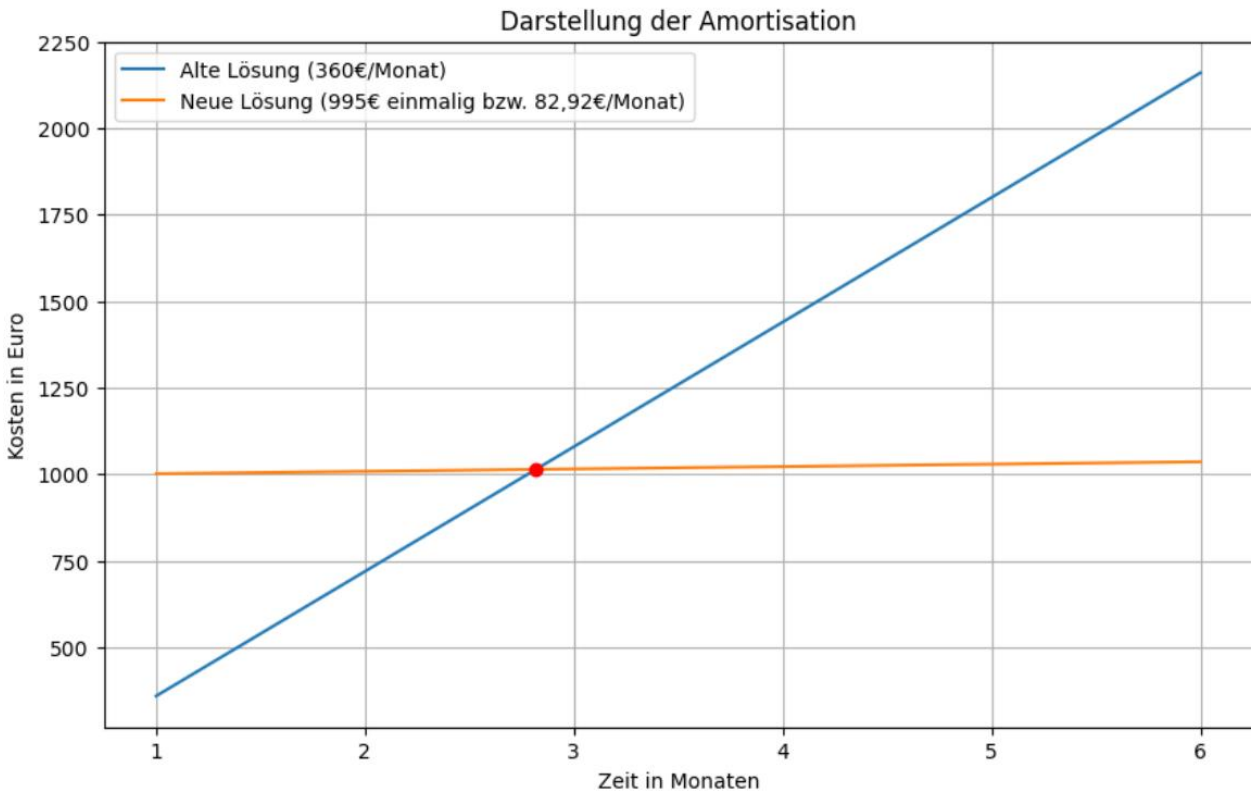


Abbildung 2: Graphische Darstellung der Amortisation

A.5 Use-Case-Diagramm

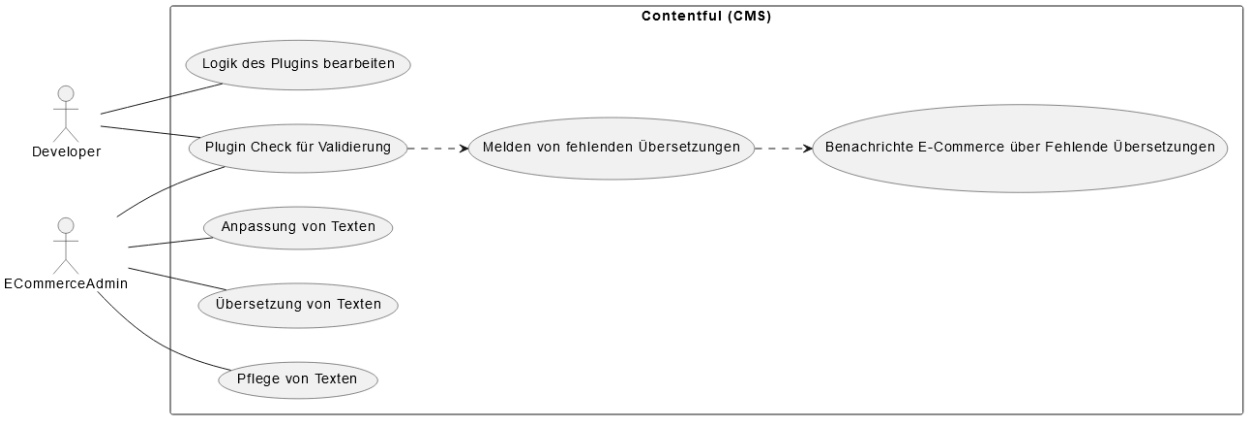


Abbildung 3: Use-Case-Diagramm

A.6 Klassendiagramm

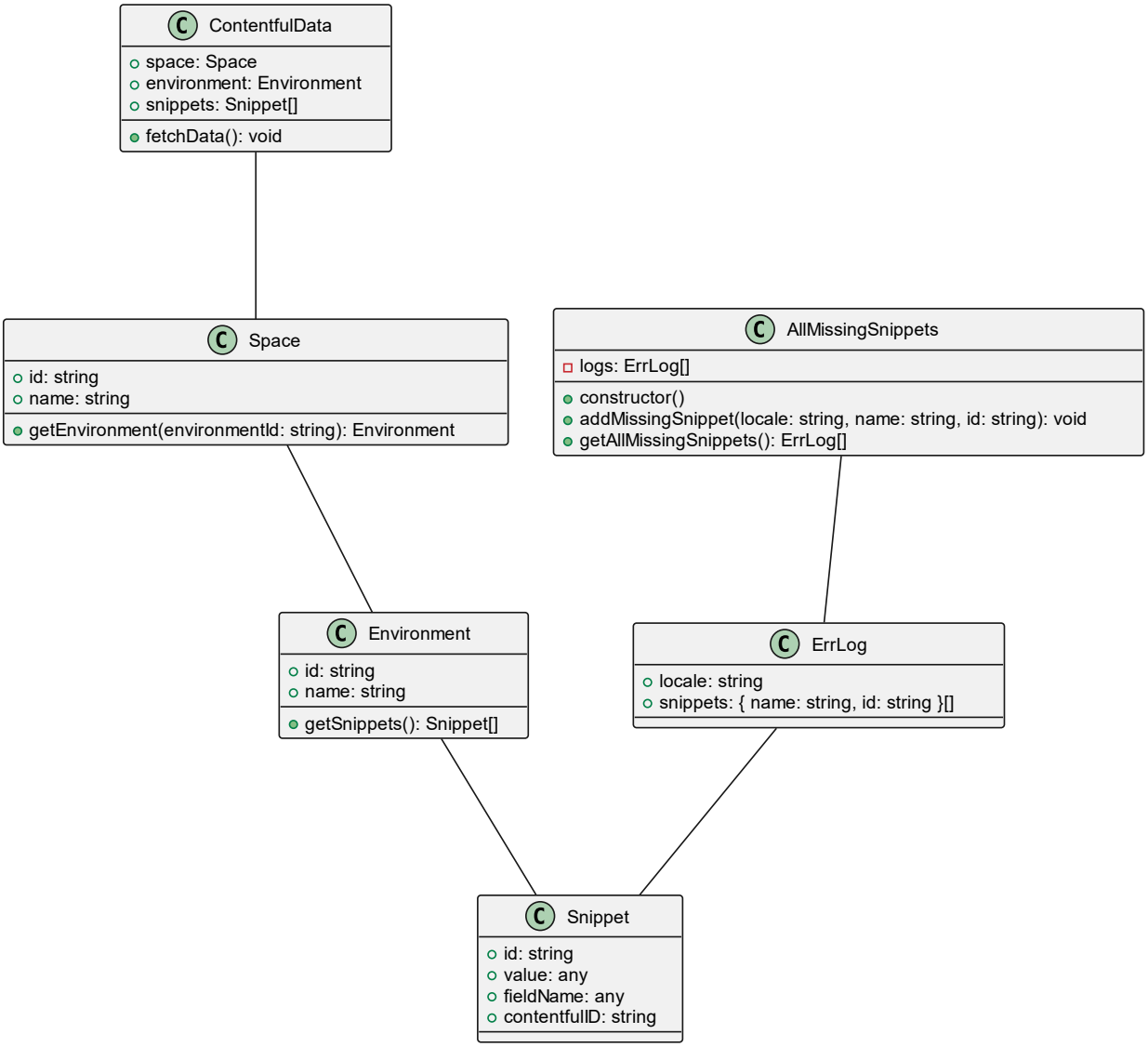


Abbildung 4: Klassendiagramm

A.7 JSON-Preview von Snippets

JSON Preview


```
{
  "name": "Snippets",
  "description": "",
  "displayField": "fieldName",
  "fields": [
    {
      "id": "fieldName",
      "name": "Field Name",
      "type": "Symbol",
      "localized": false,
      "required": true,
      "validations": [],
      "disabled": false,
      "omitted": false
    },
    {
      "id": "value",
      "name": "value",
      "type": "Text",
      "localized": true,
      "required": true,
      "validations": [],
      "disabled": false,
      "omitted": false
    },
    {
      "id": "context",
      "name": "Context",
      "type": "Text",
      "localized": true,
      "required": false,
      "validations": [],
      "disabled": false,
      "omitted": false
    },
    {
      "id": "description",
      "name": "Description",
      "type": "Text",
      "localized": false,
      "required": false,
      "validations": [],
      "disabled": false,
      "omitted": false
    },
    {
      "id": "deprecated",
      "name": "deprecated",
      "type": "Boolean",
      "localized": false,
      "required": false,
      "validations": [],
      "disabled": false,
      "omitted": false
    }
  ],
  "sys": {
    "space": {
      "sys": {
        "type": "Link",
        "linkType": "Space",
        "id": "aza65graowyr"
      }
    },
    "id": "snippets",
    "type": "ContentType",
    "createdAt": "2021-04-12T10:08:01.221Z",
    "updatedAt": "2023-02-08T15:32:23.350Z",
    "environment": {
      "sys": {
        "id": "master",
        "type": "Link",
        "linkType": "Environment"
      }
    },
    "publishedVersion": 41,
    "publishedAt": "2023-02-08T15:32:23.350Z",
    "firstPublishedAt": "2021-04-12T10:08:01.671Z",
    "createdBy": {
      "sys": {
```


Abbildung 5: JSON-Preview

A.8 Contentful App Locations

A.8.1 Dialog App Locations

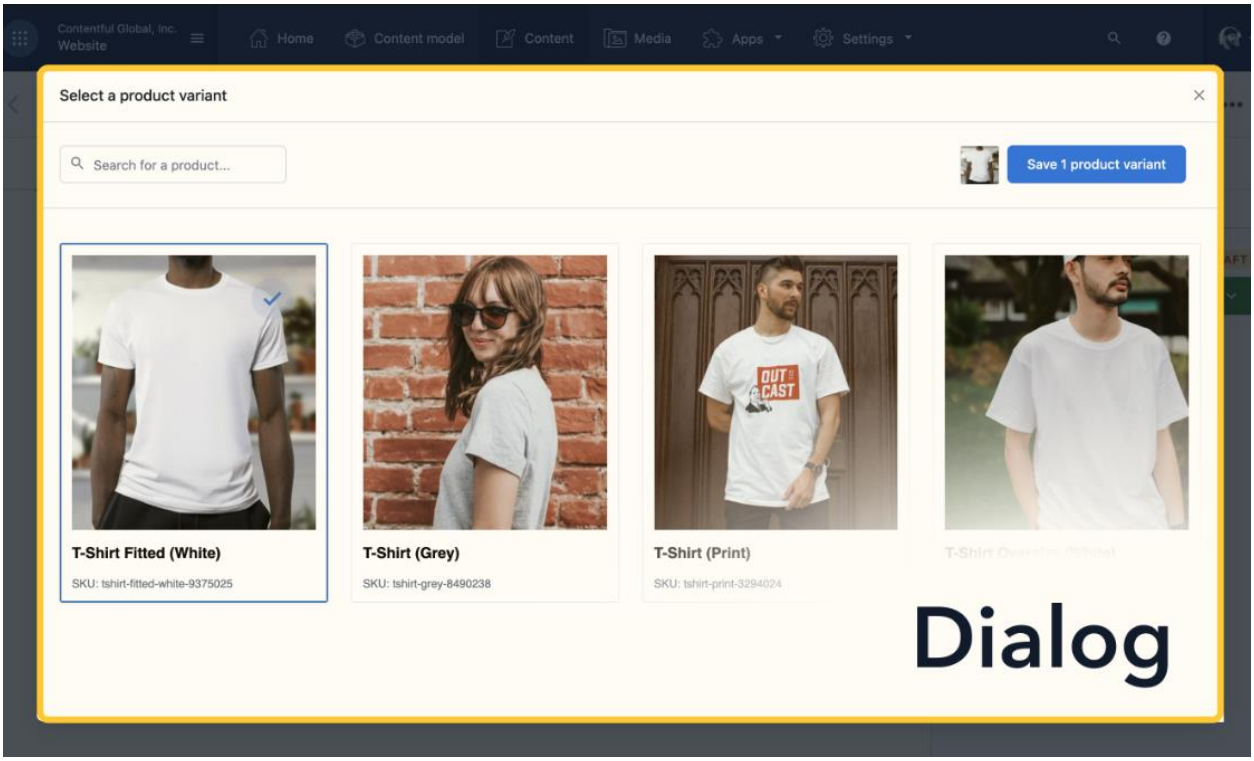


Abbildung 6: Dialog App Locations

A.8.2 Home App Location

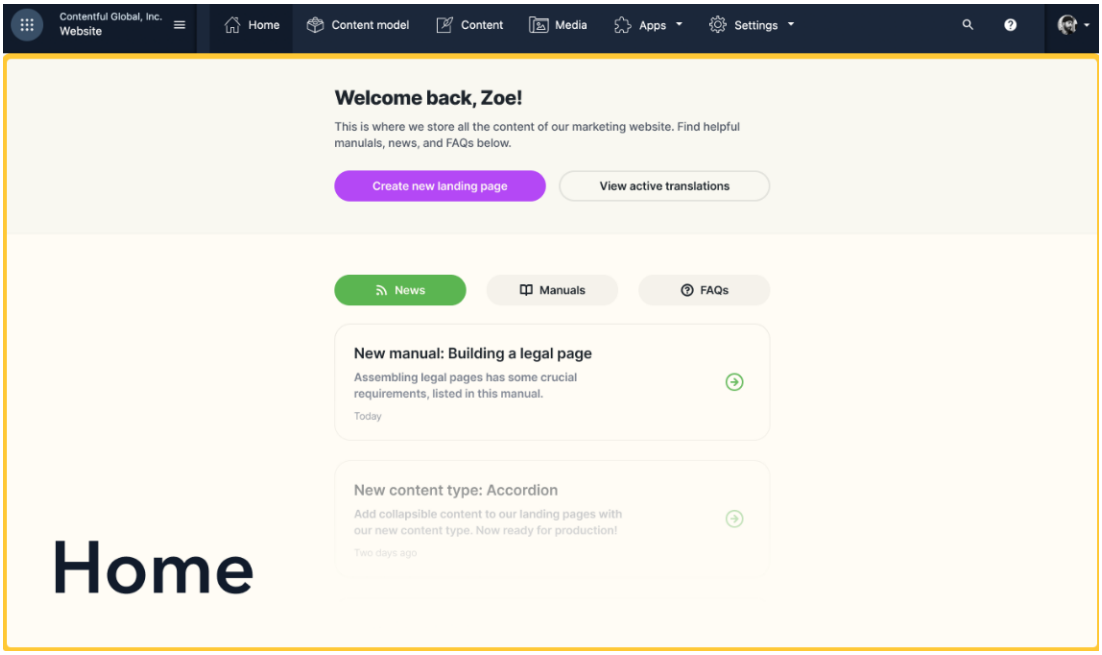


Abbildung 7: Home App Location

A.8.3 Entry Sidebar App Location

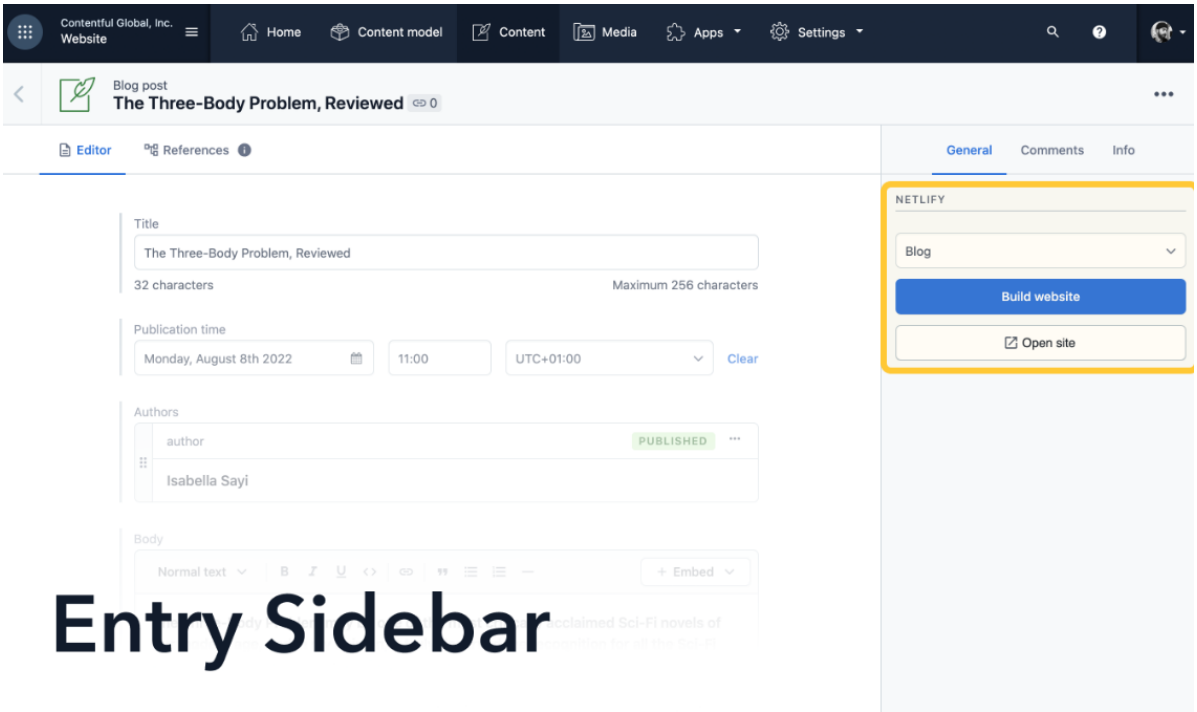


Abbildung 8: Entry Sidebar App Location

A.8.4 Entry Field App Location

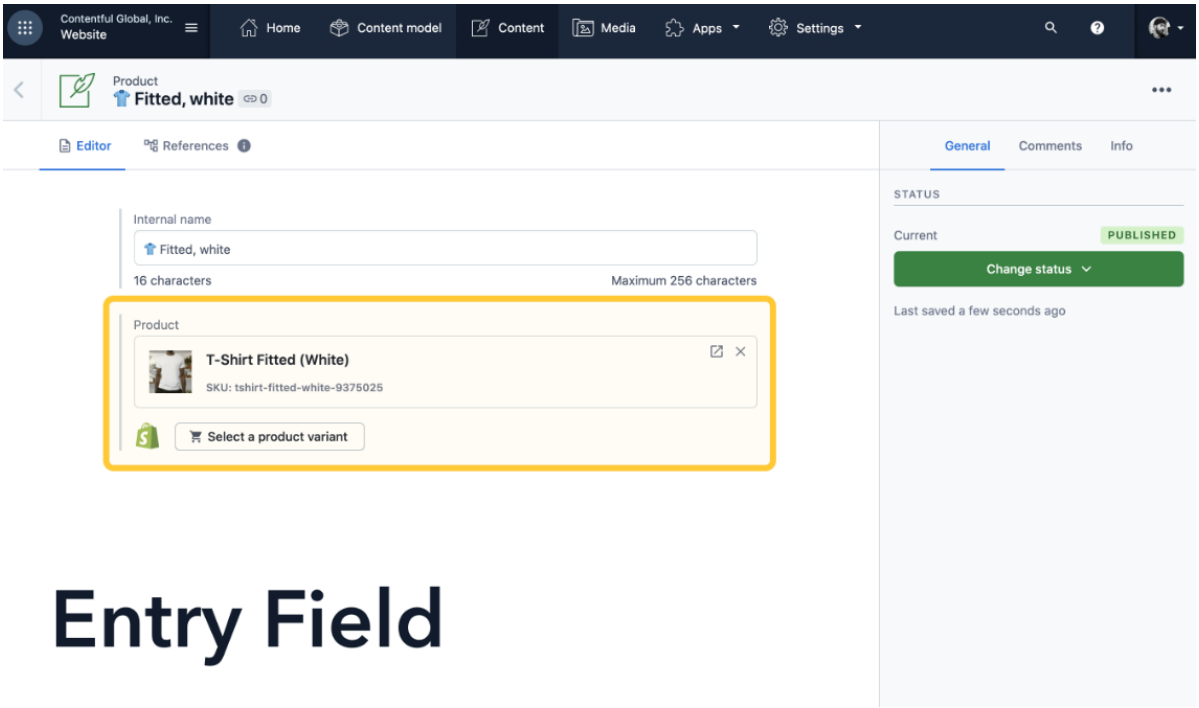


Abbildung 9: Entry Field App Location

A.8.5 Page App Location

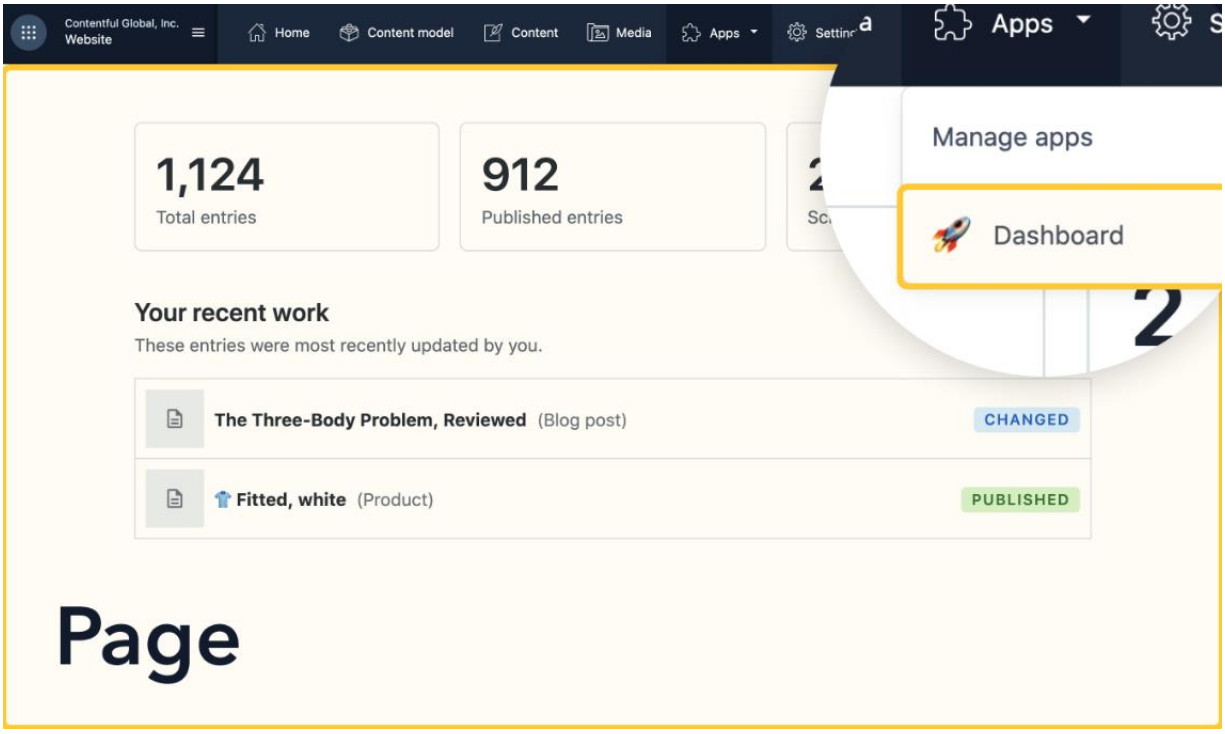


Abbildung 10: Page App Location

A.8.6 App Configuration App Location

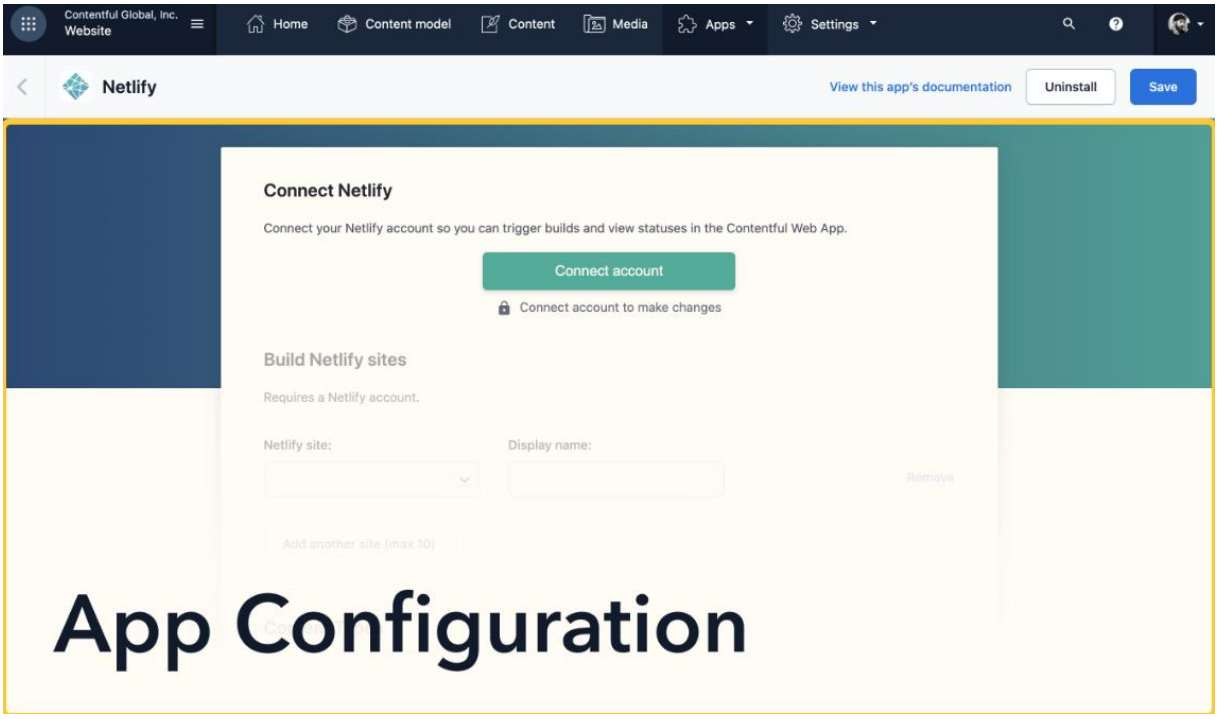


Abbildung 11: App Configuration App Location

A.9 Quellencode

A.9.1 Dialog.tsx Komponente (Quellencode)

src > components > Dialog.tsx > ...

```tsx
import React from "react";  7.3k (gzipped: 3k)
import { createClient, PlainClientAPI } from "contentful-management";  208.3k (gzipped: 41.6k)
import {
  Paragraph, Accordion, AccordionItem, List, ListItem, Note, TextLink,      You, 1 second ago • Uncommitted chi
} from "@contentful/forma-36-react-components";  198.4k (gzipped: 56k)
import getBufferedFetch from "./getBufferedEntries";
import { DialogExtensionSDK } from "@contentful/app-sdk";  16k (gzipped: 5.2k)
Philipp, 14 months ago | 2 authors (Philipp and others)
interface DialogProps {
  sdk: DialogExtensionSDK;
  cma: PlainClientAPI;
}

type ErrLog = { locale: string; snippets: { name: string; id: string }[] };

const Dialog = (props: DialogProps) => {
  const [allMissingSnippets, setAllMissingSnippets] = React.useState<ErrLog[]>(
    []
  );
  const [dataFetched, setDataFetched] = React.useState(false);
  const cma = createClient({ apiAdapter: props.sdk.cmaAdapter });
  // link to the contentful-management library by CMA
  const allLocales = props.sdk.locales.available;
  // allLocales is an array of all the locales available in the space (lusini)

  React.useEffect(() => {
    async function fetchData() {
      const space = await cma.getSpace(props.sdk.ids.space);
      // space (lusini) is the space object from the contentful-management library
      const environment = await space.getEnvironment(props.sdk.ids.environment);
      // environment is branch of the space object
      const allSnippets = await getBufferedFetch(environment, {
        content_type: "snippets",
        limit: 3000,
        bufferSize: 250,
      });
      let allMissing: ErrLog[] = [];

      allLocales.forEach((loc) => {
        // loc is the locale from the allLocales array from the contentful
        allSnippets.forEach((snippet) => {
          // snippet is of allSnippets array from the contentful
          if (!snippet.value) {
            console.log("Likely there is an Untitled Snippet!");
            // https://contentful-missing-snippets.netlify.app/
          }
          if (!(loc in snippet.value)) {
            // Handling missing snippet
            if (!allMissing.find((snips) => snips.locale === loc)) {
              // If the locale is not found in allMissing array, add it.
              allMissing.push({
                locale: loc,
                snippets: [
                  {
                    // de-DE acts like a reference to the snippet props from the contentful (like a basket)
                    name: snippet.fieldName["de-DE"],
                    id: snippet.contentfulID,
                  },
                ],
              });
            } else {
              // If the locale is found, update the existing entry.
              // This will data set will be displayed in the contentful web app as DOM Dialgoue box
              // mapping list into propper shape
              allMissing = allMissing.map((sni) => {
                if (sni.locale === loc) {
                  //sni.locale is the locale assainged from the sni object
                  return {
                    locale: loc,
                    snippets: [
                      ...sni.snippets, // not to lose the previous entries
                      {
                        name: snippet.fieldName["de-DE"],
                        id: snippet.contentfulID,
                      },
                    ],
                  };
                } else {
                  return sni;
                }
              });
            }
          }
        });
      });
      setAllMissingSnippets(allMissing);
      setDataFetched(true);
    }
    if (!dataFetched && allMissingSnippets.length === 0) {
      fetchData();
      return;
    }
    if (dataFetched && allMissingSnippets.length === 0) props.sdk.close();
  }, [allMissingSnippets, allLocales, cma, props.sdk, dataFetched]);

  if (allMissingSnippets.length === 0) return <></>;

```

Abbildung 12: Dialog.tsx Komponente (Quellencode)

A.9.2 getBufferedEntries.js (Quellencode)

src > components > JS getBufferedEntries.js > ...

You, 22 seconds ago | 2 authors (You and others)

```
1  export default async function getBufferedFetch(client, config) {
2    const { bufferSize, ...contentfulConfig } = config;
3
4    // Check if bufferSize is provided in the config
5    if (!bufferSize) throw new Error('bufferSize is required');
6
7    // Initialize an empty buffer to store fetched entries
8    const buffer = [];
9    let i = 0;
10
11   // Iterate until the total number of fetched entries reaches the specified limit
12   while (i * bufferSize < config.limit) {
13     // Fetch entries from Contentful with specified limit and skip
14     const entries = await client
15       .getEntries({
16         ...contentfulConfig,
17         limit: bufferSize,
18         skip: i * bufferSize,
19       })
20       .then((response) =>
21         // Extract necessary fields from each fetched entry metadata and format them
22         response.items.map((o) => ({
23           ...o.fields,
24           contentfulID: o.sys.id, // Required
25           updatedAt: o.sys.updatedAt, // Optional
26         }))
27       );
28
29     // Push fetched entries into the buffer
30     buffer.push(...entries);
31
32     // Break the loop if the number of fetched entries is less than bufferSize
33     if (entries.length < bufferSize) break;
34     i++;
35   }
36
37   // Return the buffer containing fetched entries, limited by the specified limit
38   return buffer.slice(0, config.limit);
39 }
40
```

Abbildung 13: getBufferedEntries.js (Quellencode)

A.9.3 Dialog.tsx HTML Teil (Quellencode)

```
97
98  // The following code is used to display the missing snippets in the contentful web app as
    a dialoge box location, when allMissingSnippets is not empty.
99    return (
100     <Paragraph>
101       <Note title="Please check the locales" noteType="negative">
102         Please add the translation for the following snippets:
103       </Note>
104
105       <Accordion>
106         {allMissingSnippets.map((missingSnippet) => {
107           return (
108             <AccordionItem
109               key={missingSnippet.locale}
110               title={missingSnippet.locale}
111             >
112               <List>
113                 {missingSnippet.snippets.map((snippet, i) => {
114                   return (
115                     <ListItem key={i}>
116                       <TextLink
117                         linkType="primary"
118                         href={
119                           "https://app.contentful.com/spaces/aza65graowyr/entries/" +
120                           snippet.id
121                         }
122                         target={"_blank"}
123                       >
124                         {snippet.name}
125                       </TextLink>
126                     </ListItem>
127                   );
128                 })}
129               </List>
130             </AccordionItem>
131           );
132         })}
133       </Accordion>
134     </Paragraph>
135   );
136 };
137
138 export default Dialog;
139
```

Abbildung 14: Dialog.tsx HTML Teil (Quellencode)

A.9.4 Unittest von Dialog.tsx (Quellencode)

Dialog.tsx | Dialog.spec.tsx 6, M

src > components > Dialog.spec.tsx > ...

You, 15 seconds ago | 2 authors (You and others)

```tsx
import React from "react";  7.3k (gzipped: 3k)
import { render, waitFor } from "@testing-library/react";  248.2k (gzipped: 60.5k)
import Dialog from "./Dialog";

// Mock the DialogExtensionSDK and createClient function
jest.mock("@contentful/app-sdk");
jest.mock("contentful-management", () => ({
  createClient: jest.fn(() => ({
    getSpace: jest.fn(() => ({
      getEnvironment: jest.fn(() => ({
        getEntries: jest.fn(() => ({
          items: [
            // mock snippets here if needed
          ],
        })),
      })),
    })),
  })),
}));

Run | Debug
it("renders nothing when there are no missing snippets", async () => {
  const sdk = { locales: { available: [] } }; // Provide a default value for locales
  const { container } = render(<Dialog sdk={sdk} />);
  await waitFor(() => expect(container.firstChild).toBeNull());
});


  Run | Debug
  it("renders missing snippets when there are some", async () => {
    const missingSnippets = [
      { locale: "en-US", snippets: [{ name: "Snippet 1", id: "123" }] },
    ];
    const { getByText } = render(
      <Dialog
        sdk={{ close: jest.fn() }}
        cma={{
          getSpace: jest.fn(() => ({
            getEnvironment: jest.fn(() => ({
              getEntries: jest.fn(() => ({
                items: [
                  {
                    // mock snippets here if needed
                  },
                ],
              })),
            })),
          })),
        }}
      />
    );

    await waitFor(() => {
      missingSnippets.forEach((missingSnippet) => {
        expect(getByText(missingSnippet.locale)).toBeInTheDocument();
        missingSnippet.snippets.forEach((snippet) => {
          expect(getByText(snippet.name)).toBeInTheDocument();
        });
      });
    });

  Run | Debug
  it("closes the dialog when there are no missing snippets", async () => {
    const closeMock = jest.fn();
    render(<Dialog sdk={{ close: closeMock }} />);
    await waitFor(() => expect(closeMock).toHaveBeenCalled());
  });
});
```

Abbildung 15: Unittest von Dialog.tsx (Quellencode)

A.9.6 ConfigScreen.tsx (Quellencode)

src > components > ConfigScreen.tsx > ...

Hadi Deniz, 2 years ago | 1 author (Hadi Deniz)

```tsx
import React, { useCallback, useState, useEffect } from 'react';  7.4k (gzipped: 3k)
import { AppExtensionSDK } from '@contentful/app-sdk';  16k (gzipped: 5.2k)
import { PlainClientAPI } from 'contentful-management';  182.7k (gzipped: 36.3k)
import { Heading, Form, Workbench, Paragraph } from '@contentful/forma-36-react-components';
import styles from './ConfigScreen.module.css';

export interface AppInstallationParameters {}

```

Hadi Deniz, 2 years ago | 1 author (Hadi Deniz)

```tsx
interface ConfigScreenProps {
  sdk: AppExtensionSDK;
  cma: PlainClientAPI;
}

const ConfigScreen = (props: ConfigScreenProps) => {
  const [parameters, setParameters] = useState<AppInstallationParameters>({});

  const onConfigure = useCallback(async () => {
    // This method will be called when a user clicks on "Install"
    // or "Save" in the configuration screen.
    // for more details see https://www.contentful.com/developers/docs/extensibility/
    ui-extensions/sdk-reference/#register-an-app-configuration-hook

    // Get current the state of EditorInterface and other entities
    // related to this app installation
    const currentState = await props.sdk.app.getCurrentState();

    return {
      // Parameters to be persisted as the app configuration.
      parameters,
      // In case you don't want to submit any update to app
      // locations, you can just pass the currentState as is
      targetState: currentState,
    };
  }, [parameters, props.sdk]);

  useEffect(() => {
    // `onConfigure` allows to configure a callback to be
    // invoked when a user attempts to install the app or update
    // its configuration.
    props.sdk.app.onConfigure(() => onConfigure());
  }, [props.sdk, onConfigure]);

  useEffect(() => {
    (async () => {
      // Get current parameters of the app.
      // If the app is not installed yet, `parameters` will be `null`.
      const currentParameters: AppInstallationParameters | null =
        await props.sdk.app.getParameters();

      if (currentParameters) {
        setParameters(currentParameters);
      }

      // Once preparation has finished, call `setReady` to hide
      // the loading screen and present the app to a user.
      props.sdk.app.setReady();
    })();
  }, [props.sdk]);

  return (
    <Workbench className={styles.Container}>
      <Form>
        <Heading>App Config</Heading>
        <Paragraph>Welcome to your contentful app. This is your config page.</Paragraph>
      </Form>
    </Workbench>
  );
};

export default ConfigScreen;

```

Abbildung 16: ConfigScreen.tsx (Quellencode)

A.9.7 Unittest von ConfigScreen.tsx (Quellencode)

```
import React from 'react';  7.3k (gzipped: 3k)
import ConfigScreen from './ConfigScreen';
import { render } from '@testing-library/react';  248.2k (gzipped: 60.5k)
import { mockCma, mockSdk } from '../../test/mocks';

Run | Debug
describe('Config Screen component', () => {
  Run | Debug
  it('Component text exists', async () => {
    const { getByText } = render(<ConfigScreen cma={mockCma} sdk={mockSdk} />);

    // simulate the user clicking the install button
    await mockSdk.app.onConfigure.mock.calls[0][0]();

    expect(
      getByText('Welcome to your contentful app. This is your config page.')
    ).toBeInTheDocument();
  });
});
```

Abbildung 17: Unittest von ConfigScreen.tsx (Quellencode)

A.10 Benutzeroberfläche (UI)

A.10.1 Liste von Contentful Applications

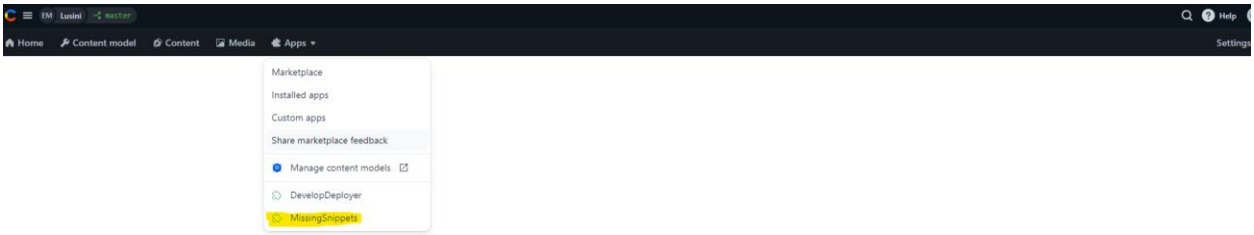


Abbildung 18: Liste von Contentful Applications

A.10.2 Benutzeroberfläche von MissingSnippets (geschlossen)

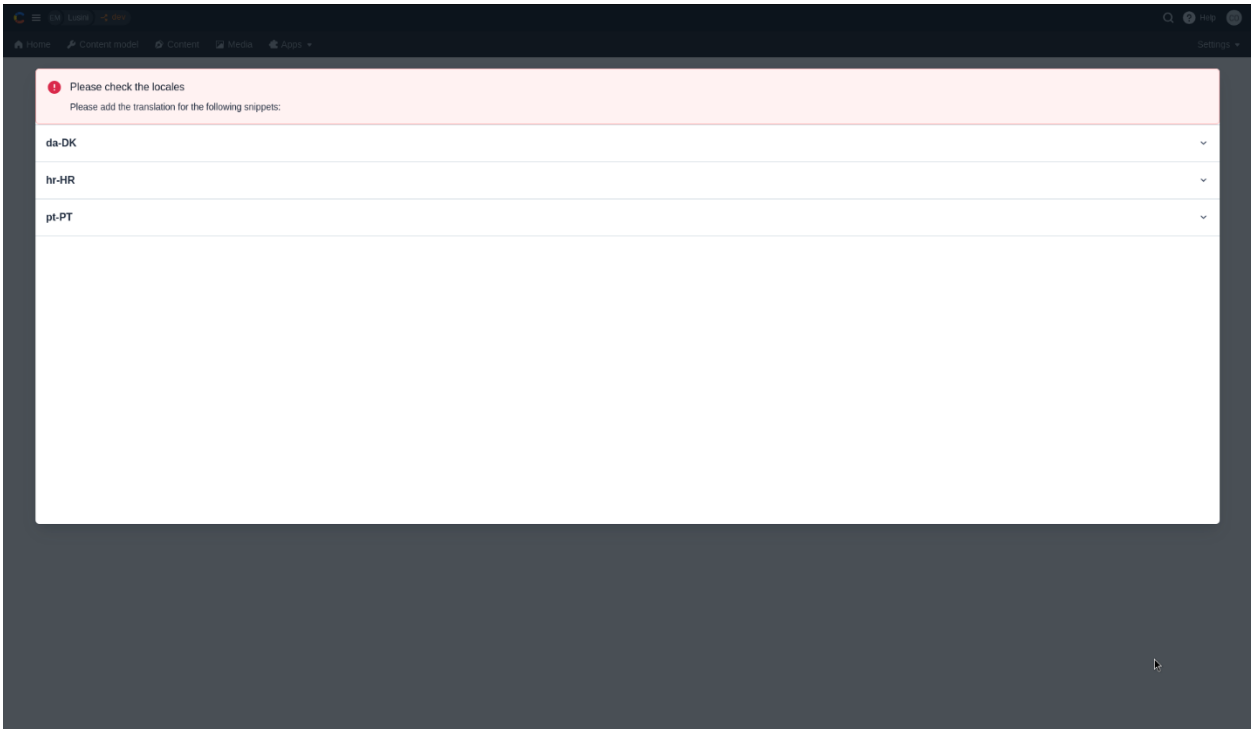


Abbildung 19: Benutzeroberfläche von MissingSnippets (geschlossen)

A.10.3 Benutzeroberfläche von MissingSnippets (geöffnet)

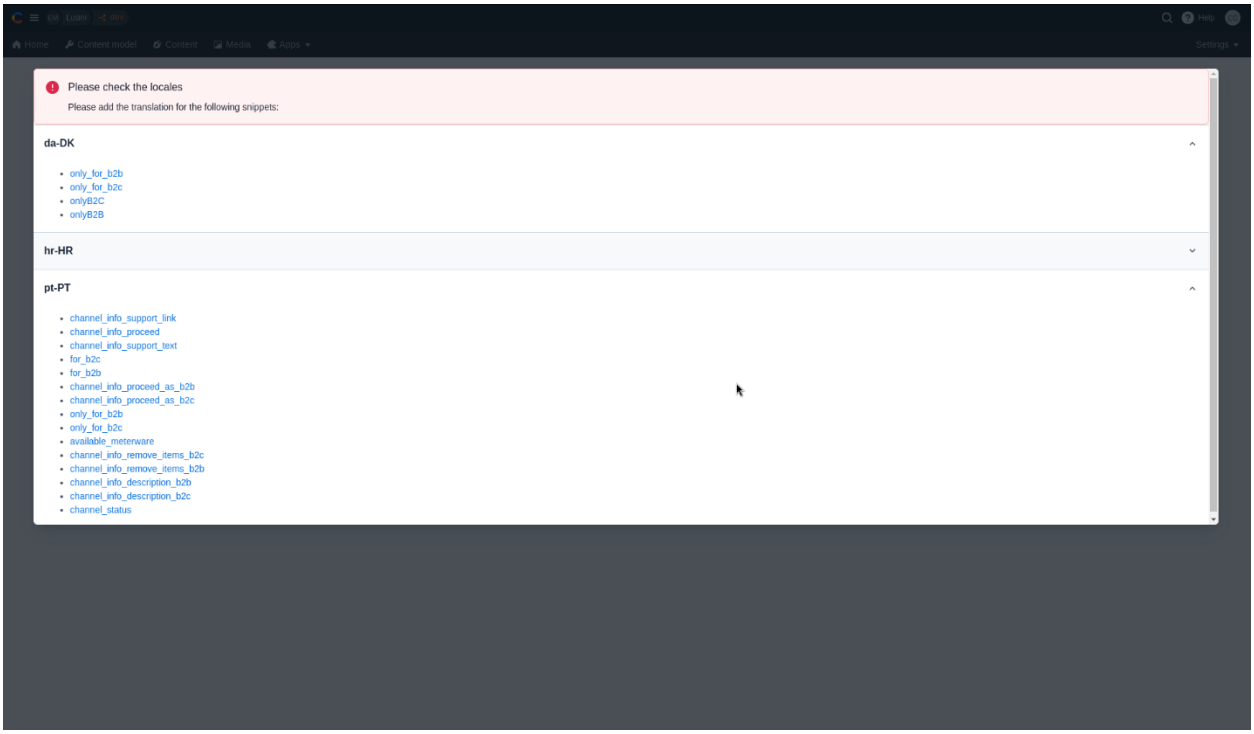


Abbildung 20: Benutzeroberfläche von MissingSnippets (geöffnet)

A.10.4 Benutzeroberfläche nach Weiterleitung durch MissingSnippets-link

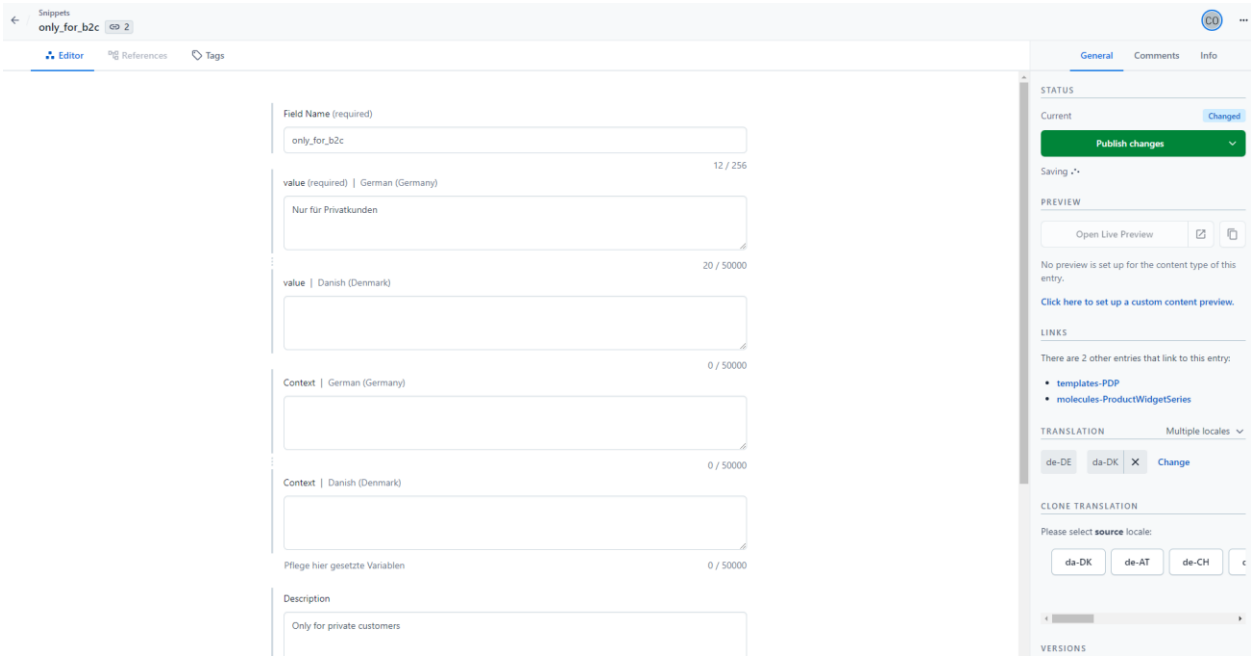
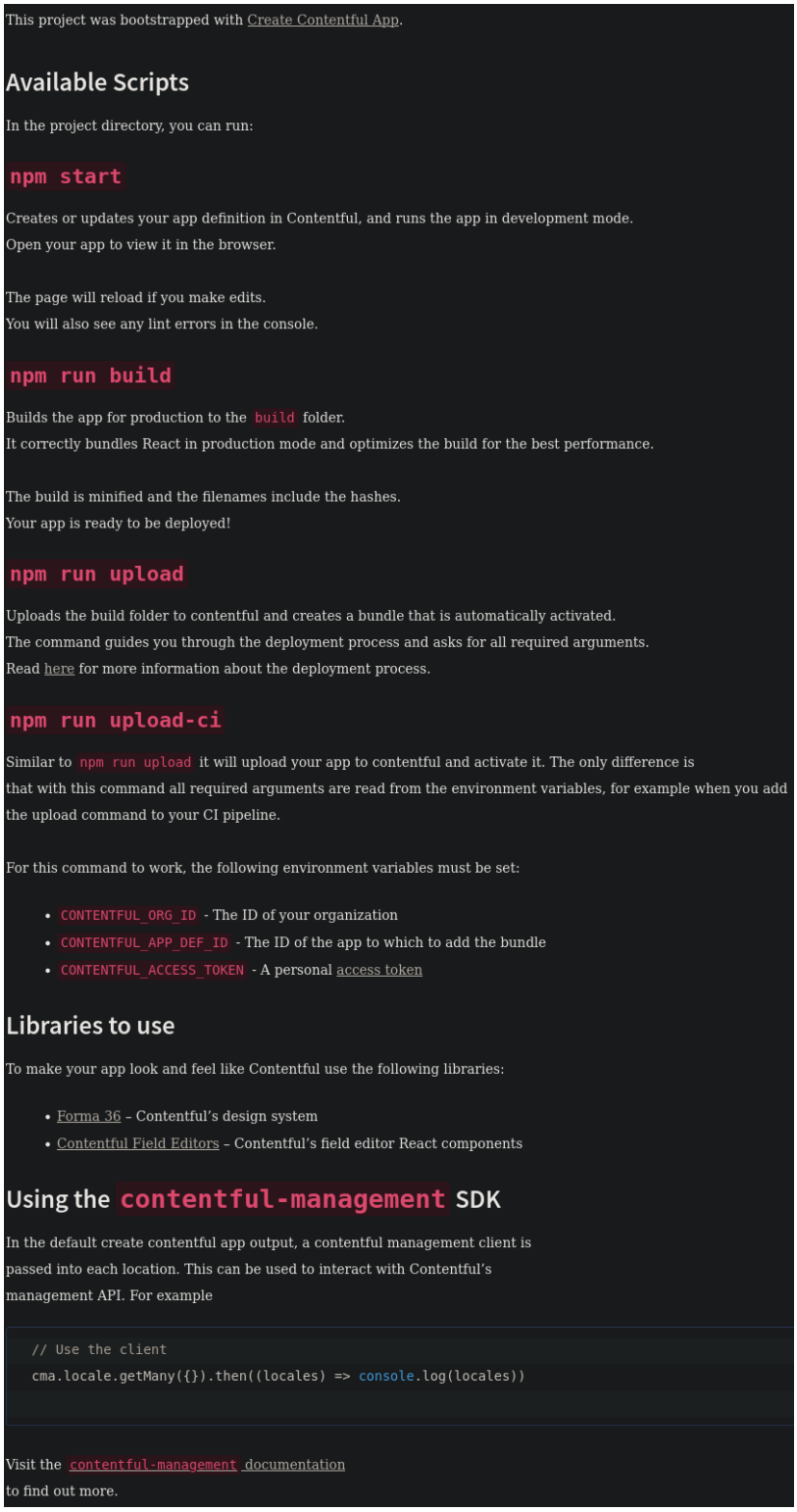


Abbildung 21: Benutzeroberfläche nach Weiterleitung durch MissingSnippets-link

A.11 Auszug aus Entwickler Dokumentation



This project was bootstrapped with Create Contentful App.

Available Scripts

In the project directory, you can run:

`npm start`

Creates or updates your app definition in Contentful, and runs the app in development mode.
Open your app to view it in the browser.

The page will reload if you make edits.
You will also see any lint errors in the console.

`npm run build`

Builds the app for production to the `build` folder.
It correctly bundles React in production mode and optimizes the build for the best performance.

The build is minified and the filenames include the hashes.
Your app is ready to be deployed!

`npm run upload`

Uploads the build folder to contentful and creates a bundle that is automatically activated.
The command guides you through the deployment process and asks for all required arguments.
Read here for more information about the deployment process.

`npm run upload-ci`

Similar to `npm run upload` it will upload your app to contentful and activate it. The only difference is
that with this command all required arguments are read from the environment variables, for example when you add
the upload command to your CI pipeline.

For this command to work, the following environment variables must be set:

- `CONTENTFUL_ORG_ID` - The ID of your organization
- `CONTENTFUL_APP_DEF_ID` - The ID of the app to which to add the bundle
- `CONTENTFUL_ACCESS_TOKEN` - A personal access token

Libraries to use

To make your app look and feel like Contentful use the following libraries:

- Forma 36 – Contentful's design system
- Contentful Field Editors – Contentful's field editor React components

Using the `contentful-management` SDK

In the default create contentful app output, a contentful management client is
passed into each location. This can be used to interact with Contentful's
management API. For example

```
// Use the client
cma.locale.getMany({}).then((locales) => console.log(locales))
```

Visit the `contentful-management` documentation
to find out more.

Abbildung 22: Auszug aus README.md für Entwickler

Eidesstattliche Erklärung

Ich, Kacper Sobczak, versichere hiermit, dass ich meine Dokumentation zur betrieblichen Projektarbeit mit dem Thema

Entwicklung eines Contentful Plugins zur Überprüfung der Validität von Einträgen von Contentfeldern

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, wobei ich alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.