Hochschule Fulda
Fachbereich Angewandte Informatik
Studiengang WI (B.Sc.)

Low code Applikation Entwicklung mit SAP AppGyver im Vergleich zu nativen Fiori Entwicklung

Bachelorarbeit von Fangfang Tan
Matrikelnummer: 1222047

Erstgutachterin:	Prof. Dr. Norbert Ketterer
Zweitgutachter:	M. A. Mike Zaschka
Abgabetermin:	6. Februar 2023

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die Zitate deutlich kenntlich gemacht zu haben. Ich erkläre weiterhin, dass die vorliegende Arbeit in gleicher oder ähnlicher Form noch nicht im Rahmen eines anderen Prüfungsverfahrens eingereicht wurde.

Bad Hersfeld, den 25. Januar 2023

DEINE Unterschrift

Danksagung

An erster Stelle möchte ich meiner Betreuer Prof. Dr. Norbert Ketterer danken, der mich richtungsweisend und mit viel Engagement während meiner Arbeit begleitet hat.

Besonderen Dank gebührt meinem Betreuer Mike Zaschka für die hilfreichen Anregungen und die konstruktive Kritik und vieles mehr bei der Erstellung dieser Arbeit. Ein herzliches „Dankeschön! “ geht auch an allen anderen Kollegen und Kolleginnen der Firma p36, die mich herzlich aufgenommen und mir während der Schreibphase meiner Bachelorarbeit wertvollen Unterstützungen gegeben haben.

Zusammenfassung

In dieser Arbeit werden xxxx.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quelltextverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface – Programmierschnittstelle
BTP	Business Technology Plattform
BAS	Business Application Studio
CAP	SAP Cloud Application Programming Model
CDS	Core Data Servicel
CORS	Cross Origin Resource Sharing
CRM	Customer Relationship Management
CRUD	Create Read Update Delete – Vier fundamentale Optionen des Datenmanagement
CSS	Cascading Style Sheets
CSV	Comma-Separated-Values – Datenformat
GPS	Global Positioning System
HTML	HyperText Markup Language – Auszeichnungssprache
IDE	Integrierte Entwicklungsumgebung
JSON	JavaScript Object Notation – Datenserialisierungsformat
LCNC	Low-Code/No-Code
MVC	Model-View-Controller – Software Designmuster
OData	Open Data Protocol
NDC	Native Device Capabilities

NPM	Node Package Manager
PDF	Portable Document Format
PT	Personentage
SAP	Systemanalyse Programmentwicklung – Softwarekonzern
SAPUI5	SAP UI Development Toolkit für HTML5
SDK	Software Development Kit
UI	User Interface
URL	Uniform Resource Locator – Internetadresse
UX	User Experience
VS-Code	Visual Studio Code – Software Entwicklungstool
XML	Extensible Markup Language – erweiterbare Auszeichnungssprache
YAML	YAML Ain't Markup Language – Datenserialisierungsformat

Kapitel 1

Einleitung

1.1 Problemstellung und Motivation

Seit die Branchenanalysten von Forrester Research im Jahr 2014 erstmals das Konzept von Low-Code erwähnten [RJR22], hat sich der zugehörige markt rasant entwickelt. Immer mehr Unternehmen nutzen Low-Code-Plattformen, um Anwendungen schneller zu entwickeln und damit den digitalen Wandeln zu beschleunigen. Die Analysten von Gartner erwarten, dass im Jahr 2025 rund 70% der von Unternehmen entwickelten Anwendungen auf Low-Code- Technologien basieren werden [HV22].

Im Bereich der Enterprise Software werden die meisten Low-Code-Plattformen verwendet, um eine bestimmte Anwendungsform in einem spezifischen Kontext zu entwickeln. In der Studie „No-Code/Low-Code 2022 “ im Magazin COMPUTERWOCHE, gibt die Mehrheit der gefragten Unternehmen an, dass sie Low-Code hauptsächlich in den Bereichen CRM (34%) und ERP (31%) einsetzen. Speziell ausgerichtete Low-Code Plattformen werden ebenfalls im HR-Umfeld (19%) verwendet, sowie für die Erstellung digitaler Workflows und Verwaltungsprozesse (jeweils 16%). Nur 10% der Befragten nutzen eine universell einsetzbare Plattform, die sich für übergreifende und flexible Geschäftsprozesse eignet. Laut Jürgen Erbeldinger, einem Low-Code-Experten der Low-Code-Plattform ESCRIBA, fehlt den Plattformen hierfür die entsprechende Tiefe [AS22].

AppGyver betrachtet sich selbst als die weltweit erste professionelle Low-Code Plattform, die es ermöglicht, Anwendungen für unterschiedliche Geschäftsprozesse, Anwendungsszenarien und auch Endgeräte zu erstellen [SAP22b]. Im Februar

2021 wurde AppGyver von dem Marktführer im Bereich Enterprise Software SAP übernommen und wird seitdem in deren Entwicklungsportfolio rund um die SAP Business Technology Plattform eingegliedert [Cen21], [Com22b]. Seit dem 15. November 2022 wurde SAP AppGyver in SAP Build App umbenannt und ist nun Teil von SAP Build. AppGyver steht damit in Konkurrenz zu etablierten Tools und Frameworks zur Anwendungsentwicklung: SAPUI5 ist ein JavaScript-Framework und bildet die Grundlage nahezu aller heute entwickelten SAP-Oberflächen. Die Erstellung von SAPUI5-Anwendungen setzt jedoch ein tieferes technisches Verständnis voraus. Basierend auf SAPUI5 steht mit SAP Fiori Elements ein Framework zur Verfügung, welches durch Annotationen die einfache Erstellung von datengetriebenen Oberflächen erlaubt. Dank der guten Integration in die SAP-eigenen Entwicklungsumgebungen, das SAP Business Application Studio, kann SAP Fiori Elements in Kombination mit dem SAP Application Programming Model auch im Bereich der Low-Code Entwicklung platziert werden [Ele22]. Für Unternehmen ergibt sich in Zukunft nun die Fragestellung, welche Plattform und Tools im SAP-Umfeld eingesetzt werden sollten, um Anwendungen zu entwickeln. Die Aufgabe dieser Bachelorarbeit besteht darin, den Entwicklungsprozess von SAP AppGyver, SAPUI5, sowie Fiori Elements in Kombination mit dem SAP Application Programming Model zu bewerten, Vor- und Nachteile der jeweiligen Lösung herauszuarbeiten und dadurch ein Entscheidungsmatrix zu entwerfen, welche die Wahl zwischen diesen drei Technologien vereinfacht.

Diese wissenschaftliche Arbeit wird dabei unterstützt durch die Firma p36 GmbH. P36, mit dem Sitz im Bad Hersfeld, wurde 2015 von Patrick Pfau und Robin Wennemuth gegründet. Das mit 27 Mitarbeitern noch recht kleine, aber stark wachsende Softwareunternehmen besitzt einen starken Fokus auf die Entwicklung von Cloud-basierten Anwendungen im SAP-Umfeld [Gmb22]. Bei der Umsetzung der Anwendungen setzt p36 überwiegend auf die sehr technische SAPUI5-Entwicklung und evaluiert derzeit den Einsatz von Low-Code-Plattformen. p36 stellt deswegen einen, sich an reellen Kundenanforderungen orientierenden, Anwendungsfall zur Verfügung, der im Rahmen der Arbeit als Grundlage der Evaluierung dienen soll.

1.2 Ziele der Arbeit

Die folgenden Fragen werden in der Bachelorarbeit behandelt werden:

- Was genau verbirgt sich hinter dem Begriff Low-Code und wie grenzt sich Low-Code von bisherigen Arten der Entwicklung ab?

- Wie wird eine benutzerspezifische Anwendung mit dem Low-Code/No-Code basierten Tool SAP AppGyver implementiert?
- Wie wird eine benutzerspezifische Anwendung mit SAPUI5 implementiert?
- Wie wird eine benutzerspezifische Anwendung mit Fiori Elements implementiert?
- Welche Vor- und Nachteile dieser drei Technologien lassen sich durch die exemplarische Umsetzung herausstellen?
- Welche Technologie eignet sich in Zukunft für welche Umsetzungsszenarien?

1.3 Beschreibung des Anwendungsfalls

In dieser Arbeit werden die drei genannten Technologien verwendet, um eine konkrete Anwendung zu entwickeln. Der Anwendungsfall definiert sich, wie folgt:

Name:	Applikation zur Verwaltung von Produktinformationen
Umsetzung in:	• SAP Fiori Elements mit SAP Business Application Studio (Backend + UI) • SAP AppGyver (UI) • SAPUI5 (UI)
Anforderungen Backend:	• Bereitstellung einer Datenbank-Entität Products mit folgenden Eigeschaften: – ID; title; materialNumber; description; price; stock • Bereitstellung eines OData-Services zum Auslesen, Erstellen und Aktualisieren (CRUD) der Produkte
Anforderungen Frontend:	• Funktionen: – Listenansicht zur Anzeige aller Produkte – Einzelansicht für ein Produkt – Maske zum Pflegen eines einzelnen Produkts • Datenanbindung: – Anbindung an den OData-Service zum Auslesen und Schreiben von Produkten • Look and Feel: – Implementierung in Anlehnung an die SAP UX-Guideline SAP Fiori

Tabelle 1.1: Definition der Anwendungsfall

Zusätzlich zu den umzusetzenden Funktionalitäten wird eine Reihe weiterer

Funktionen ohne praktische Umsetzung untersucht, um SAPUI5, Fiori Elements und AppGyver tiefergehend zu evaluieren. Diese Funktionen umfassen:

- Integration von Suchfilter und Paginierung
- Integration von Bild und PDF-Datei
- Integration von Barcode-Scanner-Funktionen
- Nutzung mobiler Funktionen wie Sensoren
- Möglichkeiten zum Deployment für unterschiedliche Endgeräte
- Freie Gestaltungsmöglichkeiten

1.4 Aufbau der Arbeit

Die vorliegende Bachelorarbeit gliedert sich in insgesamt sechs Kapitel. In der Einleitung werden die Problemstellung und Motivation, die Ziele der Arbeit und der umzusetzende Anwendungsfall vorgestellt.

Im 2. Kapitel werden zunächst die grundlegenden Konzepte erläutert. Der erste Abschnitt beschäftigt sich mit dem Low-Code/No-Code (LCNC) Konzept und liefert eine Definition von LCNC und einen Überblick über existierende LCNC-Plattformen. Kapitel 2 beinhaltet ebenfalls einen Überblick über die Architektur von SAP-Anwendungen in der SAP Business Technology Plattform, sowie, im dritten Abschnitt, die Grundlagen von SAP Fiori. Der vierte, fünfte und letzte Abschnitt dieses Kapitels beschreibt die Grundlagen von AppGyver, SAPUI5 und Fiori Elements, sowie die Entwicklungsumgebung, in denen der genannte Anwendungsfall entwickelt wird, nämlich SAP Business Application Studio, Composer Pro und Visual Studio Code.

Kapitel 3 bis 5 bilden den Hauptteil der Bachelorarbeit. Im dritten Kapitel wird der Umsetzungsprozess des Anwendungsfalls mit Fiori Elements, AppGyver und SAPUI5 beschrieben. Die zu beschreibenden Funktionen umfassen:

- Bereitstellung eines OData-Services zum Auslesen, Erstellen und Aktualisieren der Produkte
- Erstellung einer Listenansicht zur Anzeige aller Produkte
- Erstellung einer Einzelansicht für ein Produkt
- Erstellung einer Maske zum Pflegen eines einzelnen Produkts

Im 4. Kapitel werden weitere Funktionen, ohne technische Implementierung, untersucht, um Fiori Elements, AppGyver und SAPUI5 eingehender zu bewerten. Basierend auf Kapitel 3 und Kapitel 4 konzentriert sich Kapitel 5 auf die Gegenüberstellung und Bewertung der drei Tools. Hierfür werden die Be-

wertungsmatrizen definiert, die Bewertung durchgeführt und anschließend die Bewertungsergebnisse diskutiert und interpretiert. Kapitel 6, das letzte Kapitel der Bachelorarbeit, enthält abschließend ein Fazit und einen Ausblick auf die künftige Forschung.

Kapitel 2

Grundlagen

2.1 Entwicklung mit Low-Code/No-Code

Low-Code/No-Code ist ein Ansatz der Softwareentwicklung, bei dem Anwendungen mit wenig oder gar keinem selbst programmierten Code erstellt werden können. Pro-Code hingegen bezieht sich auf die klassische Entwicklung, bei der die Codezeilen von Hand geschrieben werden. Anstatt auf komplexe Programmiersprachen zurückzugreifen, kann LCNC-Entwicklung die Anwendungen durch visuelle Programmierung, also durch Anklicken, Ziehen und miteinander verbinden von Anwendungskomponenten, erstellen. Die Low-Code/No-Code-Plattformen bieten hierfür spezielle visuelle Programmierumgebungen an, die aus einer Reihe an vorgefertigten Code-Bausteinen und den Möglichkeiten diese in Form einer Anwendung zusammenzusetzen, bestehen. No-Code-Plattformen ersetzen die traditionelle codebasierte Entwicklungsumgebung dabei vollständig, während bei Low-Code-Plattformen möglicherweise Basis-Programmierkenntnisse erforderlich sind.

Auch wenn es bereits in der Vergangenheit Ansätze zur visuellen Programmierung gab, so ist die derzeitige LCNC-Entwicklung aufgrund des Reifegrades des Toolings eine ernsthafte Alternative zur Pro-Code-Entwicklung. Einige Experten glauben, dass LCNC die Zukunft der Softwareentwicklung ist, weil es einen schnelleren Entwicklungsprozess ermöglicht. Mit den einfach zu bedienenden visuellen Benutzeroberflächen, sowie den ausgereiften Entwicklungstoolkits ist man in der Entwicklung deutlich schneller, als wenn man Tausende von Codezeilen schreiben muss. Neben dem Zeitfaktor spielt auch die damit einzusparenden Kosten eine große Rolle [Con22].

Ein weiterer Vorteil der LCNC-Entwicklung ist, dass sie den Mangel an qualifizierten Entwicklern kompensiert. LCNC-Entwicklung eignet sich für Entwickler aller Niveaus. Die No-Code-Plattformen sind insbesondere für Citizen Developer sinnvoll, die möglicherweise überhaupt keine Programmierausbildung haben. Ein Citizen Developer ist ein Mitarbeiter, der mit zugelassenen Tools Anwendungen für eigene Nutzung oder die Nutzung durch andere erstellt [GG22]. Darüber hinaus bieten LCNC-Plattformen professionellen Entwicklern Unterstützung, um die aufwändigen zugrundeliegenden architektonischen und infrastrukturellen Aufgaben zu reduzieren.

Der Markt für LCNC-Plattformen ist in den letzten Jahren deutlich gewachsen. Nach Angabe von G2, eine der Website für Softwarelisten und Bewertungen, gibt es (Stand November 2022) 226 Low-Code Plattformen [Ove22a] und 288 No-Code Plattformen [Ove22b] auf dem Markt. Neben spezialisierten Unternehmen/Start-Ups, stellen auch größere Unternehmen LCNC-Platformen für ihr jeweiliges Ökosystem zur Verfügung. Dazu einige Beispiele:

App Engine von ServiceNow wurde im März 2021 veröffentlicht [doc22c]. Es ermöglicht großen Unternehmen Low-Code-Anwendungen zu erstellen und breitzustellen. ServiceNow stellt eine Reihe an Entwicklungsvorlagen für gängige Anwendungsfälle bereit, um die Erstellung der Anwendungen zu erleichtern [CVE22]. App Engine erlaubt den Nutzern allerdings auch, Code mit traditionellen Programmiersprachen wie HTML, Javascript sowie CSS zu schreiben und zu bearbeiten.

Salesforce, vom gleichnamigen Unternehmen, ist eine Plattform für Customer-Relationship-Management (CRM) und besitzt umfangreiche Funktionen einer App-Entwicklungsplattform, um die Standard-CRM-Funktionalitäten der Plattform zu erweitern. Mit Hilfe der visuellen Programmierung können Workflow-basierte Anwendungen schnell erstellt werden, um Geschäftsprozesse abzubilden oder Kunden Zugang zu wichtigen Informationen zu gewähren. Die Salesforce-Plattform bietet neben dem Low-Code-Ansatz auch eine vollständig angepasste Anwendungsentwicklung für unterschiedliche Programmiersprachen und ist daher auch geeignet für den Code-basierten Ansatz [G222].

OutSystems vom gleichnamigen deutschen Hersteller, ist ein Beispiel für eine spezialisierte LCNC-Plattform ohne direkte Einbindung in ein größeres Ökosystem. Auch hier steht die visuelle Full-Stack-Entwicklung im Vordergrund, mit der Benutzeroberflächen, Geschäftsprozesse, Logik und Datenmodelle aufgebaut und implementiert werden können. Auch bei OutSystems ist es jedoch möglich, eigenen Code für die Anwendungserstellung hinzufügen

Der Wettbewerb im Trend-Thema LCNC ist heute sehr stark. Auf der einen Seite gibt es die großen Unternehmens wie ServiceNow und Salesforce, die über viele Ressourcen und Fachkräfte für die Entwicklung ihrer Plattformen verfügen und ihre Kunden mit der Bereitstellung von LCNC-Funktionalitäten weiter an die Plattform binden wollen. Die resultierenden Anwendungen sind zumeist plattformabhängig, haben jedoch den großen Vorteil, dass die Daten aus dem jeweiligen Ökosystem auch anwendungsübergreifend wiederverwendet werden können. Auf der anderen Seite gibt es die spezialisierten LCNC-Anbieter, wie OutSystems und Mendix, mit denen die Benutzer unabhängige Anwendungen entwickeln können und weniger an eine Plattform oder einen Anbieter gebunden sind [CVE22].

Im weiteren Verlauf dieser Arbeit soll der Fokus auf der LCNC-Platform SAP AppGyver liegen. AppGyver ist einer der Top-Anbieter für die LCNC-Entwicklung und kann als hybride Plattform angesehen werden. Ursprüngliche unabhängig und spezialisiert, entwickelt sich AppGyver durch den Aufkauf durch SAP und der Integration in das SAP Ökosystem zu einer leistungsfähigen Plattform, die beide Welten miteinander verbindet. Auf AppGyver wird in Abschnitt 2.4 näher eingegangen.

2.2 Architektur von SAP-Anwendungen in der SAP Business Technology Plattform

Der praktische Teil dieser Arbeit befasst sich mit der Erstellung von Anwendungen in den drei gewählten Technologien: AppGyver, SAP Fiori Elements und SAPUI5. Die grundlegende Architektur der Anwendungen orientiert sich an der heutigen Referenzarchitektur von Anwendungen auf der SAP Business Technology Plattform. Frühere SAP-Standards zur Erstellung von web-basierten Anwendungen waren eng gekoppelt mit den Backendsystemen und wurden in den jeweiligen Programmiersprachen erstellt – wie z.B. WebDynpro für Java oder WebDynpro für ABAP. Der vollständige HTML-Code, inklusive der darzustellenden Daten, wurde serverseitig generiert und das Resultat an das Endgerät übermittelt und dort durch den Browser interpretiert [Eng20, S.46]. Aufgrund der Notwendigkeit des so genannten Server-Roundtrips hatte diese Technologie einige Nachteile:

- Enge Kopplung von Daten und Darstellung.
- Aufgrund der Komplexität musste der generierte HTML-Code server-seitig gerendert werden. Deshalb war es möglich, dass die Anwendung auf dem

Endgerät nicht optimal zur Darstellung kam.
- Es gab nur sehr eingeschränkte Möglichkeiten, die Benutzeroberfläche zu gestalten.
- WebDynpro unterstützt keine Gestensteuerung oder sonstige Technologien, die für mobile Endgeräte notwendig sind.
- Wenn die Bandbreite zwischen dem Endgerät und dem Server unzureichend ist, kann die Wartezeit sehr lang sein.

Seit 2012 verfolgt SAP einen neuen Ansatz für webbasierte Anwendungen. Die Daten und ihre Bereitstellung als OData-Service werden von der eigentlichen Darstellung im Browser getrennt. Der OData-Service dient als Kommunikator zwischen Backend und Frontend und wird von der UI konsumiert.

Der erste Schritt dieses Ansatzes wurde in der On-Premise-Welt mit SAP Netweaver Gateway realisiert. Danach wurde das Konzept auch in die Cloud überführt und bietet dort die Möglichkeit, eigene OData-Services bereitzustellen oder Services aus der On-Premise-Welt zu integrieren. Dieser Ansatz hat viele Vorteile:

- Durch die Trennung von Daten und Benutzeroberfläche kann die UI sehr flexibel gestaltet werden.
- Die einmal bereitgestellten Daten können von unterschiedlichen Frontend-Applikationen genutzt werden.
- Die UI kann in unterschiedlichen Technologien und auf unterschiedlichen Endgeräten erstellt werden und dort auch native (mobile) Funktionen unterstützen.
- Die reine Bereitstellung der Daten auf dem Server ist deutlich schneller und demnach sind die Wartezeiten kürzer.

Abbildung 2.1 zeigt die Architektur einer moderner SAP-Anwendung in der SAP Business Technology Plattform. Auf der mittleren Ebene befindet sich die SAP Business Technology Platform, welche die zentrale Plattform zur Bereitstellung von Daten für webbasierte Anwendungen ist. In der BTP lassen sich eigene Datenbanken halten und eigene Services zur Verfügung stellen. Es ist jedoch ebenfalls möglich, Daten aus der SAP On-Premise-Welt (via Cloud Connector) oder auch von Drittanbieter-Systemen zentral zu integrieren. Die Daten werden jeweils als REST oder OData-Services zur Verfügung gestellt und können von Anwendungen auf der Benutzerseite konsumiert werden.

Für diese Bachelorarbeit wird ein OData-Service auf der SAP Business Technology Plattform erstellt. Auf die Anbindung eines On-Premise-Systems oder

das eines Drittanbieters wird an dieser Stelle verzichtet. Für die Erstellung des Backend-Parts (Datenbank + OData-Service) wird auf Fiori Elements und das SAP Cloud Application Programming Model (kurz: SAP CAP) zurückgegriffen. Zwar stehen auf der BTP technologisch auch andere Backend-Frameworks zur Verfügung, der Entwicklungsansatz mit SAP CAP und Fiori Elements ist durch die native Integration in das Business Application Studio als Low-Code-Entwicklungsumgebung jedoch der Quasi-Standard.

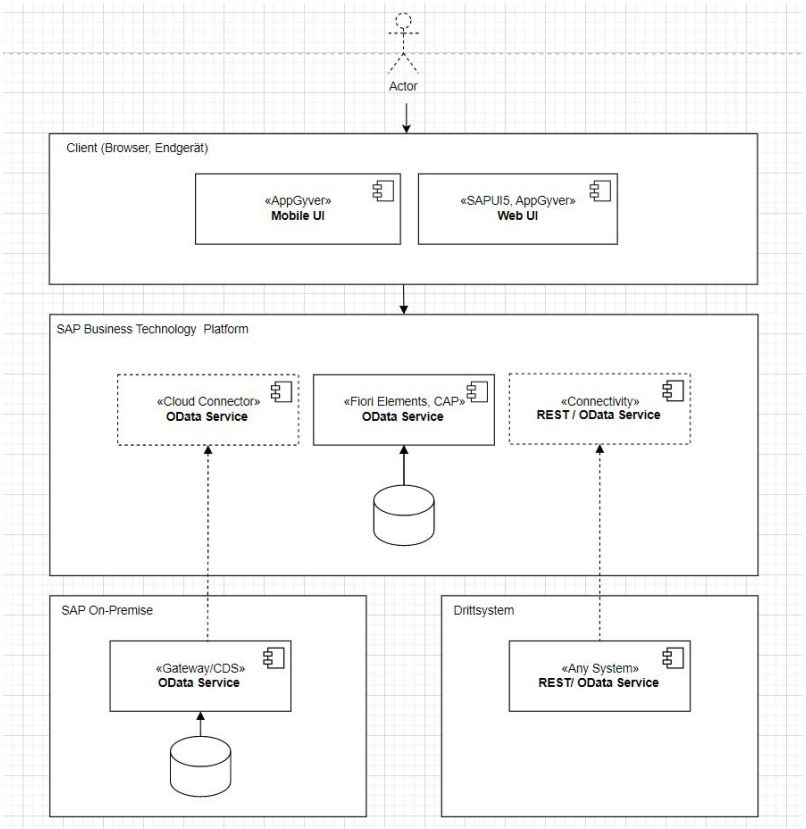

Abbildung 2.1: Architektur einer SAP-Anwendung in der SAP BTP

2.3 SAP Fiori

SAP Fiori wurde von der SAP als visuelle Leitlinie eingeführt, um User Interfaces über unterschiedliche Anwendungen hinweg zu standardisieren. Hinter dem Begriff verbergen sich jedoch ebenfalls technologische Aspekte, wie beispielsweise SAP Fiori Elements.

Das Grundkonzept von SAP Fiori ist es, Benutzeroberflächen so zu gestalten, dass Nutzer von Geschäftsanwendungen in ihrer täglichen Arbeit bestmöglich unterstützt werden, unabhängig davon, welche Endgeräte sie benutzen. Im Mit-

telpunkt stehen dabei Themen wie Usability, die Haptik und die User Experience der Anwendungen und folgende Grundsätze [Eng20, S.31]:

- Eine SAP-Fiori-App stellt dem Anwender nur die Funktionen zur Verfügung, die seiner Rolle entsprechen, sodass er nicht durch irrelevante Optionen abgelenkt wird.
- Mit SAP Fiori können Anwender sowohl auf mobilen Geräten als auch auf PCs arbeiten, wobei die Fiori-Anwendungen an das jeweilige Gerät angepasst werden müssen.
- SAP-Fiori-App statten exakt die Funktionen des Anwendungsfalles aus. Die Funktionen, die nicht für Abarbeiten des Anwendungsfalles erforderlich sind, werden nicht in die Fiori-App ausgerichtet.
- SAP Fiori folgt einer einheitlichen Interaktion Designsprache und verfügt über ein standardisiertes Oberflächendesign.
- Die wesentlichen Funktionen der Fiori-Apps sollten für den Anwender intuitive bedienbar sein. Die Fiori-Apps sollen auch ansprechend sein [Eng20, S.34-35].

Der Fokus auf spezialisierte Applikationen macht es notwendig, diese zentral bereitzustellen. Das SAP Fiori Launchpad ist deswegen der zentrale Bereich, in welchem die SAP Fiori-Anwendungen zusammengeführt werden. Es stellt den Fiori-Apps Services wie Navigation und Anwendungskonfiguration zur Verfügung. Das Launchpad ist rollenbasiert und muss anwenderspezifisch angepasst werden. Die Rolle des Anwenders definiert somit, welche Fiori-App auf den Launchpad angezeigt werden [Gui22].

SAP Fiori als Design-Richtlinie wird im weiteren Verlauf nicht weiter betrachtet. Die Grundsätze finden sich jedoch in SAP Fiori Elements und auch in SAPUI5 wieder.

2.4 SAP AppGyver

2.4.1 Grundlage von AppGyver

AppGyver ist ein Pionier in der LCNC-Entwicklung. Das Unternehmen mit Hauptsitz in Helsinki wurde im Jahr 2010 gegründet und hat seit Gründung den Fokus auf der LCNC-Entwicklung [Lea22]. Mit Composer Pro stellt AppGyver eine zentrale Entwicklungsumgebung bereit, um Anwendungen für unterschiedliche Geschäftsprozesse und Anwendungsszenarien zu entwickeln, ohne eigenen Code zu schreiben. Diese Anwendungen können nicht nur als Webanwendungen, sondern auch als mobile Anwendungen eingesetzt werden. AppGyver unterstützt

sowohl iOS mit Bereitstellung der Anwendungen im App Store als auch Android Phone mit Bereitstellung in Google Play.

Im Februar 2021 wurde AppGyver von SAP übernommen und seitdem gibt es 2 Editionen: die Community Edition und die SAP Enterprise Edition. Die Community Edition basiert auf der initialen Version von AppGyver und bleibt zunächst unabhängig von SAP. Die Benutzer können es weiterhin kostenlos nutzen. Die SAP Enterprise Edition dagegen wird in das SAP-Ökosystem integriert und ist Bestandteil der SAP BTP. Zu den zusätzlichen Funktionen der Enterprise-Version zählen:

- Integration mit der SAP BTP Authentifizierung für Webanwendungen direkt in AppGyver.
- Erweiterte Integration von Daten aus anderen SAP-Systemen.
- Neue Enterprise-Funktionen, wie beispielsweise die Einführung einer Übersetzungsvariablen.
- Nutzer können das Projekt in Echtzeit mit anderen teilen

Am 15. November 2022, während der Anfertigung dieser Arbeit, erfolgte ein Rebranding von SAP AppGyver in SAP Build Apps. Zudem wird das Tool in Zukunft Teil einer Suite an Applikationen unter dem Label SAP Build sein, welche den Fokus auf die gesamtheitliche LCNC-Entwicklung von Anwendungen, die Automatisierung von Prozessen sowie das Design von Unternehmenswebsites legt. Neben SAP Build Apps sind auch Build Process Automation und Build Work Zone in SAP Build enthalten [Lea22]. Neben der reinen Integration, wird SAP Build App zudem in Zukunft um neue Funktionen erweitert, wie beispielsweise "Visual Cloud Functions", die die Speicherung von Daten in der Cloud und die Ausführung von Geschäftslogik ermöglichen [App22b]. Diese Erweiterungen werden jedoch im Rahmen dieser Thesis nicht weiter betrachtet.

2.4.2 Entwicklungsumgebung: Composer Pro

Eine Entwicklungsumgebung ist eine Zusammenstellung von Funktionen und Werkzeugen, die zur Entwicklung einer Anwendung notwendig sind. Werden diese gesamtheitlich und zentralisiert (via Internet) bereitgestellt, dann spricht man auch von einer Entwicklungsplattform. Composer Pro ist die zentrale Entwicklungsplatform von AppGyver. Dabei handelt es sich um eine spezialisierte LCNC-Plattform, mit der Anwendungen visuell und ohne Programmierung erstellt werden können. Der Aufbau der Plattform und die Funktionen sind dabei an die Zielgruppe, Citizen Developer ohne Programmiererfahrung, angepasst. Dennoch ist ein Verständnis des grundlegenden Aufbaus der Umgebung, sowie

der Prinzipien zur Entwicklung einer Anwendung notwendig.

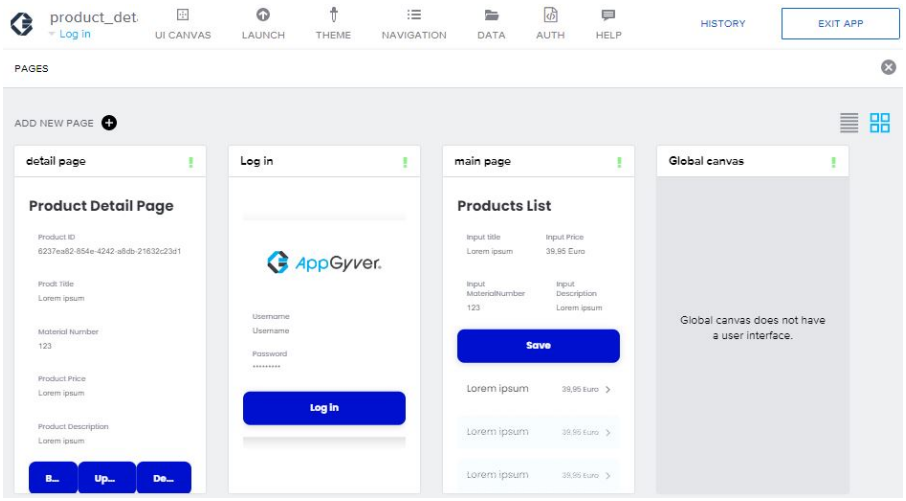

Abbildung 2.2: Pages einer AppGyver-Anwendung

Eine Anwendung in AppGyver ist grundsätzlich in mehrere Pages unterteilt. Jede Page besitzt einen eigenen Canvas, auf dem weitere Inhalte platziert werden können. Am oberen Rand der Benutzeroberfläche befindet sich die zentrale Toolbar, über die der Entwickler auf alle Ressourcen und Werkzeuge von AppGyver zugreifen kann.

Abbildung 2.3: Toolbar in Composer Pro

Dem LCNC-Ansatz folgend, finden sich in AppGyver keine programmatischen Bausteine (Code-Files, Klassen, etc.), sondern die diversen Funktionalitäten sind in eigenen, proprietären, Strukturen abgelegt. Dabei lassen sich diese grob in die Schichten des MVC-Models einteilen. Das MVC-Paradigma strukturiert die Implementierung einer Anwendung in folgende drei Schichten:

- **M** steht für Model und repräsentiert das Datenmodell. Das Datenmodell stellt die relevanten Daten bereit.
- **V** bezieht sich auf View, d.h. die Präsentation. Diese Schicht ist zuständig für die Darstellung auf den Endgeräten und die Realisierung der Benutzerinteraktionen.
- **C** steht für Controller, also die Steuerung. Controller steuern und verwalten die Views. Der Controller kommuniziert mit dem Modell, wenn eine Benutzeraktion mit einer Datenänderung stattfindet.

Applikations-Schicht	AppGyver-Baustein
View	Page; View Component; Properties; Theme
Controller	Logic Flows; Formula Functions
Model	(Data) Variables; Data Resource

Tabelle 2.1: AppGyver-Baustein in die Schichten des MVC-Models

View

Eine Anwendung in SAP AppGyver besteht aus mehreren Pages, d.h. mehreren eigenen Sichten. Diese werden aus vorgefertigten und konfigurierten View Components zusammengesetzt. Der Komponentenbibliothek in Composer Pro bietet einen Überblick über alle verfügbaren Komponenten. Diese sind in drei Registerkarten unterteilt. Unter CORE sind die Kernkomponenten verfügbar, die in den meisten Anwendungen verwendet werden. Dies sind beispielsweise Texte, Buttons oder Input-Felder. Unter BY ME sind die Komponenten aufgelistet, die der Entwickler selbst für diese Anwendung erstellt hat. Komponenten, die aus dem Marketplace für diese Anwendung hinzugefügt wurden, sind auf der Registerkarte *INSTALLED* zu finden [App22c]. Jede View Component besitzt spezifisch Eigenschaften (Properties), die sich in dem kontext-sensitiven Panels „Component Properties “ und „Style “ angepasst werden können. Der Layout-Tree, der sich unten in der rechten Seitenleiste befindet, zeigt die komplette Struktur der Komponenten in der Anwendung an und ermöglicht die direkte Auswahl. Weiterhin ist es möglich, einen grundlegenden Theme mit Styles bereitzustellen, der die grundlegenden Farben, Schriften, etc. für die Anwendung festlegt.

Model

AppGyver unterstützt in Bezug auf die Datenintegration zwei Szenarien: Es können Daten lokal im Projekt angelegt und abgespeichert werden. Diese Daten sind dann jedoch nur für die lokale Applikation gültig. Alternativ kann auf externe Datenendpunkte (REST, OData) zugegriffen werden. Die Daten werden dann über die externen Schnittstellen abgefragt und können in der Anwendung angezeigt werden. Zur Abbildung von Datenstrukturen und -flüssen gibt es in Composer Pro das Konzept der Variablen. Damit können ohne Programmierung verschiedene Arten von Strukturen festgelegt werden. „Page Variablen “ existieren nur für die aktuelle Seite und enthalten beliebige Werte, während „App Variablen “ über die

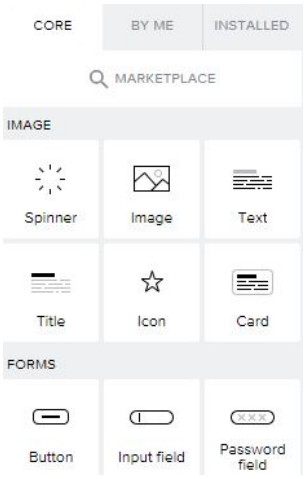

Abbildung 2.4: View-Components in Composer Pro

gesamte Anwendung hinweg existieren. „Data Variablen “ befinden sich ebenfalls nur auf der aktuellen Seite und beinhalten die Funktion, interne und externe Datenstrukturen zu mappen. Diese Variablen können an die spezifischen Eigenschaften (Properties) einer View Component gebunden werden (Data-Binding) und stellen somit das Bindeglied zwischen Model und View dar.

Weitere Variablen zur Ablage von Werten sind „Page Parameter “ (schreibgeschützte Textvariablen), die zur Übertragung von Daten zwischen den Seiten verwendet werden können und „Translation Variablen “, die für sprachabhängige Texte verwendet werden können.

Controller

Neben den Daten ist auch die Geschäftslogik ohne oder mit geringen Programmierkenntnissen umsetzbar. SAP AppGyver stellt hier Logische Ablauffunktionen bereit, mit denen per Drag&Drop, sowie einer Konfiguration, komplexere Prozesse definiert werden können. Der Logic-Canvas befindet sich in der unteren Hälfte der Benutzeroberfläche. Für jede Komponente gibt es einen eigenen Logic-Canvas-Kontext, der eine spezifische Konfiguration erlaubt. Damit auf Nutzereingaben reagiert werden kann, stellen die View Components Events zur Verfügung. Diese werden immer dann geworfen, wenn eine spezifische Interaktion auftritt.

Weiterhin existieren in AppGyver Formeln, die dazu genutzt werden können, um komplexere Logiken abzubilden. Dazu gibt es einen eigenen Formel-Editor, in dem die Logiken hinterlegt werden können. Tatsächlich stehen

dort Formel-Typen zur Verfügung, die sich nahe an der richtigen Programmierung bewegen (Logische Operatoren, IF-Statements, etc) [App22a].

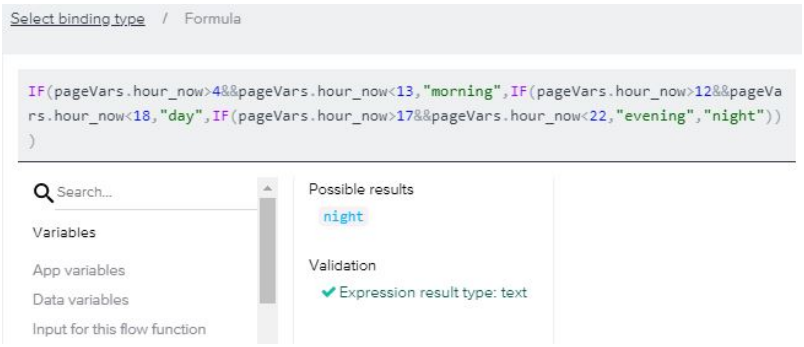

Abbildung 2.5: Exemplarische Formel in AppGyver

2.5 SAPUI5

2.5.1 Grundlage von SAPUI5

SAP Fiori beschreibt als Leitfaden die Entwicklungsrichtlinie zur Erstellung von Anwendungen im SAP-Umfeld. Für die Umsetzung sind jedoch auch technische Bausteine notwendig. Mit SAP WebDynpro, der vormals führenden UI-Technologie, ist SAP Fiori aber schwer umzusetzen, da der HTML-Code auf dem Server generiert und dann an das Endgerät übermittelt wird. In diesen Kontext ist die Entwicklung eines clientseitigen Ansatzes erforderlich. Bei einem clientseitigen Ansatz steht der Frontend-Server im Mittelpunkt der Kommunikation und lädt das UI-Framework, das die weiteren Verarbeitungsschritte übernimmt [Eng20, S.45-47].

SAPUI5 (kurz für: SAP UI Development Toolkit für HTML5) ist ein JavaScript-basiertes clientseitiges Framework und eine UI-Bibliothek, mit der SAP Fiori-Anwendungen sehr flexibel entwickelt und auf verschiedene Plattformen portiert werden können. Die SAPUI5-Bibliothek basiert auf modernen Web-Standards, wie JavaScript, HTML5 und CSS3 [Ant16, S.139]. SAPUI5 verfügt über mehr als 500 UI-Controls, die an den neuesten SAP Fiori Design Guidelines ausgerichtet sind, und bietet integrierte Unterstützung für Enterprise-Funktionen, wie Datenbindung, Routing, Message Handling, Mehrsprachigkeit, etc. Heute ist SAPUI5 der Standard für die Implementierung von Frontend-Anwendungen im SAP-Umfeld [Com22a].

Da es sich bei SAPUI5 nicht um eine LCNC-Technologie handelt, ist ein grundlegendes Programmierverständnis in der Sprache JavaScript notwendig. Zudem setzt SAPUI5 das Verständnis einiger Entwurfsmuster voraus. Das MVC-Paradigma beispielsweise strukturiert die Implementierung von SAPUI5-Anwendungen in drei Schichten, die sich so auch im Quellcode wiederfinden.

- **Model:** Hier stehen spezielle Klassen zur Verfügung, die den Zugriff auf unterschiedliche Arten von Daten abstrahieren (JSONModel, ODataModel, XMLModel).
- **View:** Die View beinhaltet den Aufbau des User Interfaces und SAPUI5 stellt hier UI Controls zur Verfügung, dafür genutzt werden. Diese lassen sich via JavaScript, HTML oder XML definieren und konfigurieren.
- **Controller:** Zu jeder View gibt es in SAPUI5 einen Controller, der eine Benutzeraktion über Events und zugehörige Callback-Implementierungen steuert, sowie komplexere Geschäftslogik enthalten kann [Ant16, S.149].

Da es sich bei SAPUI5 um ein komplexeres Framework handelt, können an dieser Stelle nicht alle Facetten im Detail erklärt und beschrieben werden. In Kapitel 3.3 werden ihm Rahmen der Implementierung jedoch einige genutzte Dinge vorgestellt.

2.5.2 Entwicklungsumgebung: Visual Studio Code

Durch den freien Programmieransatz ist SAPUI5 nicht an eine Entwicklungsumgebung/-plattform gebunden. Im Rahmen dieser Thesis wird Visual Studio Code (kurz: VS-Code) für die Umsetzung verwendet. VS-Code ist ein Quellcode-Editor von Microsoft, der im Jahr 2015 für verschiedene Betriebssysteme wie Windows, MacOS und Linux veröffentlicht wurde. Er unterstützt diverse Programmiersprachen und Frameworks (z.B. JavaScript, TypeScript und Node.js), kann jedoch über das Erweiterungskonzept um weitere Programmiersprachen, Laufzeiten und Funktionen ergänzt werden [Cod22a]. VS-Code ist heute aufgrund der ausgereiften Funktionen und der guten Bedienung ein Standard in der Entwicklung von Web-Anwendungen [Wik22c] und wird deswegen an dieser Stelle potenziellen anderen Entwicklungsumgebungen vorgezogen.

Der Arbeitsbereich des VS Code besteht aus ein oder mehreren Verzeich-

nissen. Es vereinfacht die sprachübergreifende Anwendungsentwicklung in einer integrierten Entwicklungsumgebung. VS Code integriert ein voll funktionsfähiges Terminal mit dem Editor, um Shell Skript und Kommenden durchzuführen. Das integrierte Terminal kann verschiedene Shells verwenden, die auf dem Rechner installiert sind [Cod22b].

Die lokale Implementierung der SAPUI5-Anwendung in VS-Code erfordert die zusätzliche Installation von weiteren Bibliotheken: das SAP Cloud Application Programming Model (kurz: SAP CAP) und des Easy UI5 Generators. Das SAP CAP ist ein Framework zur Erstellung von Backend-Anwendungen und kommt auch bei Fiori Elements zum Einsatz [SAP22a]. Mit Hilfe von Core Data Services als die universelle Modellierungssprache für Domänenmodelle und Servicedefinitionen, können in sehr kurzer Zeit Datenmodelle generiert und als OData-Service bereitgestellt werden [SAP22a]. Der Easy UI5 Generator enthält Vorlagen zur Erstellung einer SAPUI5-Anwendung mit aktuellen Best Practices. So lassen sich bereits die grundlegenden Strukturen einer Anwendung erstellen, auf deren Basis die Entwicklung dann stattfinden kann [SAP22c]

Um die UI5-Anwendung zu implementieren, muss die Entwicklungsumgebung wie folgt eingerichtet werden:

- Herunterladen und Installieren des aktuellen Node.js Installationspakets von https://nodejs.org/en/download/ inklusive Runtime und Package Manager (npm). Node.js bietet eine asynchrone, ereignisgesteuerte Laufzeitumgebung für Javascript und wird für das lokale Betreiben eines Webservers verwendet [Wik22a].
- Installieren von yeoman und dem Easy-ui5 Generator. Yeoman ist ein Kommandozeilen-Scaffolding-Tool für Node.js, um das Gerüst für die weitere Entwicklung der SAPUI5-Anwendung zu generieren
- Installieren der SAP Cloud Application Programming Model-Module (@sap/cds-dk und @sap/cds).
- Herunterladen und Installieren der SQLite-Datenbank-Treiber. Als ein leicht eingebettetes Datenbanksystem, lässt sich SQLite direkt in entsprechende Anwendungen integrieren, ohne keine weitere Server-Software [Wik22b].
- Installation der VS Code Erweiterungen für SAP Fiori und SAP CAP CDS, zur Unterstützung bei der Entwicklung.

Das cds-dk und SQLite werden benötigt, um bei der lokalen Entwicklung

einen OData Mock Service zu erzeugen. Die VSCode-Erweiterungen bieten Sprachunterstützung für die Core Data Services (CDS), welche die Grundlage des SAP Cloud Application Programming Model (CAP) bilden, sowie für SAPUI5 [Mar22].

2.6 Fiori Elements

2.6.1 Grundlage von Fiori Elements

SAP Fiori als gestalterische Richtlinie lässt sich mit SAPUI5 technisch umsetzen. Allerdings ist dort immer Programmieraufwand nötig. Als LCNC-Ansatz stellt die SAP jedoch mit Fiori Elements eine Technologie zur Verfügung, mit der UI Controls und sogar komplette Applikationen automatisch auf Basis von Metadaten zur Laufzeit generiert werden können. Die Metadaten werden dabei via OData-Annotationen beschrieben, die vom Backend-Service bereitgestellt werden müssen [Eng20, S.48]. SAP Fiori Elements basiert technologisch auf SAPUI5 und erweitert dieses um intelligente Komponenten und Views. SAP Fiori Elements umfasst sogenannte Floorplans, d.h. Grundrisse und UI-Patterns für gängige Anwendungsfälle. Fiori Elements verfügt über die folgenden vier grundlegenden Floorplans:

- List Report und Object Page

Ein List Report wird verwendet, wenn Elemente in einer Tabelle oder Liste dargestellt werden sollen. Mit dem List Report können die Objekte angezeigt, gefiltert und bearbeitet werden. Der List Report wird in der Regel in Verbindung mit der Objekt Page verwendet. Auf die Objekt Page kann der Nutzer durch Klicken auf ein einzelnes Element in der Liste gelangen und dort detaillierte Informationen über einzelne Objekte angezeigt bekommen und es bearbeiten.

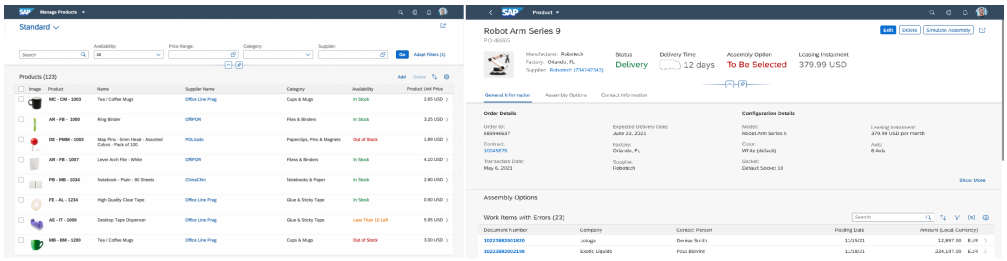

Abbildung 2.6: List Report und Object Page (Quelle: SAP Fiori Design Guidelines. URL: https://experience.sap.com/fiori-design-web/smart-templates)

- Worklist

Eine Worklist zeigt ebenfalls eine Liste von Elementen an, die von Benutzer bearbeitet werden sollen. In einer Worklist ist jedoch keine komplexe Filterung möglich und die Anzeige, bzw. das Editieren erfolgt ausschließlich in der Worklist-Ansicht [Doc22b].

- Overview Page

Ein Overview Page ermöglicht es, den Nutzern eine große Menge unterschiedlicher Informationen für einen Überblick bereitzustellen. Unterschiedliche Informationen werden in verschiedenen Cards visualisiert. Eine Card zeigt die Details zu einem bestimmten Geschäftsobjekt. Mit der Overview Page können Anzeigen, Filtern und Verarbeiten von Daten einfach und effizient gemacht werden.

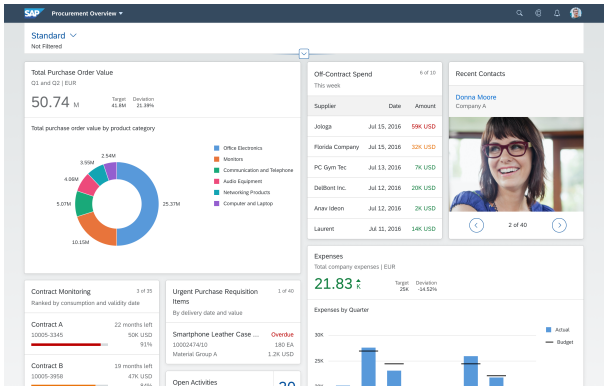

Abbildung 2.7: Overview Page (Quelle: SAP Fiori Design Guidelines. URL: https://experience.sap.com/fiori-design-web/smart-templates)

- Analytical List Page

Die Analytical List Page ist die Weiterentwicklung von List Report und verfügt über mehr Funktionen: Daten können in verschiedene Perspektiven analysiert werden, z.B. durch Drill-Downs zur Ursachenforschung [Doc22b].

Mit Hilfe von Extension Points in den Floor-Plans können in einer Fiori Element-Anwendung zusätzliche Funktionalitäten hinzugefügt werden. Dafür ist jedoch immer eine Programmierung erforderlich. SAP Fiori Elements ohne Extensions benötigt keinen Programmieraufwand, setzt jedoch einen OData-Service und eine Konfiguration der Annotationen voraus. Hierfür stehen potentiell unterschiedliche Technologien zur Verfügung. Dank der SAP-eigenen Entwicklungsumgebung, SAP Business Application Studio,

können jedoch komplette SAP Fiori Elements-Anwendungen inklusive der Defintion von Datenstrukturen und OData-Services und Annotationen visuell aufgebaut werden [Doc22a].

2.6.2 Entwicklungsumgebung: Business Application Studio

Grundsätzlich kann SAP Fiori Elements mit verschiedenen Entwicklungsumgebungen erstellt werden. Durch die sehr gute Integration und Bereitstellung einer LCNC-Umgebung, eignet sich das SAP Business Application Studio (BAS) jedoch sehr gut für das Erstellen einer Anwendung im Kontext dieser Thesis. BAS ist ein Service der SAP Business Technology Platform (SAP BTP), der seit Februar 2020 als Nachfolger der SAP Web IDE zur Verfügung steht. SAP BAS ist ein Cloud-basiertes Werkzeug, das nicht lokal installiert werden kann und im Browser des Nutzers ausgeführt wird und entspricht damit den Kriterien einer Entwicklungsplattform [Por22b].

Das Business Application Studio lässt sich via Dev Spaces für unterschiedliche Anwendungsarten konfigurieren. Die Entwicklungsszenarien umfassen zum Beispiel SAP Fiori, SAP S/4HANA Erweiterungen, Workflows und SAP HANA-Anwendungen. In jedem Dev Space werden je nach Auswahl eine Reihe von vordefinierten Erweiterungen installiert, die spezialisierte Funktionen bereitstellen [Por22b].

Neben der klassischen Code-basierten Entwicklung, stellt BAS auch die Möglichkeit bereit, Low-Code basierte „Full-Stack Anwendungen “ zu entwickeln. Dafür sind dann diverse Wizards und UI-Masken vorhanden, die im Hintergrund automatisch den notwendigen Quellcode interpretieren und erstellen. Eine Full-Stack-Anwendung besteht dabei aus dem Datenmodell und OData-Service des SAP Cloud Application Programming Models und SAP Fiori Elements als Frontend-Technologie. Alle notwendigen Parameter lassen sich via UI konfigurieren.

Im Rahmen dieser Arbeit wird ein Dev Space erstellt, der als Entwicklungsumgebung für die Low-Code-basierte Full-Stack-Anwendung ausgewählt wurde, um den in Kapitel 1, Abschnitt 1.3 beschriebenen Anwendungsfall mit Fiori-Elements zu entwickeln.

Kapitel 3

Implementierung des Anwendungsfalls

In Abschnitt 1.3 wurden die Anforderungen an die zu implementierende Anwendung bereits vorgestellt, in diesem Kapitel wird nun die Umsetzung in den einzelnen Technologien (Fiori Elements in Kombination mit SAP CAP, AppGyver und SAPUI5) näher beschrieben.

Technologie	Frontend	Backend
AppGyver	X	-
SAPUI5	X	-
Fiori Elements (inkl. SAP CAP)	X	X

Tabelle 3.1: Umsetzungsanforderung der Technologie

Dabei ist zu unterscheiden zwischen Backend- und Frontend-Funktionalität. SAPUI5 verfügt grundsätzlich über keine Backend-Funktionalität. Mit SAP AppGyver können lokale Datenstrukturen erzeugt werden. Diese Funktion wird jedoch nicht genutzt, sondern ein zentraler OData-Service inklusive Datenbackend an anderer Stelle bereitgestellt. Hierfür wird auf die „Full Stack “-Anwendungen, bestehend aus Fiori Elements und SAP CAP, im SAP Application Studio zurückgegriffen. Das Fiori Elements-Frontend wird den erstellten OData-Service dann konsumieren. Das gleiche Datenmodell und der gleiche OData-Service werden dann ebenfalls von SAP AppGyver und SAPUI5 verwendet.

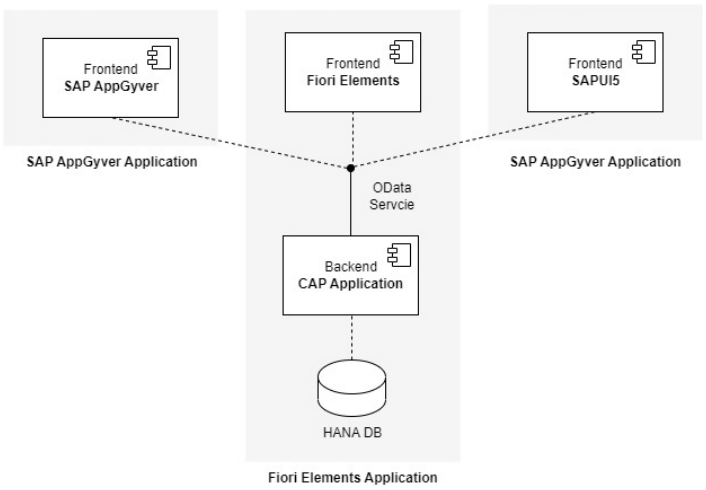

Abbildung 3.1: Umsetzungsarchitektur der Technologie

3.1 Fiori Elements

3.1.1 Dev Space erstellen

Für die Entwicklung mit Fiori Elements wird das Business Application Studio als Entwicklungsumgebung genutzt. Wie bereits in Abschnitt 2.4.2 erwähnt, stellt das BAS Dev Spaces für unterschiedliche Entwicklungsszenarien bereit. Die komplette Anwendung ließe sich auch als klassisches Entwicklungsprojekt auf- und umsetzen, im Sinne der Arbeit wird jedoch das "Low-Code-Based Full-Stack Cloud-Application" Preset für den Dev Space gewählt, um Zugriff auf die LCNC-Funktionalitäten zu bekommen.

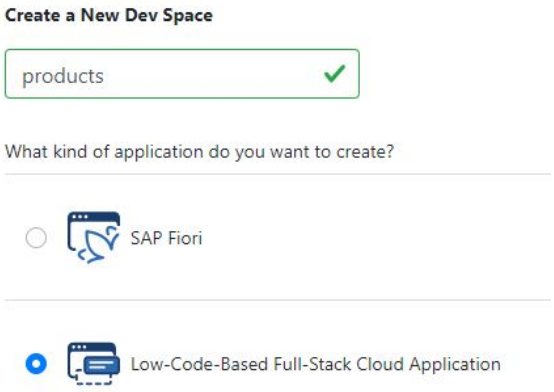

Abbildung 3.2: Create a New Dev Space

Nach der Erstellung des Dev Space wird die Entwicklungsumgebung für eine Low-Code-basierte Full-Stack Cloud-Anwendung bereitgestellt. In

dieser Umgebung stehen verschiedene „Cards “ zur Verfügung, um den auf Low-Code basierenden Implementierungsprozess durchzuführen.

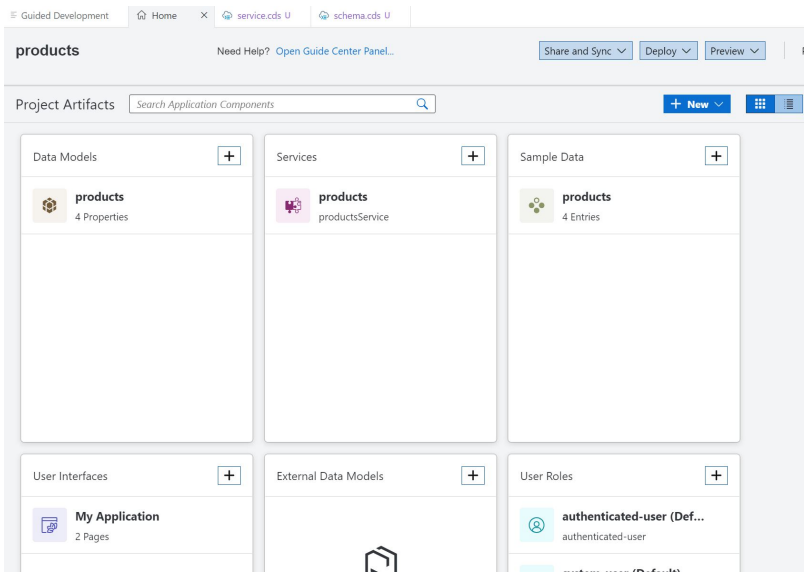

Abbildung 3.3: Grundlegenden Aufbau inkl. der Cards von BAS

3.1.2 Datenentitäten und Service definieren

Für den Anwendungsfall wird eine Datenentität mit sechs Properties festgelegt. Neue Entitäten können einfach in der Datenmodellkarte hinzugefügt und konfiguriert werden. Der Entitätsname wird als „Products “ definiert und die sechs Properties sind ID, Title, MaterialNumber, Description, Price und Stock. Um die Implementierung des Anwendungsfalls zu erleichtern, wird die ID hier als UUID und alle anderen Properties als String definiert.

Abbildung 3.4: Datenmodell

Beim Anlegen der Entität über die UI wird im Hintergrund eine Code-Datei erzeugt (model.cds) und die Definition dort abgelegt. Dies ist im LCNC-

Modus für den Entwickler nicht sichtbar und er muss sich um das Coding nicht weiter kümmern. Grundsätzlich ist es aber möglich, das Coding manuell anzupassen und die UI reagiert entsprechend auf die Änderungen im Coding. Es ist jedoch herauszustellen, dass die Wizards und UI-Masken von BAS nicht alle Funktionalitäten von SAP CAP unterstützen und so spezielle Dinge ggf. nur über das Coding hinzugefügt werden können. Für den gewählten Use-Case ist dies jedoch nicht von Belang.

Angelegte Datenbank-Entitäten können im Weiteren einfach als OData-Service exponiert werden. Ähnlich wie beim Datenmodell, gibt es dafür eine eigene Karte. Dort wird lediglich die Entität ausgewählt, Name, Namespace und Typ festgelegt, sowie einige weitere Properties konfiguriert und dann kann der Service erstellt werden.

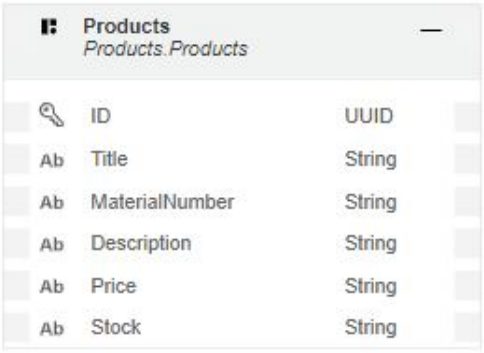

Abbildung 3.5: Service

Unter der Sample-Data-Karte können Mock-Daten hinterlegt werden, damit man während der Entwicklung direkt Zugriff auf Daten hat. Da die Property-ID als UUID definiert ist, werden die IDs automatisch vom System vergeben. Die anderen Properties wie Title, MaterialNumber, Description, Price und Stock müssen manuell eingetragen werden.

Tatsächlich hat man durch die Verwendung der drei Cards mit diesen wenigen Klicks und Eingaben in sehr kurzer Zeit ein eigenes Datenmodell und einen OData-Service definiert, den man in der Preview im BAS auch direkt aufrufen und testen kann.

3.1.3 User Interface erstellen

Die Anforderungen an die Benutzeroberfläche bestehen darin, dass eine Listenansicht für die Anzeige aller Produkte und eine Einzelansicht für ein

Products(15) Mock Data:

		ID*	TITLE	MATERIALNUMBER	DESCRIPTION	PRICE	STOCK
☐	1	0165cdd0-f1aa-4c62...	Carpendo car seat co...	252	Black front seats and...	40,19 Euro	STOCK2786
☐	2	f3e89493-d10a-4d32...	Carpendo Auto Spor...	338	Seat covers black-gr...	46,99 Euro	STOCK5817
☐	3	b1200d4e-e514-47ef...	Woltu car seat covers	111	Black, with "Super D...	26,99 Euro	STOCK9274
☐	4	e209033f-5993-4113...	Walser Comfort Car ...	760	Car seat cover S-Rac...	16,95 Euro	STOCK256
☐	5	c27cb5cb-3ad2-42ec...	Walser car seat cover...	949	Universal seat cover ...	74,95 Euro	STOCK4803
☐	6	f798ef12-f5a2-480b-...	Flying Banner Car Se...	872	Universal Set with Ai...	29,99 Euro	STOCK3350
☐	7	87de15d1-221b-432f...	Flying Banner PVC C...	848	Complete set with ai...	34,39 Euro	STOCK5088
☐	8	a31f73e0-be2c-47d3...	Flying Banner Univer...	757	Complete Cover Brea...	34,99 Euuo	STOCK5150
☐	9	0094daed-09f7-4557...	Upgrade4Cars car se...	641	Seat covers complet...	39,95 Euro	STOCK961
☐	10	071d0a34-c4e4-427...	Upgrade4Cars car se...	110	Car seat cover for sp...	13,95 Euro	STOCK778
☐	11	31901267-a58f-49b4...	Fixcape PRO car seat...	904	Car seat cover for fro...	15,90 Euro	STOCK4737
☐	12	22407997-be2a-40b...	Fixcape Neoprene C...	919	Car seat cover for th...	29,50 Euro	STOCK1005
☐	13	658b32a1-2403-4a6...	Leader Accessories u...	567	Imitation Leather Car...	24,99 Euro	STOCK9967

Abbildung 3.6: Sample Data

einzelnes Produkt, sowie eine Maske für die Pflege jedes einzelnen Produkts entworfen werden sollen. SAP Fiori Elements stellt hierfür bereits komplett vorgefertigte Floorplans zur Verfügung, die nahtlos in BAS integriert sind. Unter der User-Interfaces-Karte kann die Benutzeroberfläche der Anwendung definiert werden. Dies erfolgt in vier Schritten:

1. Eingabe der Details der UI-Anwendung wie Name und Namespace.

2. Auswahl des Anwendungstyps. In dem hier vorgestellten Anwendungsfall wird eine *Template-Based, Responsive Application* ausgesucht. Es wäre jedoch auch möglich, eine Freestyle SAPUI5-Anwendung zu erstellen.

3. Wie in Abschnitt 2.3.3 erwähnt, werden von SAP verschiedene Arten von Floorplans für Fiori Elements bereitgestellt. In diesem Fall wird *List Report Object Page* für UI Application Template verwendet.

4. Der letzte Schritt bei der Erstellung einer UI-Anwendung ist die Auswahl des richtigen Datenobjekts. Hier wird die Datenentität *Products* gewählt und es werden automatisch Tabellenspalten zur Listenseite und ein Abschnitt zur Objektseite eingeschaltet.

Nach dem Klick auf den „Finish “-Button wird die UI-Application automatisch generiert. Auch hier wird im Hintergrund Quellcode abgelegt: Zum einen werden die OData-Annotationen in weiteren cds-Dateien abgelegt. Des Weiteren wird das Grundgerüst einer SAPUI5-Anwendung angelegt und mit der entsprechenden Konfiguration für SAP Fiori Elements ver-

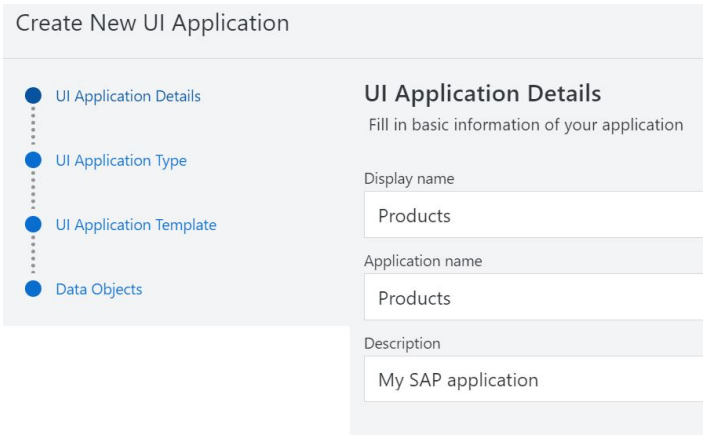

Abbildung 3.7: UI-Konfiguration

sehen. Auch dies ist für den Entwickler nicht direkt in BAS einsehbar, bzw. es ist nicht notwendig sich mit dem Code auseinander zu setzen, es sei denn, man möchte gezielt Funktionen nutzen, die von den Wizards in BAS nicht unterstützt werden. Die finale Anwendung besteht aus einer Listenseite und einer Detailseite für jedes Produkt. Die Listenseite zeigt eine Liste aller Produkte. Wird auf ein Listenelement geklickt, so wird die Detailseite für das jeweilige Produkt geöffnet. Zudem kann ein neues Produkt hinzugefügt, ein Produkt gelöscht oder bearbeitet werden. Während man über die Konfiguration einen Einfluss auf die Tabellenspalten und Inhalte der Einzelansicht nehmen kann, funktionieren Dinge wie Navigation, Daten laden, Datenbindung und auch das Speichern von neuen Datensätzen einfach so, ohne, dass vom Entwickler irgendeine komplexere Konfiguration vorgenommen werden muss. Allerdings kann man von den dem Standardverhalten auch nicht abweichen und jede Anwendung folgt dem grundlegenden Aufbau und Verhalten des ausgewählten Floorplans. Mithilfe der Page Map in BAS kann die UI-Anwendung angepasst werden. Zum Beispiel können für die hier vorgestellte Anwendung das initiale Laden aktiviert werden und auch die Properties, die in der Liste angezeigt werden, können je nach Anforderung geändert werden. Dabei sind alle Parameter über UI-Masken konfigurierbar und an keiner Stelle muss ein eigenes Coding erfolgen.

3.1.4 Deployment des OData-Services

Um auf den OData-Service auch aus den anderen Applikationen zuzugreifen und die UI-Anwendung bereitzustellen, muss ein Deployment aus

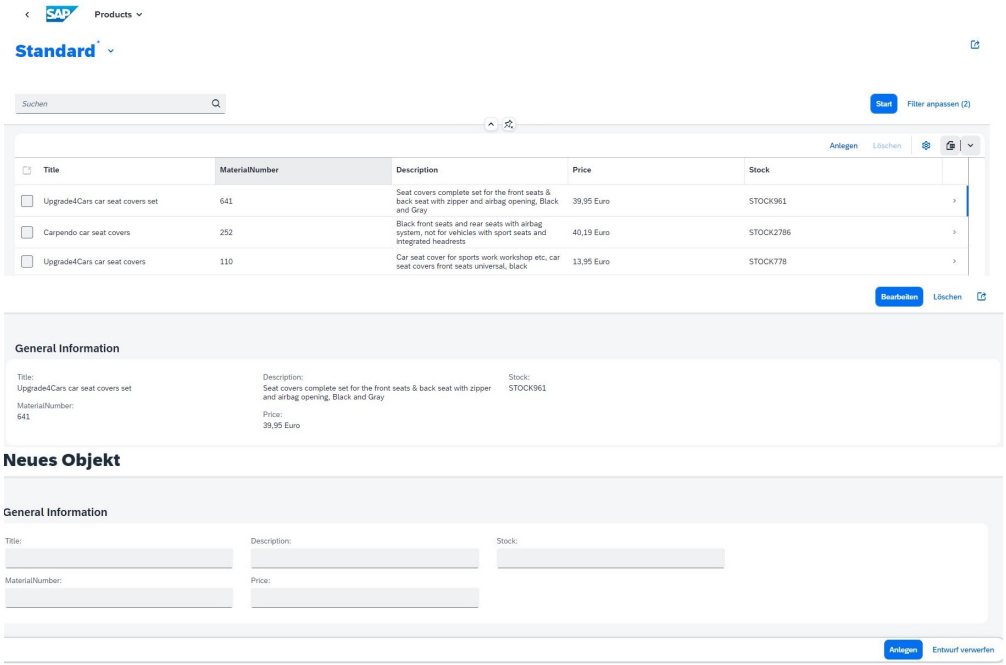

Abbildung 3.8: Listenseite, Detailseite und „neues Objekt anlegen- Seite der Fiori-Elements-Anwendung

BAS heraus erfolgen. Das Deployment erfolgt direkt auf die SAP BTP. Hierfür muss ein sogenannter „Space “ auf der SAP Business Technology Plattform vorliegen, sowie eine SAP HANA Cloud-Instanz bereitgestellt werden. Nachdem die Anwendungsbenutzeroberfläche in Business Application Studio erstellt worden ist, kann diese direkt aus BAS heraus deployed werden, wiederum ohne notwendiges Coding. Dazu muss man in BAS bei Cloud Foundry anmelden und das Cloud Foundry-Ziel angeben. Abbildung 3.9 zeigt, wie dies geht.

Danach lässt sich das Deployment starten und es werden während des Vorgangs alle notwendigen Komponenten (Applikation und notwendige BTP-Servie-Instanzen) erstellt. Nach dem Deployment stehen OData-Service, sowie UI-Applikation dann unter einer URL zur Verfügung. Auch das Deployment erfolgt vollständig als No-Code-Ansatz. Die notwendige Konfigurationsdatei (mta.yml) wird im Hintergrund des Projekts abgelegt und beim Deployment ausgelesen. Während der Implementierung gab es jedoch gelegentlich Probleme bei dem Deployment, sodass eine tiefergehende Analyse erforderlich wurde. Insbesondere in solchen Fällen musste die LCNC-Umgebung verlassen werden, damit in den Log-Dateien der Fehler gefunden werden konnte. Zudem waren die Probleme meist sehr technischer

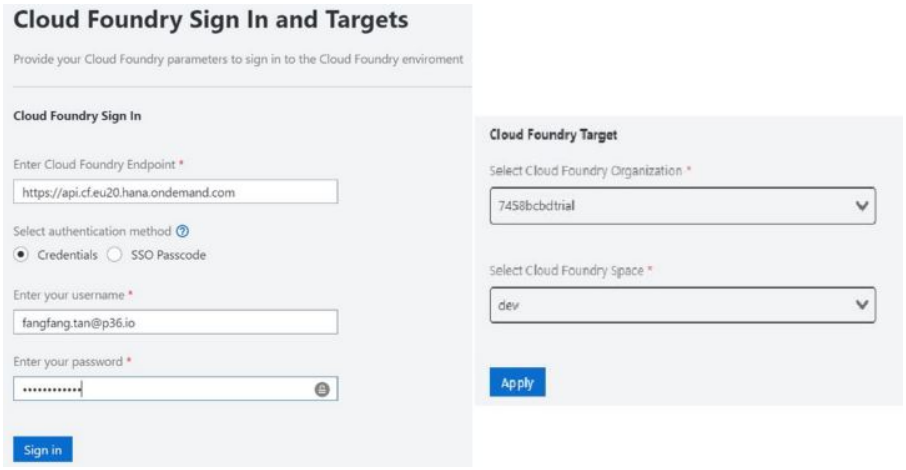

Abbildung 3.9: Cloud Foundry Sign In und Targets

Natur und erforderten ein tiefgreifenderes Verständnis der verwendeten Technologien (SAP CAP, SAP BTP).

3.2 AppGyver

3.2.1 Projektaufbau

Für die Umsetzung des Projekts wird SAP AppGyver Enterprise auf der SAP BTP verwendet. Die notwendigen Services können einfach in der SAP BTP aktiviert werden und nach Zuweisung der entsprechenden Rollen steht die Composer Pro-Entwicklungsplattform bereit.

3.2.2 OData Integration

Seit dem Kauf durch SAP wird SAP AppGyver immer mehr in das Ökosystem der SAP integriert. So ist es in der Enterprise-Edition mittlerweile möglich, Funktionen der SAP BTP zu nutzen, um Daten in AppGyver bereitzustellen. Um die Daten des OData-Services in AppGyver nutzen zu können, muss eine „Destination “ im SAP Business Technology Plattform Cockpit angelegt werden. Der SAP BTP Connectivity Service stellt via Destinations Reverse Proxy- und Authentifizierungsfunktionen zur Verfügung, sodass unter Verwendung der Destination ein einfacherer Zugriff auf einen Remote-Endpunkt erfolgen kann. Die Verbindungsdetails können via Eingabemaske gepflegt werden [Por22a]. Anzugeben sind Name, eine URL, Authentifizierungsdetails sowie weitere spezifische Konfigurationsparameter. Die Destination wird hier *products-on-trial_simple* genannt. Die URL

bezieht sich auf die OData-Service-Adresse. Die Authentifizierung wird dann als „OAuth Client Credentials Authentication “ definiert. Wichtig ist, dass die „Additional Properties “ gesetzt werden: *AppgyverEnabled* sowie *HTML5.DynamicDestination* müssen auf *true* gesetzt werden, damit der OData-Service in AppGyver eingebunden werden kann.

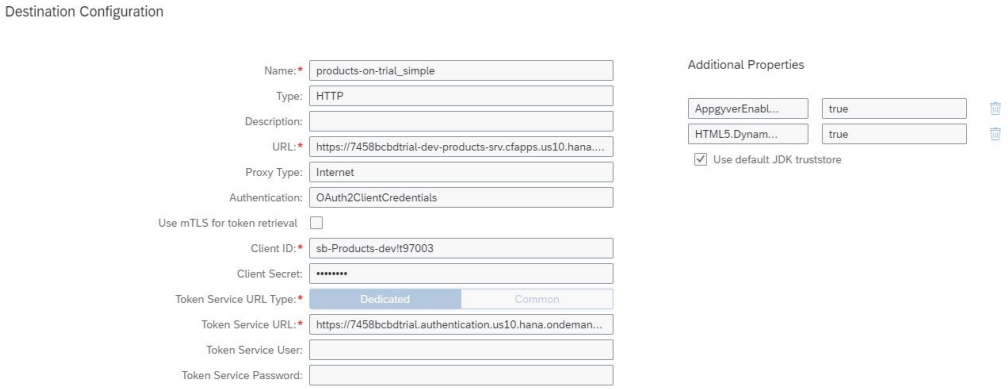

Abbildung 3.10: Destination-Konfiguration in SAP BTP Cockpit

Nachdem die Destination erstellt wurde, ist der Service *products-on-trial_simple* in SAP AppGyver unter dem Menüpunkt Data – SAP Systems verfügbar und kann importiert werden. AppGyver verfügt über eine Funktionalität zum Auslesen der Metadaten des OData-Services und listet die vorhandenen Entitäten in der UI auf. Die Entität „Products “ kann dann einfach aktiviert werden und steht dann in der Anwendung zur Verfügung. Bereits im Integrations-Wizard lassen sich Dateninhalte und Werte des OData-Services ansehen, bzw. sogar verändern.

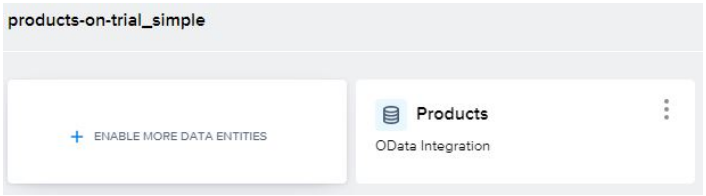

Abbildung 3.11: OData-Integration in AppGyver

3.2.3 Listenansicht zur Anzeige aller Produkte

Um eine Liste aller Produkte anzeigen zu können, muss zunächst eine Listenseite mit dem Namen „The List of Products “ erstellt werden. Diese Seite besteht aus vier Elementen: Einem Titel mit dem Text „Products “, einem Button zum Erstellen eines neuen Produkt-Datensatzes und einem

weiteren Button zum Einblenden eines Suchfeldes, sowie einem List Item. Die Erstellung eines Suchfeldes wird in Kapital 4 näher betrachtet werden. Eine Data-Variable mit dem Namen „Product1 “ wird erstellt, um die Daten von OData-Service abzurufen. Der Typ der Data-Variablen ist auf „Collection of data records “ eingestellt, wodurch eine Sammlung von Datensätzen geliefert wird. Die Data-Variable verfügt über eine Default-Logik, die bestimmt, wie sie sich mit Daten aus dem Backend füllt. Die Daten werden aus dem Backend über den Ablauffunktionskomponente „Get record collection “ bezogen, wobei die Daten hier nur als ein Ausgangsargument des Knotens existieren. Dann wird den „Set data variable “-Ablauffunktionskomponente verwendet, damit die Daten in die Data-Variable setzen können. Nun steht die Data-Variable Product1 für die Verwendung in View-Componente Binding zur Verfügung.

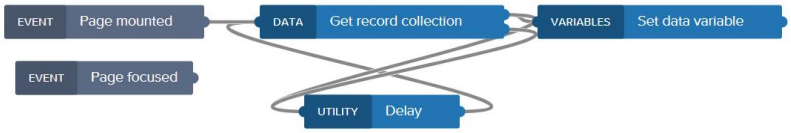

Abbildung 3.12: Data-Variable Logik

```
/**
 * constructor for the transformation engine
 *
 * Reads the Excel template of a specific agency via InputStream.
 * Reads the YAML file via InputStream that contains the mapping between
     the Excel sheet and the device data given by JSONata expressions.
 * Reads the device data via String that shall be filled into the
     template. It needs to be in a JSON format of an array containing a
     JSON object for each device with all of its data.
 *
 * @param inTemplate InputStream of the Excel template given by agency
 * @param inMapping InputStream of the YAML file describing the mapping
     between Excel template and device data
 * @param deviceData String in JSON format of an array of all device
     objects with their data
 * @throws TransformationException custom exception for any errors that
     specifically occur during the transformation
 */
public TransformationEngine(@NonNull InputStream inTemplate, @NonNull
    InputStream inMapping, @NonNull String deviceData) throws
    TransformationException {
  setTemplate(inTemplate);
  setMapping(inMapping);
  setDevices(deviceData);
}

/**
```

```
19  * Populates the Excel template with the given device data based on the
        mapping information in the YAML file.
20  * Writes the Excel workbook to the OutputStream, for example a file.
21  *
22  * @param outExcel the OutputStream for the filled Excel template
23  * @return OutputStream with the filled Excel template
24  * @throws TransformationException custom exception for any errors that
        specifically occur during the transformation
25  */
26 public OutputStream fillTemplateWithDevices(@NonNull OutputStream
        outExcel) throws TransformationException {
27   this.filledTemplate = fillTemplate(template, mapping, devices);
28   return writeExcel(filledTemplate, outExcel);
29 }
```

Quelltext 3.1: Auszüge aus der Klasse `TransformationEngine`

3.3 Ansatz mit Excel-Datei

Die Implementierung von Ansatz **??** mit der bearbeiteten Excel-Datei ist im Sequenzdiagramm in Abbildung 3.13 grob dargestellt.

Im Konstruktor werden die Produktdaten `deviceData` eingelesen und als JsonNode abgespeichert. Außerdem wird die Excel-Arbeitsmappe aus dem InputStream `inTemplate` mit der Bibliothek Apache POI geladen.

Bevor die Arbeitsblätter mit Daten gefüllt werden können, wird zunächst das Settings-Arbeitsblatt ausgelesen und nach möglichen Mappings durchsucht. Diese werden in Hashtabellen gespeichert und im Anschluss kann das Settings-Sheet gelöscht werden. Dies wurde bereits in Abschnitt **??** näher beschrieben und ist hier nur kurz skizziert.

Danach wird jedes Arbeitsblatt Zelle für Zelle nach Eingaben durchforstet, die mit `<` beginnen und `>` enden, um alle Spalten zu finden, welche mit Produktdaten gefüllt werden sollen. Der Code dafür ist im Quelltext 3.2 abgebildet.

```
1  public HashMap<String, ColumnInfo> getColumns(Sheet templateSheet,
       JmesPath<JsonNode> jmesPath) throws TransformationException {
2    HashMap<String, ColumnInfo> columns = new HashMap<String,ColumnInfo>();
3    this.startRow = -1;
4
5    for (Row row : templateSheet) {
6      for (Cell cell : row) {
7        if (cell.getCellType() == CellType.STRING) {
8          String cellValue = cell.getRichStringCellValue().getString();
9          if (cellValue.startsWith("<") && cellValue.endsWith(">")) {
```

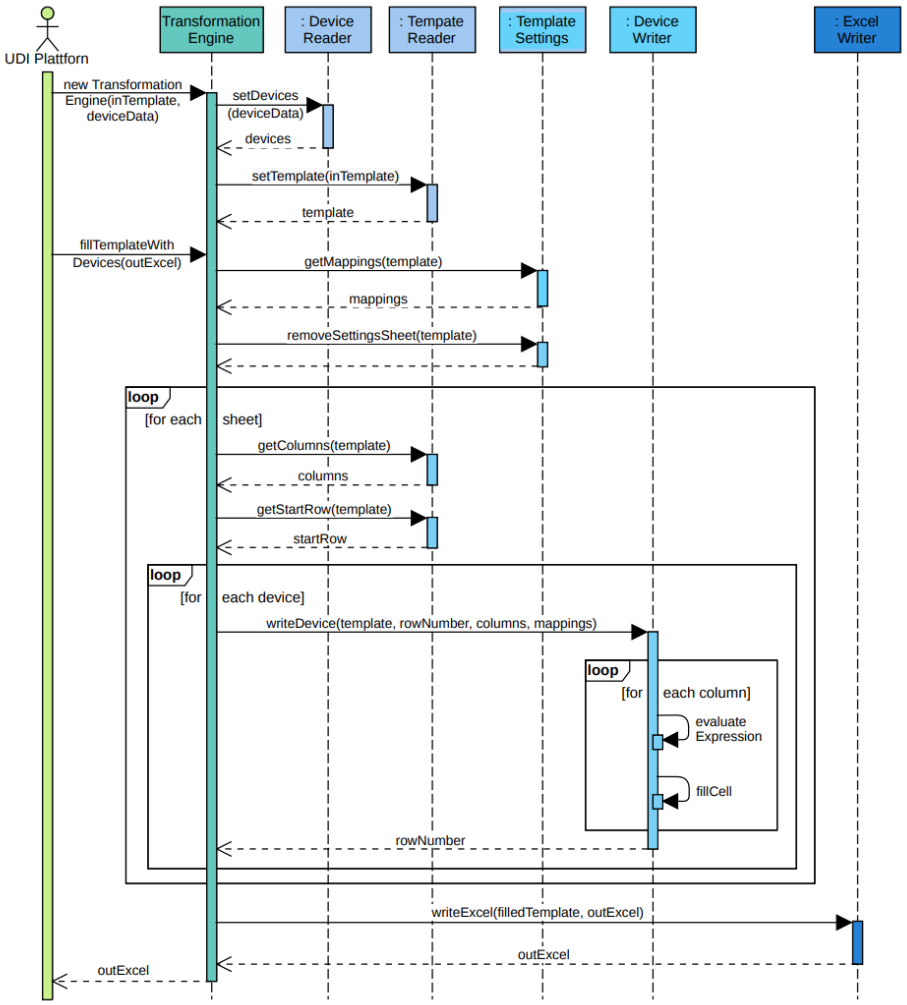

Abbildung 3.13: Sequenzdiagramm

```
10          //entry found
11          cellValue = cellValue.substring(1, cellValue.length() - 1);
12
13          //check if element is mandatory or cell can be left empty
14          boolean mandatory = false;
15          if (cellValue.statssWith("!")) {
16            mandatory = true;
17            cellValue = cellValue.substring(1, cellValue.length());
18          }
19
20          //parse JMESPath expression
21          try {
22            Expression<JsonNode> expression = jmesPath.compile(cellValue)
                  ;
23          } catch (ParseException e) {
24            String colLetter = CellReference.convertNumToColString(cell.
```

```
                getColumnIndex());
25            String message = String.format("JMESPath expression of sheet
                  \"%s\", column %s cannot be parsed.\n%s",
26                  templateSheet.getSheetName(), colLetter, e.getMessage()
                        );
27            log.error(message);
28            throw new TransformationException(message, e);
29          }
30
31          //save all infos for this column
32          ColumnInfo column = new ColumnInfo(cell.getColumnIndex(),
                mandatory, expression);
33          columns.put(cellValue, column);
34
35          checkIfRowsAlign(templateSheet, row.getRowNum());
36        }
37      }
38    }
39  }
40  return columns;
41 }
```

Quelltext 3.2: Einlesen der Spalteninformationen in `TemplateReader`

Sobald eine Zelle gefunden wurde, die ein Spalteneintrag enthält (siehe Zeile 9), werden die entsprechenden Informationen verarbeitet und abgespeichert. Wie man in Zeile 32 im folgenden Code sieht, wird nicht nur die Spaltennummer gespeichert, sondern auch die Angabe, ob die Spalte verpflichtend ist. Ebenso der kompilierte JMESPath-Ausdruck, mit dem später die geforderten Werte der einzelnen Produkte ermittelt werden können. Falls der Ausdruck fehlerhaft ist und nicht geparst werden kann, wird eine Exception ausgelöst, gefangen und in die eigens definierte `TransformationException` umgewandelt.
Zum Schluss in Zeile 35 wird außerdem in der Methode `checkIfRowsAlign` die Excel-Zeilennummer abgeglichen (siehe Quelltext 3.3). Diese beschreibt die Start-Zeile, in die der Inhalt des ersten Produkts eingetragen wird und womit die JMESPath-Ausdrücke aus der Vorlage überschrieben werden. Unterscheiden sich die Start-Zeilen innerhalb eines Arbeitsblattes, wird ebenfalls eine `TransformationException` geworfen, weil die Einträge eines Datensatzes nicht alle in gerader Fluchtlinie (d. h. in einer Zeile) angeordnet sind.

```
1 private checkIfRowsAlign(Sheet templateSheet, int rowNumber) throws
      TransformationException;
2   switch (this.startRow) {
3     case -1:        //first entry - initialize
4       this.startRow = rowNumber;
```

```
5         break;
6       case rowNumber: //all good
7         break;
8       default:        //no alignment - error
9         String message = String.format("The starting rows for the different
              device data fields do not align in sheet \"%s\".",
10             templateSheet.getSheetName());
11        log.error(message);
12        throw new TransformationException(message);
13        break;
14    }
15  }
```

Quelltext 3.3: Hilfsfunktion beim Einlesen der Spalteninformationen

Obwohl über Blätter, Spalten und Zeilen iteriert wird, liegt der Aufwand im Bereich $\mathcal{O}(n)$, wobei n die Anzahl der Spalten beschreibt bzw. umformuliert die Anzahl der von der Behörde geforderten Produktinformationen. Diese Anzahl wird nicht sonderlich groß, sondern bewegt sich im zwei- bis unteren dreistelligen Bereich[1]. Die Anzahl der Arbeitsblätter und nicht-leeren Zeilen in einer unausgefüllten Vorlage ist daher vernachlässigbar klein und entsprechend verbraucht die Suche nach den Spalten, d. h. die Ausführung von `getColumns`, wenig Zeit.

Nachdem alle Spalten- und Mapping-Informationen vorliegen und ausgewertet sind, werden die einzelnen Produkte in die Vorlage geschrieben. Dies geschieht in der Klasse `DeviceWriter`. Hierbei wird nochmal über die Spalten iteriert. Die JMESPath-Ausdrücke werden auf das explizite Produkt angewendet und der so erhaltene Wert wird zunächst anhand der vorliegenden Mappings in die behördenspezifische Format- oder Formulierungsvorgabe umgewandelt. Der ggf. „übersetzte“ Wert wird dann entsprechend seines Types in die Excel-Zelle geschrieben. Dabei werden Wahrheitswerte, Zahlen und Text sowie Datumsangaben unterstützt. Für Letztere ist das Format allerdings aktuell fest vorgegeben – hier könnte man durch zusätzliche Angaben im Settings-Arbeitsblatt behördenspezifische Wünsche inkludieren.

Wie bereits erwähnt, ist in der Regel pro Spalte und Produkt nur genau ein Wert gefordert, aber es kann auch vorkommen, dass zu einem Produkt mehrere Informationen bzgl. einer Kategorie vorliegen, die in mehrere Excel-

[1] Datensatz von Behörden mit UDI-System in [Len22]: näherungsweise Anzahl an geforderten Datenattributen – Minimum: 13 (Singapur); Maximum: 130 (EU)

Zeilen geschrieben werden müssen (siehe Abbildung ??). Der JMESPath-Ausdruck liefert in diesem Fall ein Array, deren einzelne Einträge jeweils eine eigene Excel-Zeile erhalten. Hierbei ist wichtig, dass `null`-Werte mitgegeben werden, falls ein Datenelement in einem Array nicht vorhanden sein sollte, sodass für ein Produkt die Länge des Arrays pro Kategorie in jeder Spalte gleich ist. Dadurch wird gewährleistet, dass jeweils alle Werte einer Zeile zueinander gehören. Standardmäßig werden Nullwerte bei einer JMESPath-Projektion allerdings herausgefiltert und nicht berücksichtigt. Um dies zu umgehen, kann man sich Multiselect-Listen bedienen, die mit dem Pipe-Operator dann im Nachgang wieder glättet werden. Als Beispiel kann man das komplexe Element `TradeNames` betrachten.

```
1  {
2      "TradeNames": [
3          {
4              "TradeName": "englishName",
5              "TradeNameLanguage": "EN"
6          },
7          {
8              "TradeName": "noLanguageGiven"
9          },
10         {
11             "TradeName": "deutscherName",
12             "TradeNameLanguage": "DE"
13         }
14     ],
15     ...
16 }
```

Quelltext 3.4: Beispiel eines komplexen Datenelements

Um für jedes Objekt eine neue Zeile zu erzeugen, in der sowohl `TradeName` als auch `TradeNameLanguage` in einer eigenen Spalte angegeben werden, bieten sich die folgenden JMESPath-Ausdrücke an:

- `<TradeNames[].[TradeName] | []>`
- `<TradeNames[].[TradeNameLanguage] | []>`

Durch die zusätzlichen eckigen Klammern um die Datenelemente `TradeName` und `TradeNameLanguage` wird anstatt einer normalen Projektion eine Multiselect-Liste erzeugt, die in diesem Fall aus ein-elementigen Arrays besteht, weil nur ein Datenelement gesucht ist. Jedes Auswertungsteilergebnis ist dabei enthalten, auch wenn es `null` beträgt – in diesem Beispiel die nicht vorhandene Angabe der Sprache im zweiten Objekt (Zeile 8). Mit

`|[]` wird das erzeugte Array von Arrays wieder geglättet. Alternativ kann mit der eingebauten Funktion `map` gearbeitet werden. Der Ausdruck `map(&TradeNameLanguage, TradeNames[])` führt zu demselben Ergebnis, da auch hier eine neue Liste erzeugt wird, die die gleiche Länge wie die ursprüngliche Liste hat. Jedes Objekt in `TradeNames` wird auf die `TradeNameLanguage` abgebildet, falls vorhanden – ansonsten auf `null`. Im Excel-Template wird der Nullwert dann als leere Zelle interpretiert.

Liegen fehlerhafte Produktdaten vor oder können JMESPath-Ausdrücke nicht interpretiert werden, bricht das Programm mit einer Fehlermeldung ab und die Informationen zum Produkt bzw. zur Spalte im Arbeitsblatt werden geloggt. Ursprünglich war die Logik so, dass Produkte mit Datenfehlern (z. B. ein fehlender, verpflichtender Wert) aus der Excel-Vorlage gelöscht und geloggt, alle restlichen Produkte aber eingetragen wurden. Dadurch konnte immer eine valide Excel-Datei erzeugt werden, die möglicherweise aber nur zu einem Bruchteil ausgefüllt war. Die Rücksprache im Team ergab, dass das erfolgreiche Durchlaufen trotz einzelner Fehler dazu führt, dass diese übersehen werden. Anstatt lediglich Einträge im Log zu hinterlassen, wird nun eine Exception geworfen.

Die Implementation des `DeviceWriters`, um die Daten eines Produktes in die Arbeitsmappe zu schreiben, folgt bei beiden Ansätzen identischen Prinzipien. Ansatz 3.4 ist etwas umfangreicher und es wurden einige Details optimiert, daher wird zur Veranschaulichung an dieser Stelle auf das später folgende Aktivitätsdiagramm 3.15 verwiesen.

Konnten alle Produktdaten erfolgreich in die Arbeitsmappe eingetragen werden, wird das ausgefüllte Excel-Template in der Klasse `ExcelWriter` in einen OutputStream geschrieben. Dabei kann es sich beispielsweise um eine Datei handeln, die dann erzeugt wird.

3.4 Ansatz mit Mapping-Datei

Die Implementierung für den Ansatz mit der extra YAML-Datei für die Projektion von Produktdaten in die Excel-Datei ist in Abbildung 3.14 in einem Sequenzdiagramm dargestellt. Er ist ähnlich aufgebaut wie Ansatz 3.3. Im Konstruktor werden wie bisher die Excel-Vorlage geladen und die Produktdaten deserialisiert. Zusätzlich wird nun die Mapping-Datei eingelesen und direkt validiert (siehe Abschnitt **??**). Um die Produktdaten

in die Arbeitsmappe zu schreiben, wird ebenfalls wieder über die Arbeitsblätter, die Produkte und die einzelnen Spalten iteriert. Dies geschieht in der Klasse `DeviceWriter`.

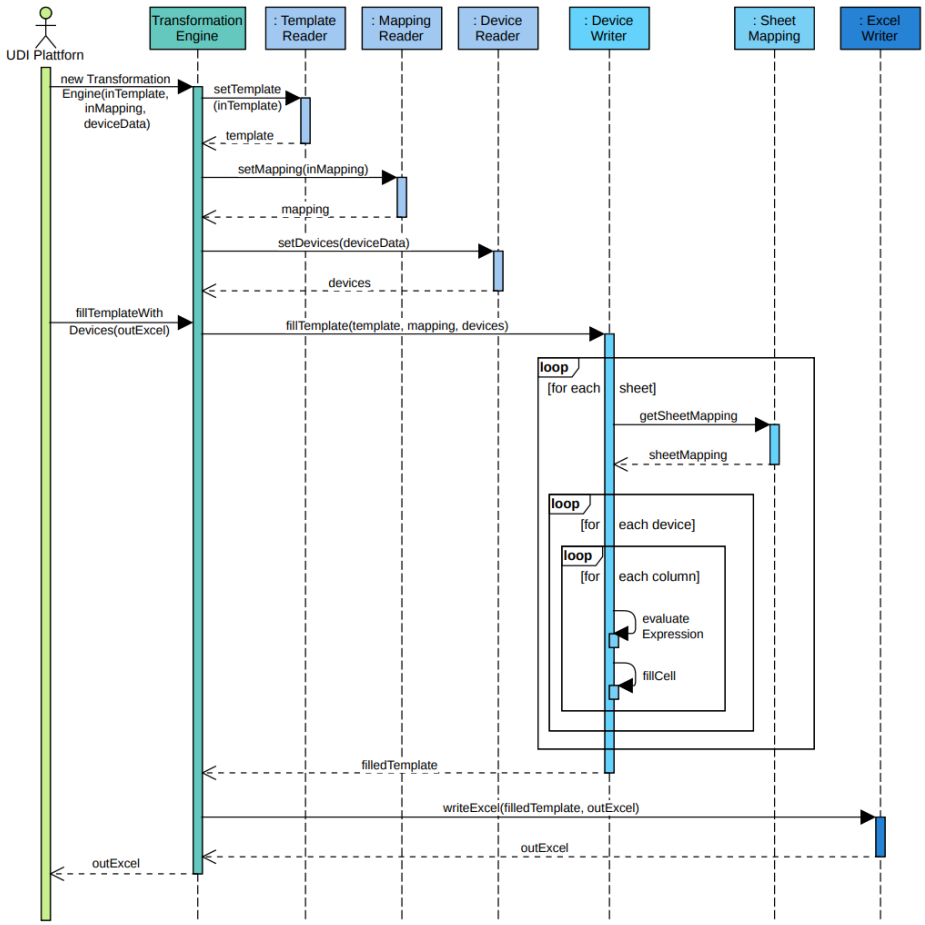

Abbildung 3.14: Sequenzdiagramm

Eine interne Verbesserung zur Implementierung des vorherigen Ansatzes besteht darin, dass die Produktdaten, die auf behördenspezifische Werte abgebildet werden sollen, direkt als erstes vor den verschiedenen Schleifendurchläufen transformiert werden. Das gilt auch für die Datums- und Zeitformate. Dabei wird für jedes gewünschte Excel-Format ein `CellStyle` angelegt, in einer Hashtabelle gespeichert und später auf diejenigen Zellen angewendet, die ein Datum im angegebenen Format enthalten (sollen). Für jeden in der Mapping-Datei angegebenen Arbeitsblatt-Titel werden die folgenden vier Schritte durchgeführt:

1. Das zugehörige Arbeitsblatt in der Excel-Vorlage suchen: Kann kein

Arbeitsblatt mit dem angegebenen Namen gefunden werden, wird ein Fehler geworfen und die Transformationsengine bricht ab.

2. Die Angaben zum Arbeitsblatt aus der Mapping-Datei per Datenbindung in ein Objekt der Klasse `SheetMapping` übertragen: Hierauf wird im folgenden Abschnitt 3.4.1 noch näher eingegangen.

3. Die Variablen aus `SheetMapping` abrufen, unter anderem die Startzeile, die verschiedenen Spaltennamen mit deren JSONata-Ausdrücken sowie eine Liste aller verpflichtenden Spalten: Wurden dabei syntaktisch falsche Werte übergeben, wird ebenfalls ein Fehler geworden.

4. Jedes Produkt in eine oder ggf. mehrere Zeilen des Excel-Arbeitsblatts schreiben: Dabei wird wie in Kapitel **??** vorgegangen, indem spaltenweise der JSONata-Ausdruck ermittelt und dann im richtigen Format in die passende Zelle geschrieben wird. Dies wird in Abschnitt 3.4.2 erläutert.

3.4.1 Datenbindung in SheetMapping

Alle JSON-Daten zu einem Arbeitsblatt werden dank der Bibliothek Jackson an eine Instanz der Klasse `SheetMapping` gebunden, wie der Codeausschnitt 3.5 zeigt. Man spricht hier von Daten-Deserialisierung, da JSON-Knoten in Java-Objekte umgewandelt werden. Die Datenbindung an ein Objekt hat gegenüber der Speicherung im Baummodell den Vorteil, dass Abfragen deutlich bequemer durchführbar sind und einfacher mit den Daten gearbeitet werden kann. Anstatt wiederholt die Pfade des Baumes abzulaufen, können Parameter direkt referenziert werden.

```
1  ObjectMapper jsonReader = new ObjectMapper();
2  jsonReader
3      .enable(DeserializationFeature.ACCEPT_EMPTY_STRING_AS_NULL_OBJECT);
```

```
4  private SheetMapping getSheetMapping(ObjectMapper jsonReader, String
       sheetName) throws TransformationException {
5    InjectableValues injectableValues =
6        new InjectableValues.Std().addValue(String.class, sheetName);
7    try {
8      return jsonReader
9          .reader(injectableValues)
10         .treeToValue(sheetMappings.get(sheetName), SheetMapping.class);
11   } catch (JsonProcessingException | IllegalArgumentException e) {
12     String message = String.format("Sheet called \"%s\" could not be
           filled, because the mapping cannot be parsed properly.",
           sheetName);
```

```
13      log.error(message);
14      throw new TransformationException(message, e);
15    }
16  }
```

Quelltext 3.5: Datenbindung für Arbeitsblätter der Mapping-Datei

Umgesetzt wird die Anbindung durch Annotationen in der Klasse `SheetMapping`. Mit `@JacksonInject` wird der Name des Arbeitsblatts zur Klasse hinzugefügt, um diese Info den Fehlermeldungen mitgeben zu können. Dies wird in Zeile 5 im Code 3.5 vorbereitet. Mit `@JsonProperty` werden alle weiteren Attribute gesetzt, die im Anschluss aufgelistet sind.

- die Kategorie `category` als `String`:
  Falls kein komplexes Datenelement angegeben wurde, auf welches sich das Arbeitsblatt bezieht, beträgt der Wert `null`.

- die Startzeile `row` als `Integer`:
  Die Zeile muss größer als 0 sein, ansonsten bricht die Transformationsengine mit einer Fehlermeldung ab. Außerdem wird sie um eins reduziert, da die Bibliothek Apache POI mit 0-basierten Zeilennummern arbeitet, während die Zeilen in Excel bei 1 beginnen.

- die Spalten `columns` als `HashMap <String, String>` mit dem Spaltennamen als Schlüssel und dem JSONata-Ausdruck als Wert:
  In der zugehörigen Getter-Methode werden die Spaltennamen zunächst auf syntaktische Korrektheit überprüft (1-2 Buchstaben von A bis Z) und dann in die entsprechende 0-basierte Spaltenzahl umgewandelt, mit welcher Apache POI rechnet. Die Funktion ist im Code 3.6 in Zeile 14– 28 dargestellt. Außerdem werden die JSONata-Ausdrücke mittels der Bibliothek JSONata4Java geparsed und in `Expressions` umgewandelt. So wird also der Typ `Map <Integer, Expressions>` zurückgegeben. Tritt ein Fehler im Spaltennamen oder während des JSONata Parsens auf, wird eine `TransformationException` geworfen.

- die verpflichtenden Spalten `mandatoryColumns` als `HashSet <String>`:
  Bevor die Menge der Spalten herausgegeben wird, werden auch hier die Spaltennamen in Zahlen, beginnend bei 0, umgerechnet und gegebenenfalls ein Fehler geworfen.

Im folgenden Codeausschnitt 3.6 ist exemplarisch die Implementierung für die obligatorischen Spalten dargestellt. Alle gewöhnlichen Setter- und Get-

ter-Methoden ohne besondere Funktionalitäten werden einfachheitshalber mit Hilfe von Lombok-Annotationen deklariert, wie z. B. in Zeile 2.

```
1  @JsonProperty("mandatoryColumns")
2  @Setter
3  private Set<String> mandatoryColumns = new HashSet<String>();
4
5  //Getter
6  public Set<Integer> getParsedMandatoryColumns() throws
       TransformationException {
7    Set<Integer> mandatoryColumnsParsed = new HashSet<Integer>();
8    mandatoryColumns.forEach((columnLetter) -> {
9      mandatoryColumnsParsed.add(parseCol(columnLetter));
10   });
11   return mandatoryColumnsParsed;
12 }
13
14 private Integer parseCol(String columnLetter) throws
       TransformationException {
15   columnLetter = columnLetter.toUpperCase();
16   if (!columnLetter.matches("[A-Z]{1,2}")) {
17     String message = String.format(
18         "Yaml file describing agency template is not filled correctly: in
               sheet \"%s\", column %s is not a valid excel column name.",
19         sheetName, columnLetter);
20     log.error(message);
21     throw new TransformationException(message);
22   }
23   int columnNumber = 0;
24   for (int i = 0; i < columnLetter.length(); i++) {
25     columnNumber = columnNumber * 26 + columnLetter.charAt(i) - 'A' + 1;
26   }
27   return columnNumber - 1;
28 }
```

Quelltext 3.6: Setter / Getter für verbindliche Spalten in `SheetMapping`

Die `TransformationException` ist eine benutzerdefinierte Ausnahme (siehe dazu [SDW20, S. 32f] und [GHK14, S. 309–338]), welche möglichst alle checked Exceptions zusammenfasst, die bei der Transformation entstehen können. Ausgenommen sind ungeprüfte Laufzeit-Fehler. NullPointer-Exceptions werden mittels Lombok erzeugt, falls ein Eingabeparameter `null` ist. Betrachtet man die Transformationsengine im Zusammenspiel mit der restlichen Plattform, wird so direkt ersichtlich, dass ein Fehler innerhalb der Transformationsengine aufgetreten ist und im Log kann die genaue Ursache nachgelesen werden.

3.4.2 Produktdaten schreiben

Die spaltenweise Auswertung des JSONata-Ausdrucks und das Füllen der Zelle in der Excel-Vorlage funktioniert ganz anlog zu Abschnitt 3.3. Es ergibt sich das Aktivitätsdiagramm aus Abbildung 3.15 für die Methode `writeOneDevice` der Klasse `DeviceWriter`, um – wie der Name schon sagt – die Daten zu einem Produkt in ein Arbeitsblatt zu schreiben.

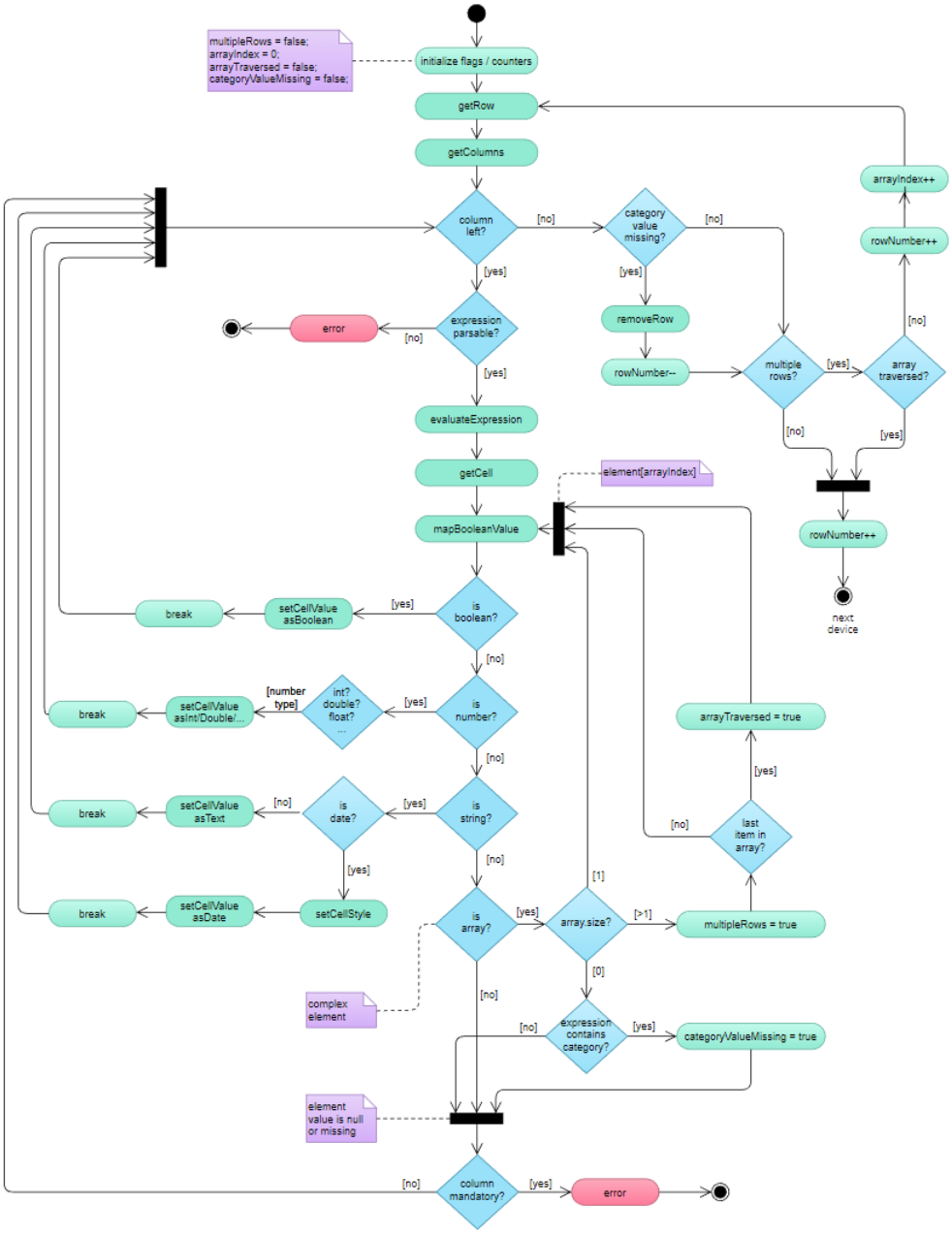

Abbildung 3.15: Aktivitätsdiagramm für `writeOneDevice`

Um den Prozess im Aktivitätsdiagramm nachvollziehen zu können, wird zunächst der Unterschied zwischen einfachen und komplexen Datenelementen näher erläutert. Einfache Datenelemente stehen in einer $1:1$-Beziehung zum Produkt, das heißt es liegt nur eine einzige Ausprägung vor, die dem Datenelement zugeordnet ist. Dies kann als String, Zahl oder Wahrheitswert sein. Ein Spezialfall bildet der in Abschnitt **??** erwähnte `ProduktionIdentifier`, welcher in einem Array von Strings unterschiedliche einfache Elemente zusammenfasst. Er wird jedoch bereits über den JSONata-Ausdruck abgefangen und in einen Wahrheitswert umgewandelt, sodass hierfür keine zusätzliche Logik implementiert werden muss. Die komplexen Datenelemente bestehen hingegen aus einem Array von Objekten, welche wiederum einfache Datenelemente enthalten. Ein Beispiel wurde bereits in Quelltext 3.4 gegeben. Sie stehen also in einer $1:n$-Beziehung, d. h. ein Produkt kann mehrere unterschiedliche Ausprägungen / Einträge für ein komplexes Datenelement haben. In Excel können alle einfachen Elemente in einem Arbeitsblatt in einer Zeile aneinander gereiht werden, da ein Element genau einer Zelle entspricht. Die komplexen Elemente werden in der Regel in zusätzlichen, „relationalen“ Arbeitsblättern dargestellt, in denen jedes Objekt des Arrays des komplexen Elements einer Zeile entspricht. Ein Produkt füllt in diesem Fall also n Zeilen, wobei $n \geq 0$ gilt. Falls für ein Produkt keine Daten bzgl. eines komplexen Elements vorliegen (d. h. $n = 0$), wird es im entsprechenden relationalen Arbeitsblatt auch nicht aufgelistet, ansonsten wird über alle n Objekte im Array iteriert.

Die Methode `writeOneDevice` beginnt mit der Initialisierung verschiedener Flags und Counter, die im weiteren Verlauf Verwendung finden. Danach wird zunächst die aktuelle Zeile anhand der `rowNumber` im aktuellen Arbeitsblatt zur Bearbeitung ausgewählt. Die Liste der Spalten für das jeweilige Arbeitsblatt entsprechend der Mapping-Datei wird geladen und im Folgenden durchlaufen. Dabei wird spaltenweise der JSONata-Ausdruck evaluiert (falls dies möglich ist), sowie die aktuelle Zelle ausgewählt und gefüllt. Beim Füllen werden zunächst mögliche Wahrheitswerte in behördenspezifische Werte umgewandelt und dann wird per Switch-Statement über den Datentyp die Zelle typgerecht mit dem Ergebnis der JSONata-Auswertung beschrieben. Es werden primitive Datentypen, wie Booleans und die unterschiedlichen Zahlenwerte, sowie Strings unterstützt. Liegen bei Strings spezielle Datums- oder Zeitangaben vor, muss die Excel-Zelle zusätzlich in ihrem Style formatiert werden. Ansonsten kann die Zelle über die

entsprechende Apache POI-Methode `setCellValueAs<Type>` beschrieben werden. Eine Ausnahme ergibt sich jedoch, wenn der JSONata-Ausdruck ein Array zurückliefert. In dem Fall handelt es sich um ein komplexes Datenelement – hier muss überprüft werden, ob überhaupt Einträge vorliegen. Wenn dies der Fall ist, wird über den `arrayIndex` iteriert und für jeden Eintrag eine neue Zeile geschrieben, bis das gesamte Array durchlaufen ist. Die restlichen simplen Dateneinträge aus anderen Spalten werden einfach wiederholt. Ist das komplexe Datenelement allerdings leer, wird geprüft, ob es mit der Kategorie übereinstimmt und ggf. die Zeile gelöscht. Dies trifft zu, wenn ein Produkt beispielsweise gar nicht sterilisierbar ist, sodass auch keine Sterilisationsmethoden vorliegen können. In diesem Fall soll das entsprechende Produkt auch nicht im zugehörigen Arbeitsblatt aufgelistet werden. Liefert ein JSONata-Ausdruck kein Ergebnis oder beträgt `null`, obwohl die Spalte verpflichtend ist, kommt es zur Fehlermeldung. Ansonsten werden alle Spalten durchgegangen und beim erfolgreichen Ausfüllen wird die aktualisierte `rowNumber` für das nächste Produkt zurückgegeben.

Auch hier ist bei Arrays bei der Wahl des JSONata-Ausdrucks Vorsicht geboten. Es muss bedacht werden, dass nicht vorhandene Datenelemente von komplexen Eltern explizit mit `null` besetzt werden müssen, anstatt sie implizit wegzulassen. Der JSONata-Ausdruck wird dadurch ein wenig komplizierter, was man in den Zeilen **??** und **??** des YAML-Codes **??** sieht. `TradeNames.TradeName` muss beispielsweise zu `TradeNames.($exists(TradeName) ? TradeName : null)` erweitert werden. Dadurch wird gewährleistet, dass in jeder Zeile die einander zugehörigen Informationen stehen und sich in den relationalen Arbeitsblättern keine Verschiebungen ergeben, sondern die Zellen der unterschiedlichen Spalten zueinander ausgerichtet sind.

Sind alle Produktdaten erfolgreich in alle Arbeitsblätter geschrieben worden, wird wie in Abschnitt 3.3 im letzten Schritt die ausgefüllte Excel-Datei erzeugt und als OutputStream zurückgegeben.

Literaturverzeichnis

[Ant16] Antolovic, Miroslav: *Einführung in SAPUI5*, 2. Auflage. Rheinwerk Verlag, 2016. – isbn: 978–3–8362–8901–6

[App22a] AppGyver: *App logic*. 2022. – url: https://docs.appgyver.com/docs/app-logic [abgerufen am 29.11.2022]

[App22b] AppGyver: *Community Announcement*. 2022. – url: https://forums.appgyver.com/t/community-announcement/19453 [abgerufen am 22.11.2022]

[App22c] AppGyver: *View Components*. 2022. – url: https://docs.appgyver.com/docs/view-components [abgerufen am 29.11.2022]

[AS22] Andreas Schaffry, Directorate-COMPUTERWOCHE: *Studie No-Code/Low-Code 2022 Licht und Schatten*. 2022. – url: https://www.computerwoche.de/a/licht-und-schatten,3553554 [abgerufen am 11.10.2022]

[Cen21] Center, SAP N.: *SAP übernimmt No-Code-Pionier AppGyver*. 2021. – url: https://news.sap.com/germany/2021/02/sap-uebernimmt-no-code-pionier-appgyver/ [abgerufen am 11.10.2022]

[Cod22a] Code, Visual S.: *Overview*. 2022. – url: https://code.visualstudio.com/docs [abgerufen am 01.12.2022]

[Cod22b] Code, Visual S.: *Terminal Basics*. 2022. – url: https://code.visualstudio.com/docs/terminal/basics [abgerufen am 01.12.2022]

[Com22a] Community, SAP: *SAPUI5*. 2022. – URL: https://community.sap.com/topics/ui5 [abgerufen am 25.11.2022]

[Com22b] Community, SAP: *Sneak Peek on SAP AppGyver Integration Into SAP BTP*. 2022. – URL: https://groups.community.sap.com/t5/devtoberfest/sneak-peek-on-sap-appgyver-integration-into-sap-btp/ec-p/8958#M17 [abgerufen am 13.10.2022]

[Con22] Consulting, Hecker: *Die Zukunft der Software Entwicklung: Low-Code-Plattformen*. 2022. – URL: https://www.hco.de/blog/die-zukunft-der-software-entwicklung-low-code-plattformen [abgerufen am 11.11.2022]

[CVE22] Coen Van Eenbergen, Directorate-TECHZINE: *ServiceNow fully enters low-code market with App Engine Studio*. 2022. – URL: https://www.techzine.eu/blogs/applications/56875/servicenow-fully-enters-low-code-market-with-app-engine-studio/ [abgerufen am 17.11.2022]

[Doc22a] Documentation, SAPUI5: *Developing Apps with SAP Fiori Elements*. 2022. – URL: https://sapui5.hana.ondemand.com/#/topic/03265b0408e2432c9571d6b3feb6b1fd [abgerufen am 28.11.2022]

[Doc22b] Documentation, SAPUI5: *Using SAP Fiori Elements Floorplans*. 2022. – URL: https://sapui5.hana.ondemand.com/#/topic/03265b0408e2432c9571d6b3feb6b1fd [abgerufen am 28.11.2022]

[doc22c] documentation, ServiceNow P.: *App Engine Studio release notes*. 2022. – URL: https://docs.servicenow.com/bundle/store-release-notes/page/release-notes/store/platform-app-engine/store-rn-plat-app-engine-aes.html [abgerufen am 17.11.2022]

[Ele22] Elements, SAP F.: *SAP Fiori Design Guidelines*. 2022. – URL: https://experience.sap.com/fiori-design-web/smart-templates/ [abgerufen am 13.10.2022]

[Eng20] Englbrecht, Michael: *SAP Fiori Implementierung und Entwicklung*, 3. Auflage. Rheinwerk Verlag, 2020. – isbn: 978–3–8362–7268–1

[G222] G2: *Salesforce Platform Reviews and Product Details*. 2022. – url: https://www.g2.com/products/salesforce-platform/reviews [abgerufen am 18.11.2022]

[GG22] Gartner Glossary, Directorate-Gartner: *Citizen Developer*. 2022. – url: https://www.gartner.com/en/information-technology/glossary/citizen-developer [abgerufen am 11.11.2022]

[GHK14] Gallardo, Raymond; Hommel, Scott; Kannan, Sowmya; Gordon, Joni; Zakhour, Sharon B.: *The Java® Tutorial: A Short Course on the Basics*, 6. Auflage. Upper Saddle River, NJ: Addison-Wesley, 2014. – isbn: 978–0–13–403408–9

[Gmb22] GmbH p36: *p36 Überblick*. 2022. – url: https://p36.io/professional-services/sap-consulting/?lang=de [abgerufen am 11.10.2022]

[Gui22] Guidelines, SAP Fiori D.: *SAP Fiori Launchpad*. 2022. – url: https://experience.sap.com/fiori-design-web/launchpad/ [abgerufen am 24.11.2022]

[HV22] Heinrich Vaske, Directorate-COMPUTERWOCHE: *Low-Code-Plattformen auf einen Blick*. 2022. – url: https://www.computerwoche.de/a/low-code-plattformen-auf-einen-blick,3544905 [abgerufen am 10.10.2022]

[Lea22] Learning, SAP: *Getting Started With Low-Code/No-Code and SAP Build*. 2022. – url: https://learning.sap.com/learning-journey/utilize-sap-build-for-low-code-no-code-applications-and-automations-for-citizen-developers/getting-started-with-low-code-no-code-and-sap-build_afaa85b8-93eb-4607-94b8-221c80aa6479 [abgerufen am 18.11.2022]

[Len22] Lentz, Bethaney: *Quick reference guide - global medical device UDI requirements and timelines*. 2022. – url: https://www.ri

msys.io/blog/quick-reference-guide-global-udi-requirements-and-timelines [abgerufen am 22.08.2022]

[Mar22] Marketplace, Visual S.: *SAP CDS Language Support*. 2022. – URL: https://marketplace.visualstudio.com/items?itemName=SAPSE.vscode-cds [abgerufen am 19.12.2022]

[Ove22a] Overview, G2: *228 Listings in Low-Code Development Platforms Available*. 2022. – URL: https://www.g2.com/categories/low-code-development-platforms [abgerufen am 11.11.2022]

[Ove22b] Overview, G2: *289 Listings in No-Code Development Platforms Available*. 2022. – URL: https://www.g2.com/categories/no-code-development-platforms [abgerufen am 11.11.2022]

[Por22a] Portal, SAP H.: *Destinations*. 2022. – URL: https://help.sap.com/docs/SAP_BUILD_CLASSIC/acb8ee53fa0e448f9b4a4125591f5d5a/4be99e779fcb4ff9a8e5179b74bd72cc.html?locale=en-US [abgerufen am 09.12.2022]

[Por22b] Portal, SAP H.: *What Is SAP Business Application Studio*. 2022. – URL: https://help.sap.com/docs/SAP%20Business%20Application%20Studio/9d1db9835307451daa8c930fbd9ab264/8f46c6e6f86641cc900871c903761fd4.html [abgerufen am 29.11.2022]

[RJR22] Richardson, Clay; John Rymer, Directorate-FORRESTER: *New Development Platforms Emerge For Customer-Facing Applications*. 2022. – URL: https://www.forrester.com/report/New-Development-Platforms-Emerge-For-CustomerFacing-Applications/RES113411 [abgerufen am 10.10.2022]

[SAP22a] SAP: *About CAP*. 2022. – URL: https://cap.cloud.sap/docs/about/ [abgerufen am 01.12.2022]

[SAP22b] SAP: *AppGyv*. 2022. – URL: https://www.appgyver.com/ [abgerufen am 10.10.2022]

[SAP22c] SAP: *Generator-easy-ui5*. 2022. – URL: https://github.com/SAP/generator-easy-ui5 [abgerufen am 01.12.2022]

[SDW20] Schaaf, Sebastian; Dörpinghaus, Jens; Weil, Vera: *Java für die Life Sciences: Eine Einführung in die angewandte Bioinformatik*, 1. Auflage. Heidelberg: O'Reilly, 2020. – ISBN: 978–3–96009–125–7

[Wik22a] Wikipedia: *Node.js*. 2022. – URL: `https://en.wikipedia.org/wiki/Node.js` [abgerufen am 14.12.2022]

[Wik22b] Wikipedia: *SQLite*. 2022. – URL: `https://de.wikipedia.org/wiki/SQLite` [abgerufen am 14.12.2022]

[Wik22c] Wikipedia: *Visual Studio Code*. 2022. – URL: `https://de.wikipedia.org/wiki/Visual_Studio_Code` [abgerufen am 01.12.2022]

Anhang A

JSON-Schema

JSON-Schema wird verwendet, um die Struktur von JSON-Dateien zu validieren. Bei der Transformationsengine wird dieses Prinzip bei der Mapping-Datei aus Kapitel ?? angewendet – besonders in Abschnitt ?? wird detailiert auf JSON-Schema eingegangen. Das verwendete Schema ist in der Datei `MappingJsonSchema.json` gespeichert und wie folgt definiert:

```
{
  "$schema": "https://json-schema.org/draft-06/schema#",
  "title": "jsonataExcelMapping",
  "description": "A mapping for JSONata input into excel template",
  "type": "object",
  "properties": {
    "SheetMappings": {
      "description": "mappings for the sheets in the excel template",
      "type": "object",
      "additionalProperties": {
        "description": "different sheetNames",
        "type": "object",
        "properties": {
          "row": {
            "description": "start row in excel sheet where the device
                data shall be written",
            "type": "integer",
            "minimum": 1
          },
          "category": {
            "description": "category of the sheet (complex data
                element key)",
            "type": "string"
          },
          "columns": {
            "description": "different columns to be filled",
```

```
25            "type": "object",
26            "patternProperties": {
27              "^[a-zA-Z]{1,2}$": {
28                "description": "the key is the column letter in excel
                      format, the value is the JSONata expression to
                      filter the right device data for this column",
29                "type": "string"
30              }
31            },
32            "additionalProperties": false,
33            "minProperties": 1
34          },
35          "mandatoryColumns": {
36            "type": "array",
37            "description": "a list of those columns that are required
                    to be filled",
38            "default": [],
39            "items": {
40              "type": "string",
41              "pattern": "^[a-zA-Z]{1,2}$"
42            }
43          }
44        },
45        "required": [ "row", "columns" ]
46      }
47    },
48    "ElementMappings": {
49      "description": "mappings for booleans, date formats and/or
          specific data elements",
50      "type": "object",
51      "properties": {
52        "Boolean": {
53          "description": "mapping for boolean values in this template
                ",
54          "type": "object",
55          "properties": {
56            "true": {
57              "description": "mapping of true in this agency",
58              "type": ["number", "string", "boolean"]
59            },
60            "false": {
61              "description": "mapping of false in this agency",
62              "type": ["number", "string", "boolean"]
63            }
64          },
65          "additionalProperties": false,
66          "required": ["false", "true"]
67        },
68        "Date": {
69          "description": "mapping for date or time formats",
70          "type": "object",
71          "additionalProperties": {
```

```
72              "description": "key: date format for p36 / value: date
                    format in excel for agency",
73              "type": "string"
74            }
75          }
76        },
77        "additionalProperties": {
78          "description": "data element key",
79          "type": "object",
80          "additionalProperties": {
81            "description": "key: element value for p36 / value: element
                  value for agency",
82            "type": ["number", "string", "boolean"]
83          }
84        }
85      }
86    },
87    "required": [ "SheetMappings" ]
88  }
```

Quelltext A.1: JSON-Schema für die Mapping-Datei