



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

Erstgutachter: 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 27. Januar 2023

DEINE Unterschrift

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mich bei der Erstellung dieser Bachelorarbeit unterstützt und motiviert haben.

Zunächst möchte ich mich meiner Betreuer Mike Zashka für die hilfreichen Anregungen, konstruktive Kritik und vieles mehr während der Erstellung dieser Arbeit bedanken.

Besonderer Dank gebührt auch meinem Betreuer an der Hochschule Fulda, Prof. Dr. Nobert Ketterer, der mich richtungsweisend und mit viel Engagement während meiner Arbeit begleitet hat.

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

Die vorliegende Bachelorarbeit befasst sich mit der Low-Code Anwendungsentwicklung mit SAP AppGyver im Vergleich zur nativen SAP Fiori-Entwicklung, wobei der Fokus auf drei Technologien liegt: SAP AppGyver, Fiori Elements und SAPUI5. Anhand eines kundenorientierten Anwendungsfalles werden Anwendungen mit den drei Technologien entwickelt, um die drei Technologien zu betrachten und zu vergleichen. Eine Reihe von Funktionalitäten außerhalb des Anwendungsfalles wird ebenfalls untersucht, damit die drei Technologien eingehend bewertet werden können. Anschließend werden Bewertungsmatrizen definiert, Bewertungen durchgeführt und schließlich die Bewertungsergebnisse interpretiert und diskutiert. Mit Hilfe der Bewertungsergebnisse werden die Vor- und Nachteile der Technologie ermittelt.

Abstract

This bachelor thesis is dealing with low-code application development with SAP AppGyver compared to native SAP Fiori development, with the focus on three technologies: SAP AppGyver, Fiori Elements and SAPUI5. A customer-oriented use case will be used to develop applications with the three technologies for consideration and comparison. A set of functionalities outside of the use case will also be studied so that the three technologies can be evaluated in more depth. Then, evaluation matrices will be defined, evaluations will be carried out, and finally the evaluation results will be interpreted and discussed. With the evaluation results, the advantages and disadvantages of the technologies will be identified.

Inhaltsverzeichnis

Abbildungsverzeichnis	VII
Tabellenverzeichnis	IX
Quelltextverzeichnis	X
Abkürzungsverzeichnis	XI
1 Einleitung	1
1.1 Problemstellung und Motivation	1
1.2 Ziele der Arbeit	2
1.3 Beschreibung des Anwendungsfalls	3
1.4 Aufbau der Arbeit	4
2 Grundlagen	6
2.1 Entwicklung mit Low-Code/No-Code	6
2.2 Architektur von SAP-Anwendungen in der SAP Business Techno- logy Plattform	8
2.3 SAP Fiori	10
2.4 SAP AppGyver	11
2.4.1 Grundlage von AppGyver	11
2.4.2 Entwicklungsumgebung: Composer Pro	12
2.5 SAPUI5	16
2.5.1 Grundlage von SAPUI5	16
2.5.2 Entwicklungsumgebung: Visual Studio Code	17
2.6 Fiori Elements	19
2.6.1 Grundlage von Fiori Elements	19
2.6.2 Entwicklungsumgebung: Business Application Studio	21
3 Implementierung des Anwendungsfalls	22

3.1	Fiori Elements	23
3.1.1	Dev Space erstellen	23
3.1.2	Datenentitäten und Service definieren	24
3.1.3	User Interface erstellen	25
3.1.4	Deployment des OData-Services	27
3.2	AppGyver	29
3.2.1	Projektaufbau	29
3.2.2	OData Integration	29
3.2.3	Listenansicht zur Anzeige aller Produkte	30
3.2.4	Einzelansicht für ein Produkt	32
3.2.5	Maske zum Erstellen eines einzelnen Produkts	33
3.3	SAPUI5	35
3.3.1	Erstellung einer Anwendung mit dem UI5-Generator	35
3.3.2	OData Service einbinden	37
3.3.3	Listenansicht zur Anzeige aller Produkte	38
3.3.4	Einzelansicht für ein Produkt	41
3.3.5	Maske zum Pflegen eines einzelnen Produkts	41
4	Untersuchung weiteren Funktionen	45
4.1	Integration von Suchfilter	45
4.2	Integration von Paginierung	47
4.3	Integration von Bild- und PDF-Datei	49
4.4	Integration von Barcode-Scanner-Funktion	51
4.5	Nutzung nativer mobiler Funktionen	53
4.6	Möglichkeiten zum Deployment für unterschiedliche Endgeräte	53
4.7	Freie Gestaltungsmöglichkeit	54
5	Evaluation	57
5.1	Gegenüberstellung der drei Ansätze	57
5.2	JSON-Abfragesprache	60
5.3	Massen-Upload	62
	Literaturverzeichnis	65
A	JSON-Schema	71

Abbildungsverzeichnis

2.1	Architektur einer SAP-Anwendung in der SAP BTP	10
2.2	Pages einer AppGyver-Anwendung	13
2.3	Toolbar in Composer Pro	13
2.4	View-Components in Composer Pro	15
2.5	Exemplarische Formel in AppGyver	16
2.6	List Report und Object Page (Quelle: SAP Fiori Design Guidelines. URL: https://experience.sap.com/fiori-design-web/smart-templates)	19
2.7	Overview Page (Quelle: SAP Fiori Design Guidelines. URL: https://experience.sap.com/fiori-design-web/smart-templates)	20
3.1	Umsetzungsarchitektur der Technologie	23
3.2	Create a New Dev Space	23
3.3	Grundlegenden Aufbau inkl. der Cards von BAS	24
3.4	Datenmodell	24
3.5	Service	25
3.6	Sample Data	26
3.7	UI-Konfiguration	27
3.8	Listenseite, Detailseite und „neues Objekt anlegen- Seite der Fiori-Elements-Anwendung	28
3.9	Cloud Foundry Sign In und Targets	29
3.10	Destination-Konfiguration in SAP BTP Cockpit	30
3.11	OData-Integration in AppGyver	30
3.12	Data-Variable Logik	31
3.13	Listenseite und List Item mit Repeat in AppGyver	31
3.14	Logische Ablauffunktion für List Items	32
3.15	Wert zuweisen für App-Variable <i>appVarRecord</i> in „Set app variable “ Ablauf funktionskomponente	33
3.16	Logische Ablauffunktion für Edit-, Back- und Delete-Button	33

3.17	Logische Ablauffunktion für Save-Button	34
3.18	Listenseite, Detailseite und „Create new record“-Seite der AppGyver-Anwendung	35
3.19	Aussehen der UI5 Demo-Anwendung	37
3.20	Benutzeroberfläche der Listenansicht der Produkte	39
3.21	Benutzeroberfläche der Einzelansicht	42
3.22	Maske zum Pflegen eines einzelnen Produkts	44
4.1	Suchfilter in AppGyver	47
4.2	Seitenlogik von Paginierung in AppGyver	48
4.3	Paging definieren in Ablauffunktion „Get record collection“	48
4.4	Logik für PREV- und NEXT-Button in AppGyver	49
4.5	Benutzeroberfläche nach Paginierung in AppGyver	49
4.6	Image Binding in AppGyver	51
4.7	Logik für Load-PDF-Button in AppGyver	51
4.8	Logik für Scan QR/Barcode-Button in AppGyver	52
4.9	Gestaltung von Weather-App, Barcode-Scanner-App und Product-List-App	56
5.1	Testergebnisse zum Massen-Upload	63

Tabellenverzeichnis

1.1	Definition der Anwendungsfall	3
2.1	AppGyver-Baustein in die Schichten des MVC-Models	14
3.1	Umsetzungsanforderung der Technologie	22
5.1	Evaluationsmatrix für die Transformationsengine	58
5.2	Evaluationsmatrix für die JSON-Abfragesprache	61

Quelltextverzeichnis

A.1	JSON-Schema für die Mapping-Datei	71
-----	---	----

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 Wandel 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 befragten 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 Erbdinger, 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 eine 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 realen 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:	<ul style="list-style-type: none"> • SAP Fiori Elements mit SAP Business Application Studio (Backend + UI) • SAP AppGyver (UI) • SAPUI5 (UI)
Anforderungen Backend:	<ul style="list-style-type: none"> • Bereitstellung einer Datenbank-Entität Products mit folgenden Eigenschaften: <ul style="list-style-type: none"> – ID; title; materialNumber; description; price; stock • Bereitstellung eines OData-Services zum Auslesen, Erstellen und Aktualisieren (CRUD) der Produkte
Anforderungen Frontend:	<ul style="list-style-type: none"> • Funktionen: <ul style="list-style-type: none"> – Listenansicht zur Anzeige aller Produkte – Einzelansicht für ein Produkt – Maske zum Pflegen eines einzelnen Produkts • Datenanbindung: <ul style="list-style-type: none"> – Anbindung an den OData-Service zum Auslesen und Schreiben von Produkten • Look and Feel: <ul style="list-style-type: none"> – 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 LCNC-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-Plattformen für ihr jeweiliges Ökosystem zur Verfügung. Dazu einige Beispiele:

App Engine von ServiceNow wurde im März 2021 veröffentlicht [doc22g]. Es ermöglicht großen Unternehmen Low-Code-Anwendungen zu erstellen und bereitzustellen. 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 hinzuzufügen.

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-Plattform 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 Kauf 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 Plattform, 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.

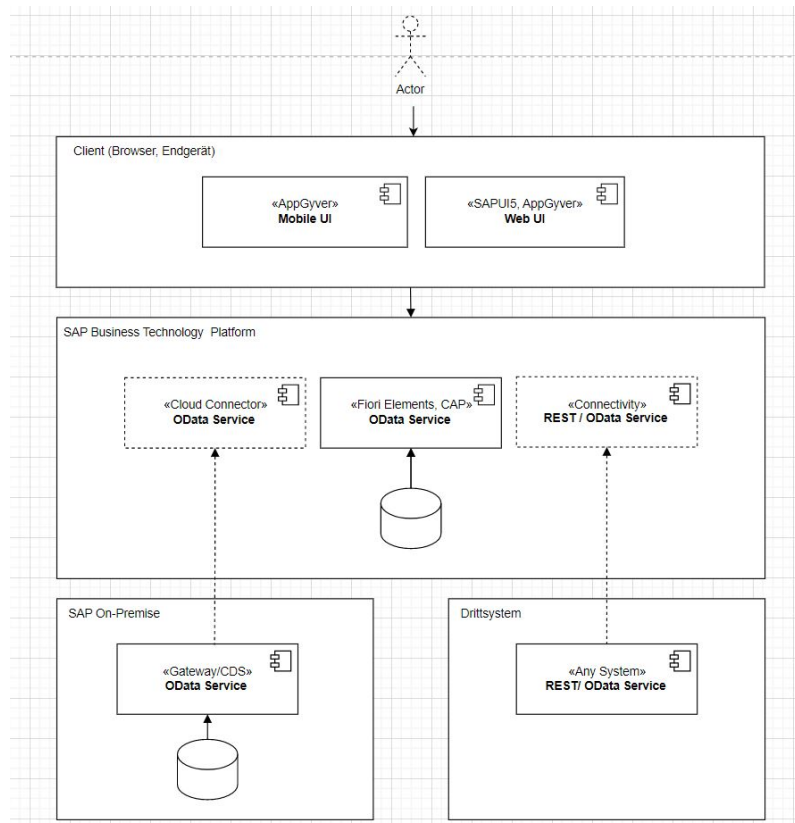


Abbildung 2.1: Architektur einer SAP-Anwendung in der SAP BTP

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 statuen 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 [App22d]. 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 Entwicklungsplattform 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.

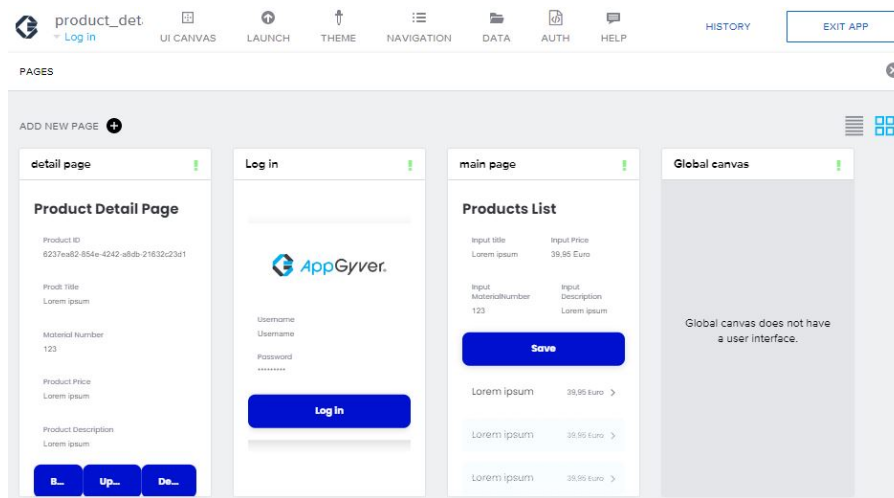


Abbildung 2.2: Pages einer AppGyver-Anwendung

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-Modells 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-Modells

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 [App22g]. 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

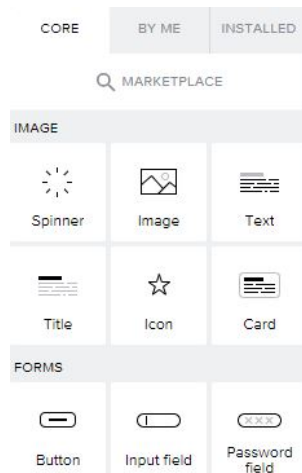


Abbildung 2.4: View-Components in Composer Pro

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) [App22b].

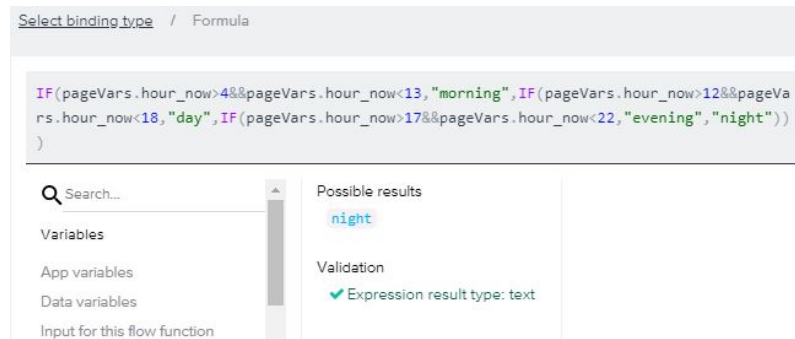


Abbildung 2.5: Exemplarische Formel in AppGyver

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 Kommanden 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 VS-Code-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.

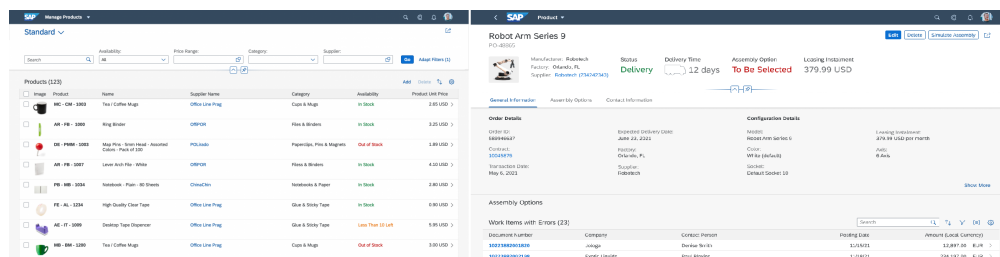


Abbildung 2.6: List Report und Object Page (Quelle: SAP Fiori Design Guidelines. URL: <https://experience.sap.com/fiori-design-web/smart-templates>)

- 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 [Doc22f].

- 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.

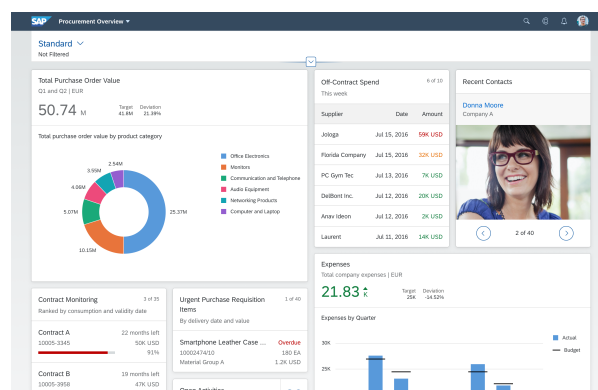


Abbildung 2.7: Overview Page (Quelle: SAP Fiori Design Guidelines. URL: <https://experience.sap.com/fiori-design-web/smart-templates>)

- 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 [Doc22f].

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 Definition von Datenstrukturen und OData-Services und Annotationen visuell aufgebaut werden [Doc22e].

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 [Por22d].

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 [Por22d].

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.

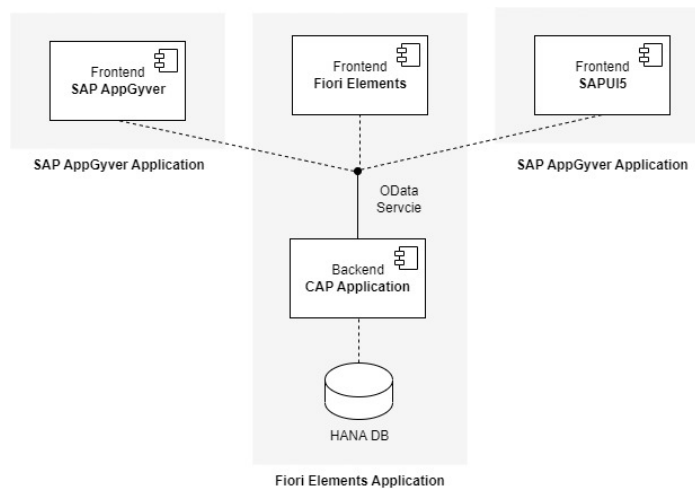


Abbildung 3.1: Umsetzungsarchitektur der Technologie

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.

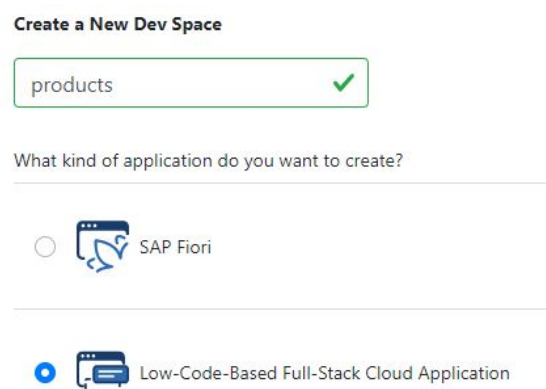


Abbildung 3.2: Create a New Dev Space

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.

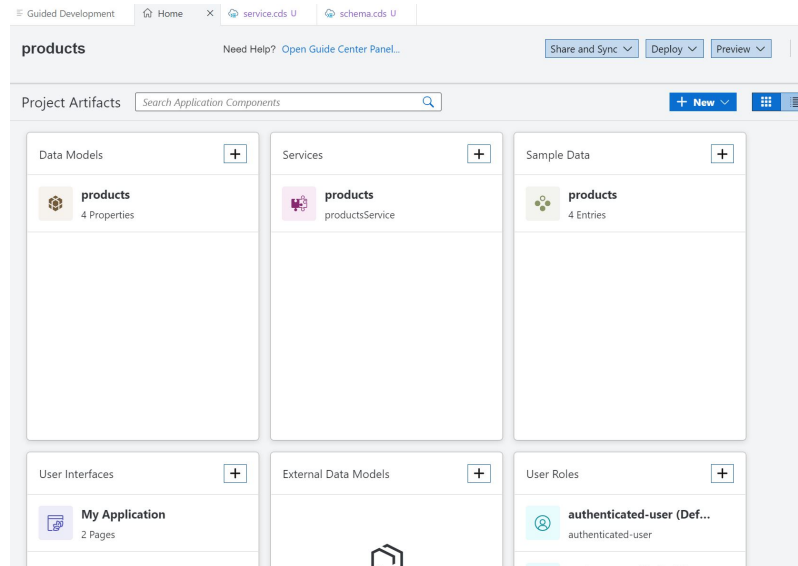


Abbildung 3.3: Grundlegenden Aufbau inkl. der Cards von BAS

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.


Products		
ID	UUID	
Title	String	
MaterialNumber	String	
Description	String	
Price	String	
Stock	String	

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.



Products Products.Products		
ID	UUID	
Ab Title	String	
Ab MaterialNumber	String	
Ab Description	String	
Ab Price	String	
Ab Stock	String	

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:

<input type="checkbox"/>	ID *	TITLE	MATERIALNUMBER	DESCRIPTION	PRICE	STOCK
<input type="checkbox"/>	1 0165cdd0-f1aa-4c62...	Carpendo car seat co...	252	Black front seats and...	40.19 Euro	STOCK2786
<input type="checkbox"/>	2 f3e89493-d10a-4d32...	Carpendo Auto Spor...	338	Seat covers black-gr...	46.99 Euro	STOCK5817
<input type="checkbox"/>	3 b1200d4e-e514-47ef...	Woltu car seat covers	111	Black, with "Super D...	26.99 Euro	STOCK9274
<input type="checkbox"/>	4 e209033f-5993-4113...	Walsen Comfort Car ...	760	Car seat cover S-Rac...	16.95 Euro	STOCK256
<input type="checkbox"/>	5 c27cb5cb-3ad2-42ec...	Walsen car seat cover...	949	Universal seat cover ...	74.95 Euro	STOCK4803
<input type="checkbox"/>	6 f798ef12-f5a2-480b...	Flying Banner Car Se...	872	Universal Set with Ai...	29.99 Euro	STOCK3350
<input type="checkbox"/>	7 87de15d1-221b-432f...	Flying Banner PVC C...	848	Complete set with ai...	34.39 Euro	STOCK5088
<input type="checkbox"/>	8 a31f73e0-be2c-47d3...	Flying Banner Univer...	757	Complete Cover Brea...	34.99 Euro	STOCK5150
<input type="checkbox"/>	9 0094daed-09f7-4557...	Upgrade4Cars car se...	641	Seat covers complet...	39.95 Euro	STOCK961
<input type="checkbox"/>	10 071d0a34-c4e4-427...	Upgrade4Cars car se...	110	Car seat cover for sp...	13.95 Euro	STOCK778
<input type="checkbox"/>	11 31901267-a58f-49b4...	Fixcape PRO car seat...	904	Car seat cover for fro...	15.90 Euro	STOCK4737
<input type="checkbox"/>	12 22407997-be2a-40b...	Fixcape Neoprene C...	919	Car seat cover for th...	29.50 Euro	STOCK1005
<input type="checkbox"/>	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 2.6.1 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-

The screenshot shows a 'Create New UI Application' wizard. On the left, a vertical list of steps is shown: 'UI Application Details' (selected), 'UI Application Type', 'UI Application Template', and 'Data Objects'. The main area on the right is titled 'UI Application Details' and contains the instruction 'Fill in basic information of your application'. It has three input fields: 'Display name' with the value 'Products', 'Application name' with the value 'Products', and 'Description' with the value 'My SAP application'.

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 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

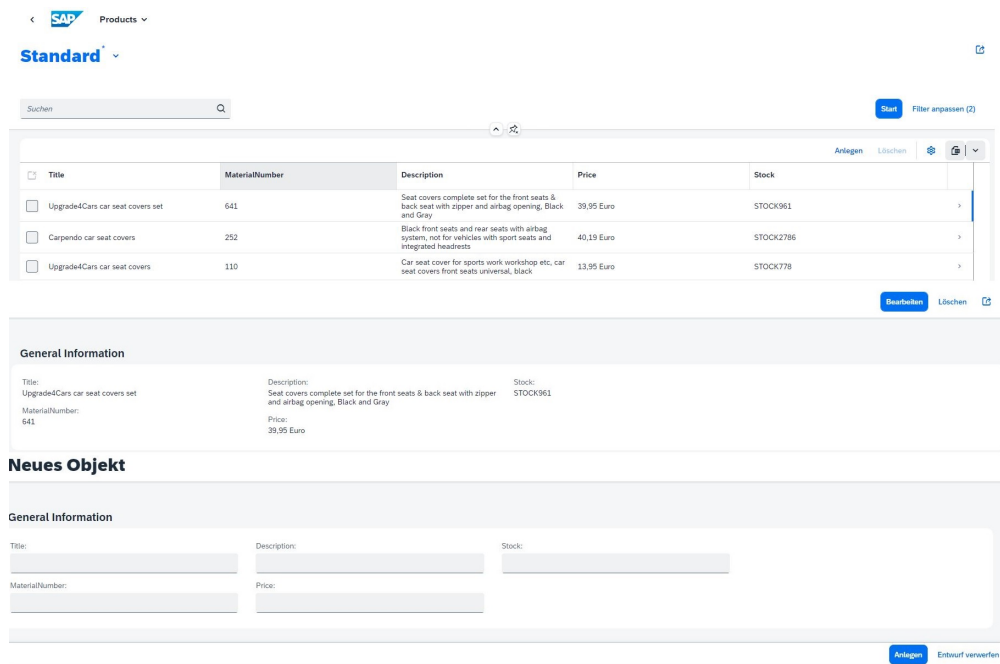


Abbildung 3.8: Listenseite, Detailseite und „neues Objekt anlegen“- Seite der Fiori-Elements-Anwendung

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 Anwenderschnittstelle 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

Cloud Foundry Sign In and Targets

Provide your Cloud Foundry parameters to sign in to the Cloud Foundry environment

Cloud Foundry Sign In

Enter Cloud Foundry Endpoint *

https://api.cf.eu20.hana.ondemand.com

Select authentication method ⓘ

☒ Credentials ☐ SSO Passcode

Enter your username *

fangfang.tan@p36.io

Enter your password *

Sign in

Cloud Foundry Target

Select Cloud Foundry Organization *

7458bcdtrial

Select Cloud Foundry Space *

dev

Apply

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 [Por22c]. 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.

Destination Configuration

Name: products-on-trial_simple

Type: HTTP

Description:

URL: https://7458bcdtrial-dev-products-srv.cfapps.us10.hana....

Proxy Type: Internet

Authentication: OAuth2ClientCredentials

Use mTLS for token retrieval: ☐

Client ID: sb-Products-devt97003

Client Secret: *****

Token Service URL Type: Dedicated Common

Token Service URL: https://7458bcdtrial.authentication.us10.hana.ondeman...

Token Service User:

Token Service Password:

Additional Properties

AppgyverEnabl...: true

HTML5.Dynam...: true

☒ Use default JDK truststore

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.

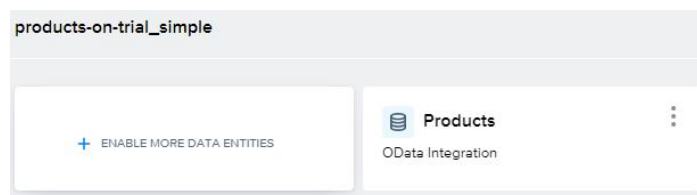


Abbildung 3.11: OData-Integration in AppGyver

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 Kapitel 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 Ablauf funktionskomponente „Get record collection “ bezogen, wobei die Daten hier nur als ein Ausgangsargument des Knotens existieren. Dann wird den „Set data variable “-Ablauf funktionskomponente verwendet, damit die Daten in die Data-Variable setzen können. Nun steht die Data-Variable *Product1* für die Verwendung in View-Component Binding zur Verfügung.

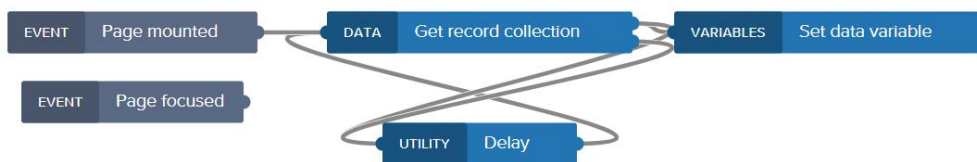


Abbildung 3.12: Data-Variable Logik

Zur Anzeige der Produktliste muss die View-Component „List item “ mit dem erstellten Data-Variable *Product1* verbunden und wiederholt werden. Das Primary und das Secondary Label bestimmen, welcher Property-Wert in der Liste angezeigt werden soll. In dieser AppGyver-Anwendung sollen der Produkttitel und der Produktpreis angezeigt werden. Deshalb sollten das Primary Label mit dem Wert *current.Title* und das Secondary Label mit *current.Price* verbunden werden. Nach dem Speichern wird dann die Liste mit allen Produkten angezeigt.

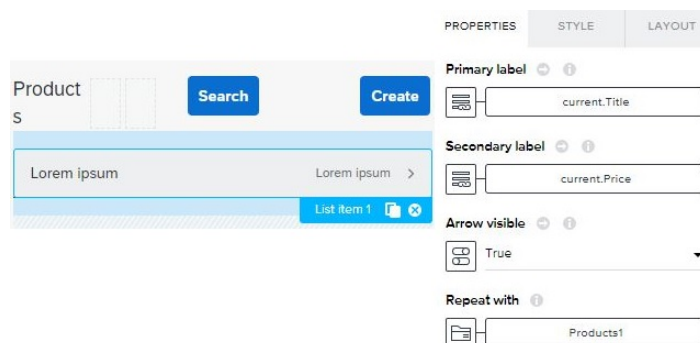


Abbildung 3.13: Listenseite und List Item mit Repeat in AppGyver

3.2.4 Einzelansicht für ein Produkt

Die Einzelansicht enthält alle Properties zu einem Produkt, nämlich den Produkttitel, die Beschreibung, die Materialnummer, den Preis, den Bestand und die Produkt-ID. Für die Darstellung der Informationen muss eine Detailseite mit dem Namen „Product Details“ erstellt werden. Eine App-Variable *appVarRecord* mit sechs Properties wird erstellt, damit die Daten an verschiedene Seiten der Anwendung übertragen werden können.

Um von der Listenseite zur Detail-Seite zu navigieren, muss die logische Ablauffunktion für das List Item in der Listenseite aufgebaut werden, dabei geht es um ein Component tap Event. Das Event wird ausgelöst, wenn das List Item vom Benutzer angeklickt wird. Der Aufbau der logischen Ablauffunktion kann Abbildung 3.14 entnommen werden.

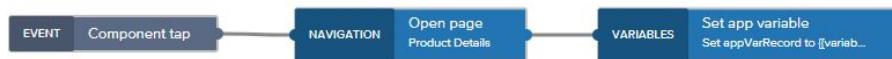


Abbildung 3.14: Logische Ablauffunktion für List Items

Eine „Open page“-Ablauffunktionskomponente ist erforderlich, um die Detailseite zu öffnen. Die Ausgangspunkte der Ablaufunktionskomponente „Open page“ sind mit der Ablaufunktionskomponente „Set app variable“ verknüpft. Mit dieser Komponente werden die Daten in dem List Item mit der neu angelegten App-Variable verbunden. Die Werte der Properties von der App-Variable werden mit den jeweiligen Property-Werten in Wiederholung von List Item gesetzt. Dabei handelt es sich um die Werte der Data-Variablen *Products1*. Da die Werte der Date-Variablen *Product1* mit dem OData Service verbunden wurden, wird der Wert aus dem Backend in die App-Variable eingesetzt.

Auf der Detailseite werden sechs Input-Felder erstellt. Die Werte der Input-Felder sind mit dem entsprechenden Property-Wert der App-Variablen *appVarRecord* verbunden, nämlich dem Titel, der Materialnummer, dem Preis, der Produkt-ID und der Beschreibung sowie dem Bestand. Außerdem werden drei Buttons für 3 Component tap Events auf der Detailseite erstellt: Der „Back“-Button wird verwendet, um zurück zur Listenseite zu navigieren. Um dies zu implementieren, muss lediglich eine Logikablauffunktion mit der Komponente „Navigation back“ erstellt werden (vgl. Abb. 3.16). Der „Edit“-Button wird zum Aktualisieren der Pro-

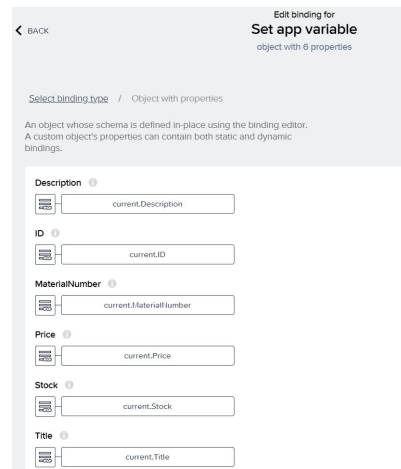


Abbildung 3.15: Wert zuweisen für App-Variable *appVarRecord* in „Set app variable“ Ablauf funktionskomponente

duktinformationen genutzt. Dazu ist die in Abbildung 3.16 dargestellte Ablauf funktionskomponente mit der Komponente „Update record“ erforderlich. Die Ablauf funktionskomponente ist mit der Datenressource Products (vgl. Abb. 3.11) und der App-Variablen *appVarRecord* verknüpft. Die „Update record“-Komponente hat zwei Ausgänge: Der obere ist mit einer „Alert“-Ablauf funktionskomponente verlinkt, die zur Anzeige von „Update Record ID“ dient, der untere ist mit einer „Toast“-Ablauf funktionskomponente verbunden, die nur benutzt wird, wenn die Aktualisierung des Datensatzes nicht erfolgreich abgeschlossen werden kann. Der „Delete“-Button in Kombination mit der „Delete record“-Komponente wird zum Löschen eines Produkts eingesetzt.

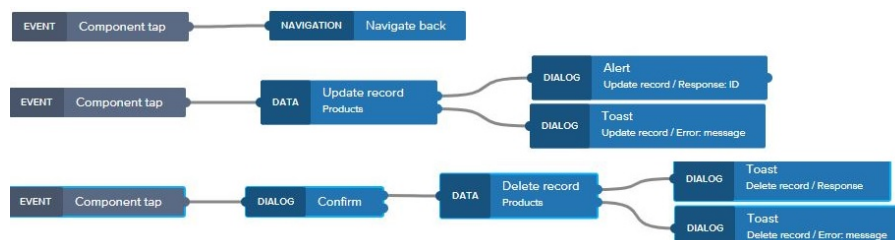


Abbildung 3.16: Logische Ablauf funktionskomponente für Edit-, Back- und Delete-Button

3.2.5 Maske zum Erstellen eines einzelnen Produkts

Zunächst muss eine Seite mit dem Namen „Create New Record“ erstellt werden. Diese Seite besteht aus fünf Input-Feldern, einem „Save“-Button

sowie einem „Back“-Button. Eine neue Data-Variable mit dem Variablentyp „New data record“ wird erstellt, die ebenfalls mit der Data-Ressource Products verbunden wird. Data-Variable mit dem Typ „New data record“ initialisiert eine leere Data-Variable, die beim Erstellen eines Formulars zum Anlegen von Datensätzen im Backend zu verwenden ist. Die Werte der Input-Felder werden mit den entsprechenden Property-Werten der Data-Variablen verknüpft. Da die Produkt-ID als UUID definiert ist und vom System generiert wird, muss das Input-Feld dafür nicht erstellt werden.

Um den Datensatz speichern zu können, wird eine Ablauffunktion mit der Komponente „Create record“ für den „Save“-Button benötigt. Ähnlich wie die Ablauffunktionskomponente „Update record“ ist die Komponente „Create record“ mit der Datenressource Products und der App-Variablen appVarRecord verbunden. Sie hat ebenfalls zwei Ausgänge: einen für die Bestätigung der erfolgreichen Erstellung von Datensätzen und einen anderen für die Anzeige des Fehlers (vgl. Abb. 3.17). Der „Back“-Button kann für die Navigation zurück zur Listenseite genutzt werden.



Abbildung 3.17: Logische Ablauffunktion für Save-Button

Wie bereits in Kapitel 2.4.2 vorgestellt, lassen sich die AppGyver-Baustein in 3 Schichten des MVC-Modells unterteilen, nämlich in View-, Controller- und Model-Schichten. Auf der View-Schicht wurden für den Anwendungsfall insgesamt 3 Seiten- und 21 View-Komponenten erstellt, darunter 3 Seitentitel, ein List Item, 11 Input-Felder sowie 7 Buttons. Damit die Buttons und das List Item dynamisch funktionieren, wurden in der Controller-Schicht 8 logische Ablauffunktionen mit Komponente tap Event erstellt. In der Modelschicht wurden 2 Daten-Variablen und eine App-Variable erstellt, um die Daten vom Odata-Service zu holen, die Daten zu aktualisieren und sie an verschiedene Seiten zu übertragen.

Ein wichtiges Konzept im AppGyver ist die Binding von Komponenten an Daten, die in Variablen gespeichert sind. Mit nur wenigen Klicks kann beispielsweise eine View-Component Property direkt mit einer Data-Variable in Binding treten. Die Binding wird automatisch aufgelöst, wenn neue Daten aus dem Backend abgerufen und in der Data-Variable gespeichert werden. Die View-Component Property wird aktualisiert und die neuen

Daten wird in View angezeigt [App22c].

The List of Products		
Products	Search	Create
Upgrade4Cars car seat covers set 2000		150,95 Euro >
Carpendo car seat covers		40,19 Euro >
Upgrade4Cars car seat covers		19,95 Euro >

Product Details	Create New Record
<div>Product Title</div> <div>Upgrade4Cars car seat covers set 2000</div> <div>Product description</div> <div>Seat covers complete set for the front seats & back seat with zipper and rolling opening. Black and Gray</div> <div>Material number</div> <div>845</div> <div>Price</div> <div>150,95 Euro</div> <div>Units</div> <div>each/1000</div> <div>Product ID</div> <div>8094460-0001-0001-0001-0001-0001-0001</div> <div>Back</div>	<div>Create a new record</div> <div>Title</div> <div>Type Text...</div> <div>Essential number</div> <div>Type Text...</div> <div>Description</div> <div>Type Text...</div> <div>Price</div> <div>Type Text...</div> <div>Stock</div> <div>Type Text...</div> <div>Save</div> <div>Back</div>

Abbildung 3.18: Listenseite, Detailseite und „Create new record“-Seite der AppGyver-Anwendung

3.3 SAPUI5

3.3.1 Erstellung einer Anwendung mit dem UI5-Generator

In Kapitel 2.5 wurden SAPUI5 und die Einrichtung der Entwicklungsumgebung für die Implementierung von SAPUI5-Anwendungen kurz vorgestellt. In diesem Abschnitt wird nun die Erstellung der SAPUI5-Applikation näher beschrieben. Mit Hilfe des `easy-ui5`-Generators lässt sich ein grundlegendes SAPUI5-Anwendungsgerüst erstellen. Da die Entwicklung der Applikation auf dem neuesten Stand der Technik erfolgen soll, wird TypeScript für das Coding verwendet. TypeScript ist eine Design-Time-Programmiersprache, die JavaScript um eine starke Typisierung erweitert. Das UI5-Typescript-Tooling nutzt den Babel-Compiler (JavaScript Compiler), um den TypeScript-Code in ausführbaren JavaScript-Code umzuwandeln [Mue22]. Dabei läuft ein Prozess im Hintergrund, der die Quellen im `src`-Ordner überwacht und die transpilierten Dateien in den `webapp`-Ordner kopiert.

Zur Erzeugung des Applikations-Gerüsts wird der Befehl *yo easy-ui5 ts-app* in der Command-Prompt-Konsole ausgeführt. Dieser Befehl führt dazu, dass eine UI5-TypeScript-Anwendung basierend auf den folgenden Parametern erstellt wird:

```
1 ? How do you want to name this application? products
2 ? Which namespace do you want to use? fanfan.products
```

```
3 ? Which framework do you want to use? SAPUI5
4 ? Which framework version do you want to use? 1.108.4
5 ? Who is the author of the application? Tan Fangfang
6 ? Would you like to create a new directory for the application? Yes
```

Nach Eingabe der Parameter müssen noch alle externen Bibliotheken via npm installiert werden und dann kann die Entwicklung beginnen.

Der Generator erzeugt die grundlegende Projektstruktur eines SAPUI5-Projekts. Das MVC-Pattern findet sich auch im Dateisystem wieder. So liegen die Controller im Verzeichnis controller, die Views im gleichnamigen Verzeichnis und es ist ebenfalls ein model-Verzeichnis vorgesehen. Dies wird jedoch im Projekt nicht verwendet, sondern es werden auf Standard Odata- und JSONModel zurückgegriffen [Doc22a].

Daneben sind noch eine Reihe an Konfigurationsdateien vorhanden, wie package.json oder manifest.json:

```
1 project-root
2 |- src
3   |- controller
4     |- App.controller.ts
5     |- BaseController.ts
6     |- Main.controller.ts
7   |- i18n
8   |- model
9   |- view
10    |- App.view.xml
11    |- Main.view.xml
12  |- Component.ts
13  |- manifest.json
14  |- index.html
15  |- [...]
16  |- package.json
17  |- README.md
18  |- ui5.yaml
```

Die Datei „manifest.json“ enthält die Grundkonfiguration der Anwendung, die für die Ausführung benötigt wird. Alle Modelle, Bibliotheken und Komponenten, die in der „manifest.json“-Datei konfiguriert sind, werden beim Starten der Anwendung automatisch geladen und instanziiert. Die „rootView“- und „routing“-Konfigurationen definieren die Hauptansicht

der Anwendung, die nach Öffnen im Browser angezeigt wird, und die Navigation zwischen den Ansichten.

In der Datei „App.view.xml“ wird die Benutzeroberfläche der Hauptansicht der Anwendung definiert. SAPUI5 unterstützt mehrere View-Typen (XML, HTML, JavaScript, JSON). In diesem Projekt wird das XML-Format verwendet. Für jede Ansicht, die in der Anwendung verwendet wird, soll eine eigene View-Datei erstellt werden, und jede View wird von einem separaten Controller gesteuert.

Das Öffnen der Webanwendung erfolgt durch Aufrufen der index.html-Seite über einen Webserver im Browser. Das UI5-Tooling bringt bereits einen lokalen Webserver mit, der via `npm run start` gestartet werden kann. Abbildung 3.19 zeigt das Aussehen der erzeugten Anwendung, die bereits erste Inhalte hat, die angepasst und ersetzt werden müssen. Während die Anwendung läuft, kann der Code in VS-Code geändert und die Änderungen direkt danach direkt im Browser angesehen werden.

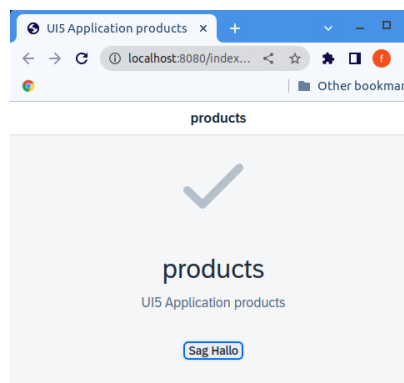


Abbildung 3.19: Aussehen der UI5 Demo-Anwendung

In den nächsten Abschnitten wird die Implementierung der UI5-Anwendung nun im Detail beschrieben.

3.3.2 OData Service einbinden

In Kapitel 3.2.1 wurde die Bereitstellung der OData-Services und der Destination in der SAP-BTP-Umgebung vorgestellt. Die SAPUI5-Anwendung muss diese Destination ebenfalls integrieren, um die OData-Services hinter der Destination konsumieren zu können.

Leider lässt sich der OData-Server lokal nicht einfach im Browser einbinden, da dies zu einem Cross-Origin-Resource-Sharing (CORS)-Problem

führt. Die Anwendung läuft unter der Domäne „localhost“ und moderne Browser erlauben es nicht, Daten via XMLHttpRequest von einer anderen Domäne zu laden (es sei denn, diese sendet entsprechende http-Header mit) [Doc22d]. Um dieses Problem zu vermeiden, wird der Request über einen lokalen Reverse-Proxy (SAP Approuter) geleitet, der über das UI5-Tooling in das Projekt integriert werden muss. Dies ist zwar in wenigen Konfigurationsschritten durchführbar, es zeigt aber bereits, dass auch gleich funktionierende Dinge (Zugriff auf die OData-Services via Destination) in SAPUI5 aufwendiger zu implementieren sind und die beiden LCNC-Toolings (Fiori Elements mit CAP und SAP AppGyver) derlei Dinge vom Entwickler fernhalten.

Der Zugriff auf die Daten in SAPUI5 erfolgt auf Basis eines ODataModels. Zwar ließen sich die Daten auch manuell als JSON abrufen, die Nutzung des ODataModels bietet jedoch den Vorteil, dass dies die http-Kommunikation komplett vom Entwickler fernhält. Zudem bietet das ODataModel weitere Features wie Databinding an, mit dem Daten sehr einfach an View-Controls gebunden und dargestellt werden können.

Das ODataModel wird in der Component.ts-Datei erstellt und damit beim Starten der Anwendung initial in allen Controllern und Views bekannt gemacht.

```
1 const oData = new ODataModel({
2   serviceUrl: "public/",
3   synchronizationMode: "None",
4   operationMode: "Server",
5 });
6 this.setModel(oData);
```

[

3.3.3 Listenansicht zur Anzeige aller Produkte

Die Listenansicht soll alle Produkte in einer Liste darstellen und es dem User ermöglichen, in die Einzelansicht eines Produkts zu springen. Da die Applikation nach dem Erstellen bereits über eine View verfügt, wird dies in der Main-View realisiert. Zentrales Element bildet die List-Control, welche Daten in einer Liste darstellen kann. Abbildung 3.20 zeigt die Benutzeroberfläche der Listenansicht der Produkte.

SAPUI5-Controls sind UI-Elemente, die das Aussehen und Verhalten kap-

Products Information

The list of the Products	Scan	Create	SearchProduct	Q
Upgrade4Cars car seat covers	13,95 Euro	>	X	
Leader Accessories universal car seat cover	24.99 Euro	>	X	
Woltu car seat covers	26,99 Euro	>	X	
Carpendo car seat covers	40,19 Euro	>	X	
Fixcape Neoprene Car Seat Covers	29,50 Euro	>	X	
Flying Banner PVC Car Seat Covers	34.39 Euro	>	X	
Upgrade4Cars car seat covers set	39,95 Euro	>	X	
Walser car seat cover Rey	74,95 Euro	>	X	
Weitere				

Abbildung 3.20: Benutzeroberfläche der Listenansicht der Produkte

seln. In SAPUI5 stehen mehr als 100 Controls wie z.B. Buttons, Inputs, Texte, Tabellen, Messages, Dialoge, Listen usw. zur Verfügung. Eine Control kann andere Controls aggregieren und diese dient dann als Container. Die List-Control ist ein Container für entsprechende Listenelemente. Die Steuerung der View wird im entsprechenden Controller „Main.controller.ts“ implementiert.

```

7 <List mode="Delete" delete="onDeletePress" items="{path: '/Products'}" id="
  productsList">
8   <headerToolbar>
9     <OverflowToolbar width="100%">
10    <Title text="The list of the Products" />
11    <ToolbarSpacer />
12    <Button icon="sap-icon://camera" text="Scan" press="onScanNavi"/>
13    <Button icon="sap-icon://add" text="Create" ariaHasPopup="Dialog" press="
      pressAddInList" />
14    <SearchField width="200px" placeholder="SearchProduct" search="
      mySearchProduct" />
15    </OverflowToolbar>
16  </headerToolbar>
17  <items>
18    <StandardListItem title="{title}" type="Navigation" info="{price}" press="
      onItemPress"/>
19  </items>
20 </List>

```

[

Das Coding zeigt den Ausschnitt der Listendefinition. Jede Control verfügbar über bereitgestellte Eigenschaften (Properties) und Events, die

im XML angegeben werden können. Mit der Eigenschaft *mode="Delete"* wird beispielsweise automatisch ein Lösch-Button in der Liste hinzugefügt. Das Event *delete="onDeletePress"* verweist dahingehend dann auf die Callback-Funktion *onDeletePress* in der Klasse „*Main.controller.ts*“, welche das Löschen dann programmatisch vornehmen muss:

```
1 public onDeletePress(event: UI5Event) {  
2     const listItem = event.getParameter("listItem") as ColumnListItem;  
3     const context = listItem.getBindingContext() as Context;  
4     context.delete();  
5 }
```

In dieser Methode zeigt sich, dass in SAPUI5 durch die Models und das Databinding weder die Daten direkt aus den UI-Controls abrufen noch direkt mit dem OData-Service kommuniziert werden muss. Es wird an dieser Stelle lediglich der BindingContext (die Information zum Binding selber) benötigt und diese aus dem Model gelöscht. Das sorgt dann automatisch dafür, dass das Model einen DELETE-Request zum OData-Service sendet und das Produkt aus der Datenbank entfernt wird.

Gleiches gilt auch für das initiale Setzen des Bindings. In obigem Coding zur Listenansicht ist zu sehen, dass in der Liste lediglich ein *StandardListItem* definiert wird. Dies dient dabei als Template für alle Produkte, die aus dem OData-Service ausgelesen werden. Über das Aggregation-Binding der Aggregation items (*items="{path: '/Products'}"*) werden alle Produkte, sich in unter dem Pfad abrufen lassen gebunden und für jedes Produkt wird ein „*StandardListItem*“ erzeugt, das den Titel und Preis anzeigt [Doc22c].

```
1 <StandardListItem title="{title}" type="Navigation" info="{price}" press="  
    onItemPress"/>
```

Ein weiteres wichtiges Event ist das *press*-Event eines *StandardListItems*. Klickt ein User auf ein Produkt, so soll er auf die Dateiseite weitergeleitet werden. Das zugehörige Coding zeigt, dass an dieser Stelle auf den Router zugegriffen wird. In SAPUI5 sorgt der Router dafür, dass URL-Pfade auf Views/Controller gemappt werden und ermöglicht es, dass das Framework im Hintergrund die notwendigen Controller/Views instanziiert.

```
1 public onItemPress(event: UI5Event){  
2     var item = event.getSource();
```

```
3   var context = item.getBindingContext();
4   var path = context.getPath().substr(1);
5   this.navTo("detail", {path: path});
6 }
```

Auch hier wird auf das Binding zurückgegriffen, um Informationen zum Produkt auszulesen, die an die Detailseite übergeben werden.

3.3.4 Einzelansicht für ein Produkt

Damit die Navigation funktioniert, müssen für die Einzelansicht ein View („Detail.view.xml“) und ein Controller erstellt werden. In der View werden alle Merkmale, wie der Name, die Beschreibung, die Materialnummer, der Preis und der Bestand eines Produkts in den Input Controls angezeigt.

```
1 <Label text="Title" />
2 <Input value="{title}" width="1000px" />
3 <Label text="Description" />
4 <Input value="{description}" width="1000px" />
5 <Label text="Material Number" />
6 <Input value="{materialNumber}" width="1000px" />
7 <Label text="Price" />
8 <Input value="{price}" width="1000px" />
9 <Label text="Stock" />
10 <Input value="{stock}" width="1000px" />
```

Damit die Werte angezeigt werden können, wird in der View wieder ein Binding verwendet (ElementBinding), dass ein komplettes Produkt an die View bindet. Dann ist es ausreichend, lediglich die Attribute zu benennen und das ODataModel lädt die relevanten Daten automatisch.

Abbildung 3.21 kann die finale Benutzeroberfläche der Einzelansicht für ein Produkt entnommen werden.

3.3.5 Maske zum Pflegen eines einzelnen Produkts

Für die Zusammenstellung der Maske zur Pflege eines Produkts werden Label-, Input- und Button-Controls und das Element „Fragment“ verwendet. Fragmente sind leichtgewichtige UI-Komponenten, die wiederverwendbar sind, aber keinen Controller haben [Doc22b]. Daher kann ein Fragment als ein Container für andere UI5-Controls betrachtet werden. Das UI-Design

< Product Detail Page

Title	Upgrade4Cars car seat covers
Description	Car seat cover for sports work workshop etc, car seat covers front seats universal, black
Material Number	110
Price	13,95 Euro
Stock	Stock110

Abbildung 3.21: Benutzeroberfläche der Einzelansicht

dieses Fragments ist in der Datei „createProductDialog.fragment.xml“ gespeichert.

```

1 <content>
2   <VBox class="sapUiMediumMargin">
3     <Label text="Title" />
4     <Input value="{listInputModel}/recipient/title}" width="200px" />
5     <Label text="Material Number" />
6     <Input value="{listInputModel}/recipient/materialNumber}" width="200px" />
7     <Label text="Description" />
8     <Input value="{listInputModel}/recipient/description}" width="200px" />
9     <Label text="Price" />
10    <Input value="{listInputModel}/recipient/price}" width="200px" />
11    <Label text="Stock" />
12    <Input value="{listInputModel}/recipient/stock}" width="200px" />
13  </Vbox>
14 </content>
15 <beginButton>
16   <Button text="Cancel" ariaHasPopup="Dialog" press="closeDialog" />
17 </beginButton>
18 <endButton>
19   <Button text="Add" ariaHasPopup="Dialog" press="pressAddInProductsList" />
20 </endButton>

```

In der Fragment-Implementierungsdatei werden mehrere Eingabefelder und zwei Buttons, „Cancel“ und „Add“ erstellt. Wird das press-Event des „Add“-Buttons geworfen, so wird die Methode *pressAddInProductsList* aufgerufen, um den Datensatz, der in das Eingabefeld eingegeben wurden, in den OData-Service zu übernehmen. Die Eingabewerte aller Eingabefelder werden zunächst via Binding an ein lokales JSON-Model übergeben. In

der *pressAddInProductsList*-Methode werden die Werte aus dem lokalen Model ausgelesen und ein neues OData-Binding erzeugt, dass die Daten automatisch an den OData-Endpoint sendet.

```
1 public pressAddInProductsList() {
2     var title = this.getView().getModel("listInputModel").getProperty("/recipient/
        title");
3     var id = this.getView().getModel("listInputModel").getProperty("/recipient/id"
        );
4     var description = this.getView().getModel("listInputModel").getProperty("/
        recipient/description");
5     var materialNumber = this.getView().getModel("listInputModel").getProperty("/
        recipient/materialNumber");
6     var price = this.getView().getModel("listInputModel").getProperty("/recipient/
        price");
7     var stock = this.getView().getModel("listInputModel").getProperty("/recipient/
        stock");
8
9     const context = (<ODataListBinding>(
10         this.getView().byId("productsList").getBinding("items")
11     )).create({
12         title: title,
13         id: id,
14         description: description,
15         materialNumber: parseInt(materialNumber),
16         price: price,
17         stock: stock,
18     });
19     this.closeDialog();
20 }
```

Da das Formular zum Anlegen in einem Dialog geöffnet wird, muss dieser nach dem Absenden dann auch wieder geschlossen werden. Abbildung 3.22 zeigt das Dialogfenster zum Einfügen eines neuen Produkts.

The image shows a mobile application interface. In the foreground, a modal window titled 'Products Information' is open, allowing for the editing of a product. The modal contains the following fields:

- Title:** Carpendo car seat covers
- Material Number:** 101
- Description:** Black front seats and rear ...
- Price:** 40,19 Euro
- Stock:** Stock101

At the bottom of the modal are two buttons: 'Cancel' and 'Add'.

In the background, a list of products is visible. The list has columns for product names, prices, and icons for editing or deleting items. The visible product names include 'Upgrade4Cars', 'Leader Accessor', 'Carpendo car s', 'Woltu car seat c', 'Carpendo car s', 'Fixcape Neopre', 'Flying Banner P', and 'Upgrade4Cars'.

Abbildung 3.22: Maske zum Pflegen eines einzelnen Produkts

Kapitel 4

Untersuchung weiteren Funktionen

In Kapitel 3 wurden Funktionen beschrieben, die im Rahmen dieser Arbeit technisch umgesetzt wurden. Dieses Kapitel betrachtet weitere Standard-Anforderungen an Webanwendungen im SAP-Kontext und Funktionalitäten von Fiori Elements, SAP AppGyver und SAPUI5, ohne praktische Umsetzung. Diese Funktionen umfassen:

- Integration von Suchfiltern und Paginierung.
- Integration von Bildern und PDF-Dateien.
- Integration einer Barcode-Scanner-Funktionen.
- Nutzung weiterer mobiler Funktionen.
- Möglichkeiten zum Deployment für unterschiedliche Endgeräte
- Freie Gestaltungsmöglichkeiten in der UI

4.1 Integration von Suchfilter

In Fiori Elements ist es nicht notwendig, die Suchfilter-Funktion zusätzlich zu implementieren, da sie bereits in der List Report Vorlage vorhanden ist. Das heißt, wenn die Benutzeroberfläche in Fiori Elements mit dem List Report Floorplan erstellt wird, ist die Suchfilter-Funktion automatisch integriert. Über die Annotationen ist es möglich, Suchfilter genauer zu beschreiben oder gezielt ein- und auszublenden. Die Erstellung der Benutzeroberfläche wurde bereit in Abschnitt 3.1.3 dargestellt.

In AppGyver ist die Integration eines Suchfilters durch das manuelle Hinzufügen von Bausteinen möglich. Dabei ist jedoch zu erwähnen, dass der Filter komplett client-seitig funktioniert und zunächst alle Daten vom Backend geladen werden. Das sorgt dafür, dass bei großen Datenmengen längere Ladezeiten entstehen können. Die Implementierung in AppGyver ist wie folgt:

Ein List Item wurde bereits in Kapitel 3.2.2 mit der angelegten Daten-Variable „Product1 “ verbunden und wiederholt. Mit der Ablauffunktionskomponente „set app Variable “ wurde der List Item ebenfalls mit die in Abschnitt 3.2.2 erstellte App-Variable eingebunden. Für den Filter wird ein Input-Field per Drag and Drop in der Benutzeroberfläche integriert. Da die Suche über den Produkttitel erfolgen soll, wird der Wert des Input-Fields an die App-Variable Property „title “ gebunden. In den Advanced Properties des List Items befindet sich ein Visible-Flag, mit dem die Sichtbarkeit des List Items eingestellt werden kann. Der Schalter kann auf true oder false gesetzt werden. Bei false wird das List Item nicht gerendert. Über dieses Flag kann der Filter implementiert werden: Eine Contains-Formel wird an die Visible-Eigenschaft gehangen, die true zurückgibt, wenn der Text den angegebenen Textabschnitt enthält. Andernfalls gibt die Formel false zurück und das Produkt wird ausgeblendet. Um die Groß- und Kleinschreibung nicht zu berücksichtigen, können alle Zeichenketten in Kleinbuchstaben umgewandelt werden (lowercase-Formel):

```
CONTAINS(LOWERCASE(repeated.current.Title),LOWERCASE(appVars  
.appVarRecord.Title))
```

Abbildung 4.1 zeigt eine beispielhafte Suche nach dem Begriff „fix “.

In SAPUI5 ist die Integration eines Suchfilters durch das Hinzufügen von weiteren Controls und einem Filter auf dem AggregationBinding der Liste möglich. Das SearchField-Control verfügt bereits über das notwendige Verhalten und die zu konfigurierenden Eigenschaften und Events. Das Search-Event wird ausgelöst, wenn der Benutzer Werte in das Suchfeld eingegeben hat und dies per Button-Druck oder Drücken der Enter-Taste bestätigt. (Siehe Quelltext 4.1)

To be done Quelltext 4.1 Implementtation von Suchfilter in View

Das Search-Event wird im Controller in der Methode „mySearchProduct

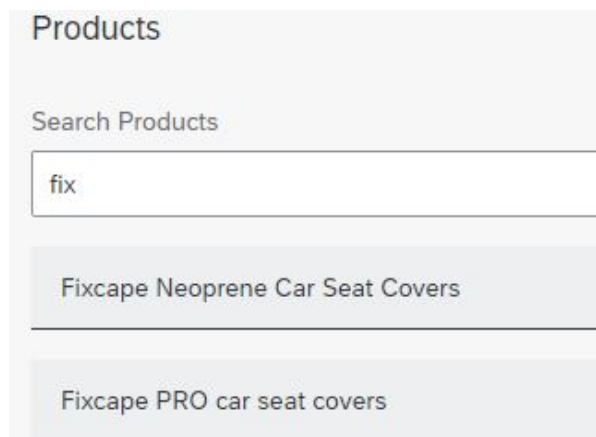


Abbildung 4.1: Suchfilter in AppGyver

“ implementiert. Über das UI5Event-Object kann dort der Suchwert als Parameter abgerufen werden. Mittels manuellem Zugriff auf die Tabelle kann das ODataListBinding angesprochen und durch die Verwendung von Filtern auf den Suchwert gefiltert werden. Dem Coding ist zu entnehmen, dass wir nur auf die Title-Eigenschaft filtern. Direkt beim Setzen des Filters wird das ODataModel das Binding aktualisieren, die gefilterten Daten vom OData-Backend anfragen und die Liste an Produkten in der UI aktualisieren.

To be done Quelltext 4.2 Implementation von Suchfilter in Controller

4.2 Integration von Paginierung

Die Paginierung in Fiori Elements ist bereits als Standard-Funktionalität vorhanden. Per Default werden in einer Fiori-Elements-Anwendung 30 Listeneinträge auf einer Seite angezeigt. Sind weitere Einträge vorhanden, werden erst beim Scrollen automatisch weitere 30 Listeneinträge geladen, bis alle Datensätzen abgerufen werden. Die Standardeinstellungen können in Low-Code Kontext nicht geändert werden.

SAP AppGyver bietet für Listenansichten keine native Unterstützung einer Paginierung. Das Verhalten kann jedoch nachgebaut werden. Hierfür werden eine List-Component und zwei Buttons zur Steuerung der Paginierung benötigt. Zusätzlich sind zwei App-Variablen notwendig, eine zum Speichern der Seitennummer (pageNumber) und die andere zum Speichern der Gesamtzahl der Einträge (recordCount). Die Logik wird dann in einer

Ablauffunktion (siehe Abbildung 4.2) implementiert. In Ablauffunktion „Get record collection “ werden zunächst die Daten von Backend geholt. Danach wird das Paging mit einem Custom Object definiert, in welchem die Page size (wie viele Einträge auf einer Seite angezeigt werden) und Page number (aktuelle Seite) abgelegt werden. (siehe Abbildung 4.3).

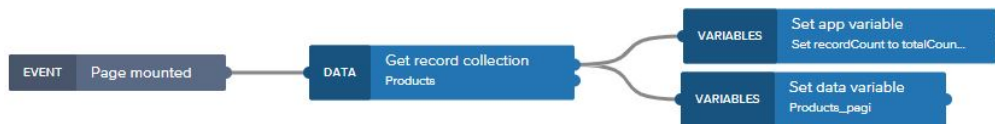


Abbildung 4.2: Seitenlogik von Paginierung in AppGyver



Abbildung 4.3: Paging definieren in Ablauffunktion „Get record collection “

Damit die Paginierung funktioniert, müssen muss die Logik der Buttons noch hinzugefügt werden. Für beide Buttons gibt es eine eigene Ablauffunktion. Abbildung 4.4 zeigt die Logik der Buttons. Für PREV-Button wird den Wert der App-Variable „pageNumber “ mit folgender Formel ermittelt:

$$SUBTRACT(appVars.pageNumber, 1)$$

Für NEXT-Button wird anstatt der SUBSTACT-Function die ADD-Function verwendet:

$$ADD(appVars.pageNumber, 1)$$

Damit der PREV-Button auf Seite 1 und der NEXT-Button auf der Letzen Seite ausgeblendet werden, muss die Sichtbarkeit der Buttons über das Visible-Flag gesteuert werden. Hier wird jeweils auf eine Formel verwiesen. Die Formel für PREV-Button lautet:

$$NOT(IS_EQUAL(appVars.pageNumber, 1))$$

Die Formel für NEXT-Button sieht wie folgt aus:

$$MULTIPLY(appVars.pageNumber, 4) \leq appVars.recordCount$$



Abbildung 4.4: Logik für PREV- und NEXT-Button in AppGyver

Abbildung 4.5 zeigt die Benutzeroberfläche in der AppGyver-Anwendung, nachdem die Paginierung implementiert wurde.

Paginierung		
Products		Back
Flying Banner PVC Car Seat Covers	34.99 Euro	>
Kinderkraft Kindersitz	109.99 Euro	>
TITLE1328	PRICE1757	>
Flying Banner Universal Car Seat Covers	34.99 Euro	>
TITLE1395	PRICE1873	>
PREV		NEXT

Abbildung 4.5: Benutzeroberfläche nach Paginierung in AppGyver

Die Integration der Paginierung in SAPUI5 ist sehr einfach, denn die entsprechenden List-Controls haben das Verhalten bereits implementiert. Setzt man die Eigenschaft „growing“ auf „true“, dann wird das Nachladeverhalten bei Scrollen aktiviert. Eine weitere Property „growingThreshold“ kann hinzugefügt werden, um die Anzahl der Listeneinträge auf einer Seite zu bestimmen. Quelltext 4.3 zeigt nur mit 2 Zeile Code kann der Paginierung in SAPUI5 implementiert werden.

to be done Quelltext 4.3 Implementation von Paginierung in SAPUI5

4.3 Integration von Bild- und PDF-Datei

Die Integration von Bild und PDF-Dateien in Fiori Elements ist im Low-Code Kontext nicht ohne weiteres möglich, da es in Fiori Elements derzeit noch keine Annotationen gibt, Datei-Formate auszuweisen. Grundsätzlich unterstützt das verwendete SAP Cloud Application Programming Model (CAP) die Bereitstellung von Mediendaten[CAP22], aber diese können nicht automatisiert in der UI dargestellt werden.

Auf der Seite von SAP CAP-Dokumentation kann man das Datenmodell wie folgt annotieren, um darauf hinzuweisen, dass die Entität Mediendaten enthält:

- *@Core.Media Type* gibt an, dass das Element Mediendaten enthält.

- *@Core.IsMediaType* gibt an, dass das Element einen MIME-Typ enthält. MIME ist die Abkürzung für Multipurpose Internet Mail Extensions und bedeutet Internet Media Type[Pag22].
- *@Core.IsURL@Core.MediaType* gibt an, dass das Element eine URL enthält, die auf die Mediendaten verweist.
- *@Core.ContentDisposition.Filename* gibt an, dass das Element als Anhang angezeigt werden soll, der heruntergeladen und lokal gespeichert wird.
- *@Core.ContentDisposition.Type* kann verwendet werden, um den Browser anzuweisen, das Element inline anzuzeigen.

Um die Medienressourcen zu lesen, sollte die GET-Anfrage in der Form */Entity/mediaProperty* verwendet werden. Um eine Medienressource zu erstellen, müssen wir zunächst eine Entität ohne Mediendaten erstellen, indem wir eine POST-Anfrage an die Entität stellen. Nach der Erstellung der Entität kann die Medien-Property mit der PUT-Methode eingefügt werden[CAP22].

Die Integration von Bild-Dateien in AppGyver ist relativ einfach zu implementieren. Hierzu gibt es in AppGyver ein Standard-View-Component Image. Mit Drag and Drop kann diese Komponente zur Benutzeroberfläche hinzugefügt werden.

Es gibt verschiedene Möglichkeiten, Bilder an die Benutzeroberfläche zu binden. (Abbildung 4.6) Mit „static image“ lässt sich ein lokales Bild auswählen und direkt hochladen. Wenn die Datei unter einer externen URL erreichbar ist, kann diese per "Formel" eingegeben werden. Weiterhin ist es möglich, auch Bilder aus Datenstrukturen auszulesen. Zudem gibt es im Marketplace verschiedene Ablauffunktionen, um die Kamera eines (mobilen) Endgeräts zu öffnen und ein Foto zu machen („Take photo“) oder ein Bild aus der Bibliothek des Geräts auszuwählen („Pick image from library“).

Die Integration von PDF-Dateien in AppGyver erfordert ebenfalls nur einen geringen Implementierungsaufwand. Im Marketplace existiert eine Ablaufkomponente „Preview PDF“, welche an einen Button angehängt werden kann. In den Eigenschaften der Ablaufkomponente kann dann dynamisch oder statisch die URL der PDF-Datei angegeben werden. Abbildung 4.7 zeigt die Logik einer Flow mit der integrierten Ablaufkomponente.

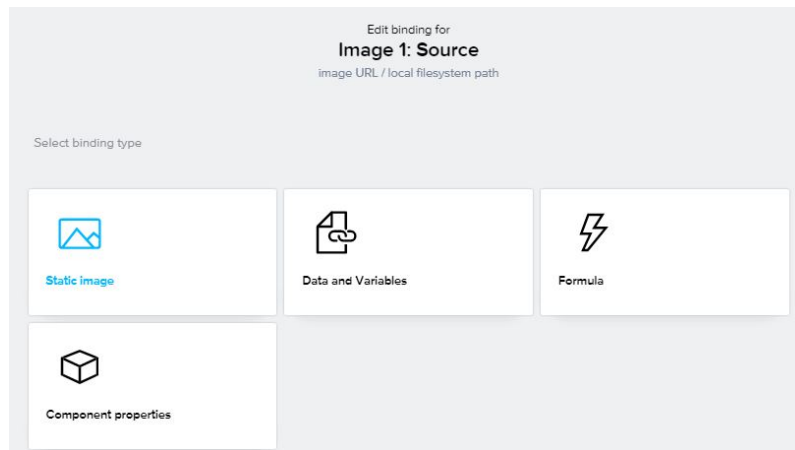


Abbildung 4.6: Image Binding in AppGyver



Abbildung 4.7: Logik für Load-PDF-Button in AppGyver

Für die Integration von Bild- und PDF-Dateien in UI5 stehen Standard-Controls zur Verfügung, die mit geringem Konfigurationsaufwand einsetzbar sind. Für Bilder kann in SAPUI5 die Control „Image“ der sap.m-Library verwendet werden. Via „src“-Property kann ein relativer oder absoluter Pfad zur URL hinterlegt werden. Zudem kann man auch die Größe („width“-Eigenschaft) oder Höhe („height“-Eigenschaft) bestimmen. Mit der Standard Control „PDFViewer“ kann man PDF-Dateien integrieren. Hier gibt es eine „source“-Eigenschaft, die den Pfad zu der anzuzeigenden PDF-Datei enthält. Quelltext 4.4 zeigt die Implementierung von Bild- und PDF-Datei Integration in UI5 auf Basis einer XML-View.

To be done Quelltext 4.4 Implementierung Bild- und PDF Datei Integration in SAPUI5

4.4 Integration von Barcode-Scanner-Funktion

Die native Integration einer Barcode-Scanner-Funktion in Fiori-Elements ist nicht vorgesehen. Über eine Extension könnte ein Button in der UI platziert werden, die eine solche Funktion bereitstellt. Die Logik dafür entspricht der SAPUI5-Implementierung (s. unten).

In AppGyver existiert eine Ablauffunktionskomponente „Scan QR/Barcode

“, welche die direkte Integration einer Barcode-Scan-Funktion ermöglicht. Das Triggern der Scan-Funktion kann über einen Button angestoßen werden. Abbildung 4.6 zeigt die Logik für einen Scan-Button. Die Scan QR/barcode Ablauf funktionskomponente kann dann mit dem Event „Component tap“ verbunden werden. Die Ablauffunktion hat mehrere Ausgänge: Es ist möglich, den Textinhalt des gescannten QR-Barcodes zu erhalten. Weiterhin kann man ebenfalls eventuell auftretende Fehler erhalten. Nachfolgende Abbildung zeigt die exemplarische Integration und Ausgabe von gefundenen Barcodes oder Fehlern in einem Dialog.



Abbildung 4.8: Logik für Scan QR/Barcode-Button in AppGyver

Die AppGyver-Anwendung kann sowohl als Webanwendung, als auch als mobile Anwendung eingesetzt werden. Die Barcode-Scanner-Funktion wird von der AppGyver allerdings in einer Desktop-Webanwendung nicht unterstützt, sondern funktioniert nur auf mobilen Endgeräten.

In SAPUI5 existieren mehrere Barcode-Scanner-Controls. Der Barcode Scanner Button (Namespace: sap.ndc) ist nur für die Nutzung auf mobilen Endgeräten vorgesehen (*ndc* steht für *native device capabilities*) und kann nur in Applikationen genutzt werden, die in der SAP Fiori Client-App geöffnet werden. Die neuere Komponente Barcode Scanner Dialog (Namespace: sap.ui.webc.fiori) kann dagegen auf allen Endgeräten im Browser eingesetzt werden.

Die Integration in eine Anwendung erfolgt durch Integration der jeweiligen Control in die View. In folgendem Beispiel wird ein BarcodeScannerButton integriert. Die Control stellt dann entsprechende Events bereit (*scanSuccess* und *scanFail*), die nach erfolgreicher oder fehlerhaftem Scan-Vorgang geworden werden.

to be Done: Quelltext 4.5 Implementierung Barcode-Scanner in XML View

Im Controller kann die Callback-Funktion dann den Wert des Barcodes oder den Fehler aus den Event-Parametern auslesen. In folgendem Beispiel werden die Informationen auf dem Bildschirm in Form eines MessageToasts ausgegeben.

to be Done: Quelltext 4.6 Implementierung Barcode-Scanner in Controller

4.5 Nutzung nativer mobiler Funktionen

Der Zugriff auf native mobile Funktionen, wie z.B. Sensor-Informationen des Smartphones, wird heute von modernen Browsern unterstützt und ermöglicht die Nutzung in Applikationen.

SAP Fiori Elements-Apps greifen standardmäßig nicht auf native Funktionen zurück. Hier können über Extensions Funktionalitäten hinzugefügt werden, was jedoch einen Programmieraufwand mit sich bringt (s. SAPUI5 unten).

SAP AppGyver unterstützt den Zugriff auf Sensordaten durch die Bereitstellung von Flow-Funktionen, sowie spezifischen Sensor-Variablen. Diese können bei der Erstellung einer Anwendung berücksichtigt werden. Folgende Sensoren werden unterstützt:

- Accelerometer
- Barometer
- Kompass
- Geolocation (GPS)
- Gyroscope
- Magnetometer

SAPUI5 bietet im Namespace *sap.ui.Device* grundlegende Informationen wie den Namen des Browsers oder der Plattform des Endgeräts, auf dem die Anwendung läuft. Tiefergreifende Funktionen zum Zugriff auf Sensordaten sind jedoch nicht verfügbar. Hierfür lässt sich im Coding jedoch auf die APIs der Browser zugreifen und die entsprechenden Informationen in den UI5-Anwendungen nutzen.

4.6 Möglichkeiten zum Deployment für unterschiedliche Endgeräte

Standardmäßig können Fiori-Elements-Anwendungen und SAPUI5-Anwendungen nur als Webanwendung deployed werden. Es gibt dennoch unterschiedliche Möglichkeiten, eine Anwendung als App auf ein mobiles Endgerät zu bekommen. Die SAP bietet mit dem SAP BTP SDK für iOS eine

native Implementierung von SAP Fiori an, die für diese Zwecke genutzt werden sollte. Quellcode oder Applikationslogik muss hier allerdings nativ in Apple Xcode und den iOS-Technologien implementiert werden. Daneben steht mit der SAP Fiori Client-App eine spezielle App zur Verfügung, mit der SAPUI5-Anwendungen auf einem mobilen Endgerät geöffnet werden können. Die Applikation wird allerdings weiter als Webapplikation deployed.

In AppGyver gibt es mehrere Möglichkeiten, die erstellte Anwendung zu deployen. Hierfür steht der SAP AppGyver Build Service zur Verfügung, der die Anwendungen Verpacken und auch direkt Verteilen kann. Die Deployment-Optionen sind folgende:

- **Android Builds**

Eine Anwendung kann via Android SDK in eine native Android-App verpackt werden. Nach dem Bauen der Anwendung, kann diese mit einem USB-Kabel auf dem Gerät installiert werden oder als Direkt-Download-Link weitergegeben werden. Es ist ebenfalls möglich, die App über den Google Play Store oder andere Android-App-Management-Lösungen zu veröffentlichen. Die Dateigröße von Android kann sehr groß sein, da die Binärdatei optimierte Unterstützung für mehrere verschiedene Prozessoren enthält[App22a].

- **iOS Builds**

Das Bauen als native iOS-App erfordert zunächst die Mitgliedschaft im Apple Developer Programm. Über das Apple Developer Portal können dann die benötigten Anlagen generiert, konfiguriert und die App gebaut werden. Nach der Generierung lässt sich die App an die Mitglieder der Apple Developer-Organisation verteilen. Ebenso ist es möglich, die App zur Review an Apple senden. Nachdem die App angenommen wurde, ist sie im öffentlichen App Store verfügbar[App22e].

- **Web Builds**

Bei der Veröffentlichung als Webanwendung mit dem Build Service, wird die Anwendung in der AppGyver Cloud unter der von uns definierten und konfigurierten Subdomain bereitgestellt.

4.7 Freie Gestaltungsmöglichkeit

Fiori-Elements basieren auf den SAP Fiori Design Guidelines und den fest definierten Floorplans(siehe Kapitel 2.6.1). Zwar sind durch die Extension-

Points und dem Hinzufügen eigener Controls geringe Abweichungen im Layout möglich, die Gestaltungsmöglichkeiten bleiben jedoch sehr beschränkt. Durch die Auswahl eines eigenen SAPUI5-Themes oder dem Überschreiben von CSS-Anweisungen könnte das Aussehen weiterhin beeinflusst werden. Die Grundidee hinter Fiori Elements, leicht zu erstellende sowie gleich aussehende und funktionierende Standard-Apps, steht jedoch generell im Widerspruch zur Idee der freien Gestaltungsmöglichkeiten.

In AppGyver ist es möglich, die Anwendungen nach dem Designvorlieben eines Nutzers zu entwickeln. Der Themen-Editor in Composer Pro ermöglicht die Auswahl, sowie die Anpassung von Themen für die Anwendungen. Ein Theme bezeichnet dabei das grundlegende visuelle Erscheinungsbild einer Anwendung und definiert Schriftgröße und -art, Farben der UI-Elemente sowie Hintergrundbilder[Por22a]. Themes besitzen zudem Inhaltspaletten und semantischen Farben, mit denen eine Anwendung gezielt Formattierungen bereitstellen kann[App22f]. Als grundlegende Themes stehen in AppGyver zwei Themes bereit:

- Das universelle Theme ist ein modernes Allround-Theme, das sich einfach als Basis für ein individuelle Theme adaptieren lässt.
- Das SAP-Fiori-Theme hilft bei der Implementierung des SAP Fiori Designsystems in AppGyver-Anwendungen[Sar21].

Im AppGyver Composer Pro haben alle View-Components Style-Klassen und Layout-Parameter. Die Style-Klassen definieren, wie eine Komponente in der UI aussieht, einschließlich Schriftart, Farben, Rahmen usw. Damit lassen sich allgemeine Werte aus den Themes überschreiben. Der Layout-Parameter bestimmt, wie eine bestimmte Komponenten auf der aktuellen Seite angeordnet wird (Abstand, Breite und Höhe und Position). Die Style und Layout können je nach Präferenz des Benutzers angepasst werden. Abbildung 4.9 zeigt 3 verschiedene Gestaltungen von AppGyver Anwendungen.

to be done: footnote

SAPUI5 orientiert sich zunächst ebenfalls an den SAP Fiori Guidelines. Anders als in Fiori Elements ist man jedoch beim Aufbau einer Anwendung in „freestyle“-Apps komplett frei. Auch das Design lässt sich anpassen, hier gibt es zwei Möglichkeiten:

- **Gestaltung einzelner UI-Elemente mit CSS**

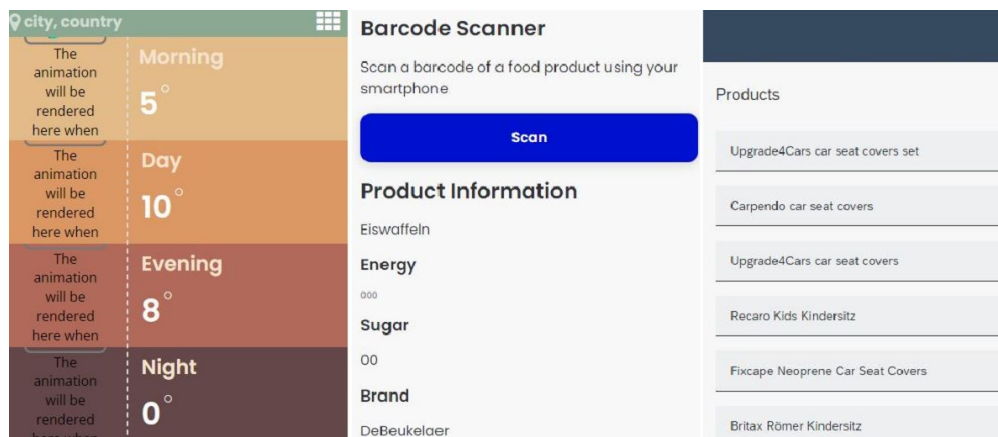


Abbildung 4.9: Gestaltung von Weather-App, Barcode-Scanner-App und Product-List-App

Einzelne Controls in SAPUI5 können durch die Vergabe von speziellen CSS-Klassen mit eigenen Style-Definition umgesetzt werden[Ant16, S.261]. Zudem ist es möglich, gezielt existierende Styles zu überschreiben.

- **Erstellung eigenständiger Themes mit Theme Designer**

SAPUI5 unterstützt ebenfalls Themes. Ein benutzerdefiniertes Theme kann mit dem UI-Theme-Designer erstellt werden. In einem Theme werden verschiedene Parameter (Schriften, Farben, etc.), Bilder und andere Ressourcen konfiguriert, die dann einen Einfluss auf alle Controls einer Anwendung haben[Por22b].

Kapitel 5

Evaluation

Das Ziel dieser Arbeit ist es, verschiedene Prototypen für die Transformationsengine zu implementieren, um daraufhin denjenigen Ansatz zu identifizieren, der am Besten für die Einbindung in die UDI Plattform geeignet ist.

Im folgenden Kapitel werden daher die vorgestellten Lösungsansätze analysiert und miteinander verglichen. Die Grundlinie stellt dabei die spezifische Version aus Kapitel ?? dar – ihr gegenüber stehen die generischen Ansätze aus Kapitel ?? und ?. Unabhängig davon wird zusätzlich die Wahl der JSON-Abfragesprache evaluiert und abschließend auf die Eignung der Transformationsengine für den Massen-Upload eingegangen.

5.1 Gegenüberstellung der drei Ansätze

Die Evaluation untersucht den Nutzen bzw. die Eignung der Transformationsengine-Varianten, indem die verschiedenen Ansätze anhand Kriterien bzgl. Funktionalität und Benutzerfreundlichkeit empirisch bewertet werden, mit dem Zweck sie zu vergleichen, um die beste Lösung für die UDI Plattform zu finden. Näheres zu Evaluationen beschreiben [DGE16] und [Heg03].

Die drei Varianten der Transformationsengine werden mit dem Kapitel, in dem ihr Konzept vorgestellt wurde, nummeriert. Es wird also der spezifische Ansatz ?? sowie die generischen Versionen, Ansatz ?? mit der Abbildung in der Excel-Datei und Ansatz ?? mit der zusätzlichen Mapping-Datei, evaluiert.


Die Kriterien basieren auf den funktionalen und nicht-funktionalen An-

forderung an die Transformationsengine, welche in Tabelle ?? spezifiziert wurden, wobei bzgl. der Benutzerfreundlichkeit ?? noch differenzierter unterteilt wird. In einem Soll/Ist-Vergleich wird die Erfüllung der Ziele überprüft.

Zur Bewertung wird ein einfaches Ampelsystem mit den Stufen

Gut  Kriterium in vollem Maße / gut erfüllt

Mittel  Kriterium teilweise / rudimentär erfüllt

Schlecht  Kriterium nicht / schlecht erfüllt

verwendet, wenngleich einige Kriterien nur  oder  zulassen.

Insgesamt ergibt sich die folgende Evaluationsmatrix:

























Nr.	Kriterium	Ansatz		
		??	??	??
??	Datentransformation nach Excel			
??	Mapping von Wahrheitswerten			
??	Mapping von Datumsformaten			
??	Mapping von Element-Wertelisten			
??	Aufwand bei Änderung/Erweiterung			
??.1	Übersichtlichkeit für Benutzer			
??.2	Fehleranfälligkeit für Benutzer			
??	Massen-Upload möglich			

Tabelle 5.1: Evaluationsmatrix für die Transformationsengine

Da Ansatz 3.2 lediglich skizziert anstatt in der Praxis implementiert wurde, werden die Kriterien bzgl. ihrer Machbarkeit aus theoretischer Sicht bewertet. Außerdem ist anzumerken, dass das Mapping für Datumsangaben in Ansatz ?? nur aus Zeitgründen eingespart wurde, aber durchaus realisiert werden kann. Im Folgenden wird die obige Bewertung noch genauer erläutert. Auf den Massen-Upload wird gesondert in Abschnitt 5.3 eingegangen – die verschiedenen Ansätze unterscheiden sich diesbezüglich nicht, da er vor allem von der Verwendung der Excel-Bibliothek abhängig ist.

Der spezifische **Ansatz ??**, welcher auf Hartkodierung setzt, hat den Vorteil, dass die Implementierung absolut unkompliziert ist. Die einzige Herausforderung ist der richtige Umgang mit der Excel-Bibliothek **Apache POI**.

Hier wirkt sich allerdings jede kleine Änderung direkt aus. Nicht nur bei neuen Behörden muss das Package erweitert werden, sondern es ergeben sich auch Codeänderungen, sobald eine Behörde ihre Anforderungen und damit ihre Vorlage minimal abändert. Ebenso wenn intern bzw. beim Hersteller Datenelemente überarbeitet werden. Aufgrund der Hartkodierung müssen Änderungen oder Erweiterungen jedes Mal manuell eingepflegt und ausgerollt werden. Die stets neuen Versionen und ihr Deployment führen zu erheblichem Aufwand, zumindest wenn auf regulatorischer Seite häufiger Anpassungen auftreten bzw. die Plattform kontinuierlich durch neue Märkte erweitert wird.

Daher werden stattdessen **generische Ansätze** zur Lösung angestrebt, die unabhängig von Behörden, Herstellern und Produktinformationen sind. Hier muss bei einer externen oder internen Änderung durch die Behörde oder p36 lediglich die neue (in Ansatz ?? überarbeitete) Excel-Vorlage im System hinterlegt und in Ansatz ?? außerdem die gespeicherte YAML-Datei aktualisiert werden. Dabei ist entscheidend, dass für diese Änderungen keine speziellen, tiefergehenden Programmierkenntnisse nötig sind, sondern dass auch jemand mit nur oberflächlichem Wissen die Umsetzung ausführen, d. h. die Transformationsengine bedienen, kann. Ansatz ?? bietet hingegen keine Benutzung in diesem Sinn.

Der Vorteil von **Ansatz ??**, bei dem man die Abbildung in den jeweiligen Excel-Arbeitsblättern verankert, besteht darin, dass nur eine einzige zusätzliche Datei nötig ist und der Nutzer visuell genau vor Augen hat, welche Daten in welche Spalten geschrieben werden. Beim Erstellen des Mappings müssen also nicht zwei Dateien miteinander abgeglichen werden, wodurch Tippfehler minimiert werden. Ein Nachteil ist allerdings, dass die Bearbeitung von Excel-Dateien unbequem ist und insbesondere längere Ausdrücke bei der JSON-Abfrage in den Excel-Zellen nicht mehr leserlich dargestellt werden können.

Ansatz ?? geht einen kleinen Umweg über eine zusätzliche Datei zur Beschreibung des Mappings. Im YAML-Format ist sie besonders schlicht, schlank und übersichtlich. Während man die JSONata-Ausdrücke auf den ersten Blick in voller Gänze lesen kann und auch mehrzeilige Funktionen unterstützt werden, muss natürlich die Abbildung als solche zunächst definiert werden. Das heißt, die Namen aller Tabellenblätter, die jeweiligen Startreihen, sowie die Spalten und ihr gewünschter Inhalt müssen abge-

tippt bzw. angegeben werden. Das bringt einen gewissen Aufwand und Fehleranfälligkeit mit sich. Zusätzliche Flexibilität und Struktur erhält man durch die unkomplizierte Definition von Mappings für Wahrheitswerte, Datumsformate oder einzelne Datenelemente. Die YAML-Datei kann hierbei auf unkomplizierte Weise beliebig erweitert werden, falls neue Funktionen implementiert werden – zum Beispiel die Angabe einer Maximalanzahl an Zeilen. Ist der Maximalwert erreicht, werden alle weiteren Produkte in eine neue Arbeitsmappe geschrieben, um Overflow zu vermeiden.

Insgesamt kristallisiert sich der letzte Ansatz als der Beste heraus. Die gewonnene Übersicht durch die klare und intuitive Struktur der YAML-Datei rechtfertigt die zusätzliche Datei und mögliche Übertragungsfehler. Für die UDI Plattform wird daher Ansatz ?? empfohlen.

5.2 JSON-Abfragesprache

Neben der bereits angesprochenen Unterscheidung, wo und wie die Mapping Informationen gespeichert werden, also konkret in welcher Datei, ist auch die Wahl der Abfragesprache von Bedeutung. Sie legt zu einem gewissen Teil Form und Umfang der Abbildung zwischen der Excel-Vorlage und den Produktdaten fest. Einen detaillierten Überblick über eine Auswahl verschiedener JSON-Abfragesprachen und ihrer Eigenschaften bzw. Vor- und Nachteile vermittelt das Grundlagenkapitel ??.

Als Einstieg in die Thematik wurde anfänglich `JSONPath` verwendet, wobei sich schnell die ersten Grenzen aufgezeigt haben. Daher wurde in Ansatz ?? zunächst auf `JMESPath` gesetzt und später in Ansatz ?? auf `JSONata` umgestellt. Die JSON-Abfragesprachen sind unabhängig von den vorgestellten Ansätzen und entsprechend beliebig kombinierbar: Variante ?? mit `JSONata`- anstatt `JMESPath`-Ausdrücken im Excel-Arbeitsblatt wäre z. B. genauso möglich.

Die Kriterien für die Evaluation, um die für p36 am besten geeignete Abfragesprache zu ermitteln, betreffen zum einen die Sprache selbst und zum anderen deren Anwendung innerhalb der Transformationsengine. Es wird die Mächtigkeit der Sprache begutachtet – im konkreten Fall heißt das, ob mit ihr die Werte der Datenelemente in die typischerweise von den Behörden gewünschte Form transformiert werden können und darüber hinaus ggf. noch komplexere Abfragen möglich wären. Daneben wird die Erweiterbarkeit durch selbst definierte Funktionen beurteilt, und wie übersichtlich und intuitiv die Syntax ist, sowie das Verhalten im Anwendungsfall

bei komplexen Datenelementen. Die vorhandenen Java-Implementierungen werden bzgl. Umfang, Verbreitung und Aktualität beurteilt. Außerdem wird der Aufwand gemessen, den die Benutzer zur Einarbeitung in bzw. im Umgang mit der Sprache haben.

Unter Verwendung des Ampelsystems aus Abschnitt 5.1 ergibt sich die folgende Evaluationsmatrix, deren Einträge im Nachgang noch genauer erläutert werden.



















Kriterium	JSONPath	JMESPath	JSONata
Mächtigkeit			
Erweiterbarkeit			
Übersichtlichkeit der Syntax			
Abfrage komplexer Elemente			
Java-Bibliothek			
Einarbeitungsaufwand			

Tabelle 5.2: Evaluationsmatrix für die JSON-Abfragesprache

JSONPath ist weit verbreitet, sehr populär und dient sicher als erste Anlaufstelle. Im Vergleich ist es allerdings auch die simpelste und am wenigsten mächtigste Abfragesprache von den hier verwendeten. Die Implementierung von **JSONPath** in Java durch Jayway bietet über die reine Spezifikation hinaus Zusatzfunktionalitäten, wie z. B. einige eingebaute Funktionen und diverse Filtermöglichkeiten, siehe [jay22]. Komplexere Funktionen, wie zum Beispiel den ternären Operator oder **contains** zur Abfrage von Array-Inhalten, werden allerdings nicht unterstützt und können auch nicht innerhalb der Abfrage selbst definiert werden, sondern nur in der Implementierung der Engine. Es gibt hier den großen Vorteil, dass mit der Konfigurationsoption **DEFAULT_PATH_LEAF_TO_NULL** automatisch Nullwerte für fehlende Pfade bzw. Blätter zurückgegeben werden, was bei den relationalen Arbeitsblättern wichtig ist. Wie bereits in Kapitel ?? und ?? angesprochen, muss hierfür sowohl bei **JMESPath** als auch **JSONata** auf einen Workaround zur absichtlichen Erzeugung von Nullwerten gesetzt werden, der die Ausdrücke etwas aufbläht.

Die Abfragesprache **JMESPath** besticht besonders durch den im Vergleich zu **JSONPath** deutlich angestiegen Umfang und ihre ausführliche Dokumentation, auch wenn die Sprache in ihrer Komplexität zugenommen hat. Mit der Hintereinanderausführung per Pipe-Symbol sind vielfältige Array-manipulationen möglich, die zum Teil Arrays aufblähen und durch [] ggf.

wieder abflachen, worunter die Übersichtlichkeit etwas leidet. Außerdem können eigene Funktionen definiert werden, wenn auch nicht in der Abfrage selbst. Insgesamt konnten mit **JMESPath** aber alle bisher aufgetretenen Abfragen in der Transformationsengine realisiert werden. **JMESPath** ist bestrebt in vielen Programmiersprachen als Schnittstelle zur Verfügung zu stehen, so auch in Java. Sie erfreut sich wachsender Popularität, wobei sie allerdings hinter **JSONPath** zurückfällt.

JSONata ist noch mächtiger, allerdings ohne dass die Syntax darunter leidet. Die Sprache hat den großen Vorteil, dass die Nutzer, das heißt die Mitarbeiter des Solution Management, schon Erfahrung mit ihr gesammelt haben, da **JSONata** bereits an anderer Stelle verwendet wird. Es ist also keine größere Einarbeitung oder Umstellung notwendig. Obwohl **JSONata** ursprünglich nur für TypeScript und NodeJS entwickelt wurde, schafft die API **JSONata4Java** Abhilfe, vgl. [IBM22]. Die Bibliothek ist unscheinbar und hat vergleichsweise wenig Nutzer, wird jedoch kontinuierlich gepflegt. Vereinzelte Funktionen wurden bisher nicht implementiert, allerdings kommen sie in der Engine nicht zum Tragen, sodass es im vorliegenden Anwendungsfall kein wirklicher Nachteil ist. Dem gegenüber besteht hier ebenso die Möglichkeit, die Sprache durch eigene Funktionsdefinitionen zu erweitern, und zwar auch innerhalb des **JSONata**-Ausdrucks.

Insgesamt fällt die Wahl der Abfragesprache daher eindeutig auf **JSONata**, deren Umfang und kompakte Schreibweise überzeugen, während sie die Benutzer nicht mit der Wissensaneignung einer zusätzlichen Sprache belastet. Anzumerken ist, dass die Performanz der Sprachen bei der Betrachtung außer Acht gelassen wurde. Diese ist nicht ausschlaggebend für die Transformationsengine, da sie sich in jedem Fall im annehmbaren Bereich bewegt (wie im anschließenden Abschnitt erläutert wird), allerdings schneidet dies bezüglich **JSONata** vermutlich weniger gut ab, vgl. [Zus20] und [mt22]¹.

5.3 Massen-Upload

Ein wichtiges Kriterium ist die Skalierbarkeit über der Anzahl der Produkte, die in die Excel-Vorlage geschrieben werden sollen. Die Hersteller registrieren gerade beim Einstieg in einem neuen Markt bzw. als Neukun-

¹ In den Benchmarks werden die JavaScript-Bibliotheken verglichen, nicht Java. Daher kann für den vorliegenden Fall keine konkrete Aussage bzgl. der Performanz getroffen werden.

de von p36 nicht nur einzelne Produkte, sondern ihr gesamtes Portfolio. Hierzu muss die Transformationsengine in der Lage sein, Daten in der Größenordnung von 1.000 bis 10.000 Produkten zu übermitteln. Für noch größere Uploads besteht keine Anforderung, da man den Prozess ggf. auch wiederholt ausführen kann, im Folgenden werden allerdings die maximalen Grenzen ausgetestet.

Dazu wurden beim finalen Ansatz ?? Modultests mit unterschiedlich vielen Produkten ausgeführt. Die Tests haben ergeben, dass bis zu 16.000 Mal das Beispielprodukt in die Vorlage geschrieben werden kann. Bei dem Versuch die Excel-Datei mit 17.000 Produkten abzuspeichern, kommt es zum Heap-Overflow und der `OutOfMemoryError` wird geworfen. Man muss hierbei bedenken, dass in den relationalen Arbeitsblättern für jedes Produkt nicht nur eine Zeile erzeugt wird, sondern mehrere. In den Tests wurde mit dem Faktor 10 gerechnet, das heißt ein Produkt erzeugt zehn Zeilen in der Excel-Datei.

Zur Überprüfung, ob eine Arbeitsmappe erfolgreich ausgefüllt wurde, wird diese nach dem Schreiben erneut geladen, um so die Anzahl der Zeilen auszulesen. Hierbei ergeben sich schon früher Probleme. Ab 10.000 Produkten, das heißt also 100.000 Zeilen, konnte die Excel-Datei nicht mehr über die Bibliothek **Apache POI** eingelesen werden, sondern es kam zum Überlauf des Speichers und der `OutOfMemoryError`-Ausnahme. Daher wurde ab dieser Größe ein Test schon als erfolgreich gezählt, wenn die Datei nur erzeugt wurde und existiert, ohne sie erneut zu öffnen und genauer zu untersuchen. Manuell ist das Öffnen natürlich ohne Probleme möglich und die Daten wurden auch korrekt gefüllt.

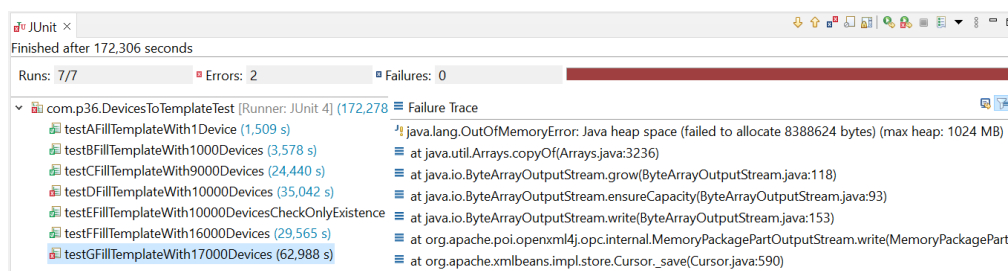


Abbildung 5.1: Testergebnisse zum Massen-Upload

Die Ergebnisse eines Testdurchlaufs sind in Abbildung 5.1 festgehalten. Hier sieht man die Fehlermeldung bei 17.000 Devices. Auch im ersten Test mit 10.000 Devices, bei dem die Datei zur Überprüfung eingelesen wird, kommt es zu einem kleinen Heap-Overflow. Die anderen Tests für bis zu 16.000

Produkten sind grün unterlegt und laufen erfolgreich durch. Der Massen-Upload ist also mit der Transformationsengine realisierbar. Die Dauer des Schreibvorgangs bis zur erfolgreichen Excel-Datei-Erzeugung ist mit nur wenigen Sekunden sehr angenehm. Auch bei vielen Produkten bleibt die Zeit im Bereich von etwa 30 bis 60 Sekunden. Hierbei ist anzumerken, dass die Zeiten im Normalfall (wenn die Testreihenfolge nicht zur besseren Übersicht fixiert ist) stärker variieren, je nachdem welcher Test zuerst durchgeführt wird und damit ohne vorheriges Aufwärmen der JVM am längsten braucht. Trotz dieser Ungenauigkeiten verdeutlichen die Tests gut die Größenordnung der Geschwindigkeit des Transformationsprozesses.

Literaturverzeichnis

- [Ant16] ANTOLOVIC, Miroslav: *Einführung in SAPUI5*, 2. Auflage. Rheinwerk Verlag, 2016. – ISBN: 978-3-8362-8901-6
- [App22a] APPGYVER: *Android builds*. 2022. – URL: <https://docs.appgyver.com/docs/android-builds> [abgerufen am 22.12.2022]
- [App22b] APPGYVER: *App logic*. 2022. – URL: <https://docs.appgyver.com/docs/app-logic> [abgerufen am 29.11.2022]
- [App22c] APPGYVER: *Binding data*. 2022. – URL: <https://docs.appgyver.com/docs/en/binding-data?highlight=data%20binding> [abgerufen am 13.01.2023]
- [App22d] APPGYVER: *Community Announcement*. 2022. – URL: <https://forums.appgyver.com/t/community-announcement/19453> [abgerufen am 22.11.2022]
- [App22e] APPGYVER: *iOS builds*. 2022. – URL: <https://docs.appgyver.com/docs/ios-builds> [abgerufen am 22.12.2022]
- [App22f] APPGYVER: *Theme*. 2022. – URL: <https://docs.appgyver.com/docs/theme?highlight=theme> [abgerufen am 26.12.2022]
- [App22g] 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]

- [CAP22] CAP, SAP: *Serving Media Data*. 2022. – URL: <https://cap.cloud.sap/docs/guides/media-data> [abgerufen am 20.12.2022]
- [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]
- [DGE16] DEGEVAL, Gesellschaft für Evaluation (Hrsg.): *Standards für Evaluation – Erste Revision*. Mainz, 2016. – ISBN: 978–3–941569–06–5
- [Doc22a] DOCUMENTATION, SAP U.: *App Overview: The Basic Files of Your App*. 2022. – URL: <https://sapui5.hana.ondemand.c>

- om/#/topic/28b59ca857044a7890a22aec8cf1fee9 [abgerufen am 24.12.2022]
- [Doc22b] DOCUMENTATION, SAP U.: *Step 16: Dialogs and Fragments*. 2022. – URL: <https://sapui5.hana.ondemand.com/sdk/#/topic/4da72985139b4b83b5f1c1e0c0d2ed5a> [abgerufen am 28.12.2022]
- [Doc22c] DOCUMENTATION, SAP U.: *Step 19: Aggregation Binding*. 2022. – URL: <https://sapui5.hana.ondemand.com/sdk/#/topic/bf71375454654b44af01379a3c3a6273> [abgerufen am 28.12.2022]
- [Doc22d] DOCUMENTATION, SAP U.: *Step 25: Remote OData Service*. 2022. – URL: <https://sapui5.hana.ondemand.com/#/topic/44062441f3bd4c67a4f665ae362d1109> [abgerufen am 27.12.2022]
- [Doc22e] DOCUMENTATION, SAPUI5: *Developing Apps with SAP Fiori Elements*. 2022. – URL: <https://sapui5.hana.ondemand.com/#/topic/03265b0408e2432c9571d6b3feb6b1fd> [abgerufen am 28.11.2022]
- [Doc22f] DOCUMENTATION, SAPUI5: *Using SAP Fiori Elements Floorplans*. 2022. – URL: <https://sapui5.hana.ondemand.com/#/topic/03265b0408e2432c9571d6b3feb6b1fd> [abgerufen am 28.11.2022]
- [doc22g] 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]
- [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]
- [Heg03] HEGNER, Marcus: *Methoden zur Evaluation von Software* (IZ-Arbeitsbericht Nr. 29). Bonn: Informationszentrum Sozialwissenschaften, 2003. – ISSN: 1431–6943
- [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]
- [IBM22] IBM, International Business Machines: *JSONata4Java*. 2022. – URL: <https://github.com/IBM/JSONata4Java> [abgerufen am 20.07.2022]
- [jay22] *Jayway JsonPath*. 2022. – URL: <https://github.com/json-path/JsonPath> [abgerufen am 10.08.2022]
- [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]

- [Mar22] MARKETPLACE, Visual S.: *SAP CDS Language Support*. 2022. – URL: <https://marketplace.visualstudio.com/items?itemName=SAPSE.vscode-cds> [abgerufen am 19.12.2022]
- [mt22] *JSON Query Languages*. 2022. – URL: <https://www.measurethat.net/Benchmarks/Show/20654> [abgerufen am 27.08.2022]
- [Mue22] MUESSIG, Peter: *Getting Started with TypeScript for UI5 Application Development*. 2022. – URL: <https://blogs.sap.com/2021/07/01/getting-started-with-typescript-for-ui5-application-development/> [abgerufen am 24.12.2022]
- [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]
- [Pag22] PAGERANGERS: *MIME-Typ*. 2022. – URL: <https://pagerangers.com/seo-handbuch/onpage/allgemein/was-ist-ein-mime-typ/> [abgerufen am 20.12.2022]
- [Por22a] PORTAL, SAP H.: *Basics of Theming*. 2022. – URL: https://help.sap.com/docs/SAP_NETWEAVER_750/ab06dedc873746eaba1c041200c068e0/91ebfe2a14204244bd052a03018d6781.html?version=7.5.6&locale=en-US [abgerufen am 26.12.2022]
- [Por22b] PORTAL, SAP H.: *Basics of Theming*. 2022. – URL: https://help.sap.com/docs/SAP_NETWEAVER_750/ab06dedc873746eaba1c041200c068e0/91ebfe2a14204244bd052a03018d6781.html?version=7.5.6&locale=en-US [abgerufen am 26.12.2022]
- [Por22c] 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]

- [Por22d] 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]
- [Sar21] SARSA, Harri: *New style and theme features + SAP Fiori Theme*. 2021. – URL: <https://blog.appgyver.com/new-style-and-theme-features-sap-fiori-theme-68667796e6fb> [abgerufen am 25.12.2022]
- [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]
- [Zus20] ZUSCHLAG, Cody: *The JSONata Performance Dilemma*. 2020. – URL: <https://www.nearform.com/blog/the-jsonata-performance-dilemma/> [abgerufen am 24.08.2022]

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 detailliert auf JSON-Schema eingegangen. Das verwendete Schema ist in der Datei `MappingJsonSchema.json` gespeichert und wie folgt definiert:

```
1 {
2   "$schema": "https://json-schema.org/draft-06/schema#",
3   "title": "jsonataExcelMapping",
4   "description": "A mapping for JSONata input into excel template",
5   "type": "object",
6   "properties": {
7     "SheetMappings": {
8       "description": "mappings for the sheets in the excel template",
9       "type": "object",
10      "additionalProperties": {
11        "description": "different sheetNames",
12        "type": "object",
13        "properties": {
14          "row": {
15            "description": "start row in excel sheet where the device
16              data shall be written",
17            "type": "integer",
18            "minimum": 1
19          },
20          "category": {
21            "description": "category of the sheet (complex data element
22              key)",
23            "type": "string"
24          },
25          "columns": {
26            "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   "required": ["row", "columns" ]
45 },
46 {
47   "ElementMappings": {
48     "description": "mappings for booleans, date formats and/or
                specific data elements",
49     "type": "object",
50     "properties": {
51       "Boolean": {
52         "description": "mapping for boolean values in this template",
53         "type": "object",
54         "properties": {
55           "true": {
56             "description": "mapping of true in this agency",
57             "type": ["number", "string", "boolean"]
58           },
59           "false": {
60             "description": "mapping of false in this agency",
61             "type": ["number", "string", "boolean"]
62           }
63         },
64         "additionalProperties": false,
65         "required": ["false", "true"]
66       },
67       "Date": {
68         "description": "mapping for date or time formats",
69         "type": "object",
70         "additionalProperties": {
71           "description": "key: date format for p36 / value: date format
                in excel for agency",

```

```
73         "type": "string"
74     }
75 },
76 "additionalProperties": {
77     "description": "data element key",
78     "type": "object",
79     "additionalProperties": {
80         "description": "key: element value for p36 / value: element
81             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