
# Darstellung und Vergleich mehrerer Möglichkeiten zur Umsetzung eines sequentiellen HR-Prozesses im RESTful API-Umfeld

## Projektarbeit 1

im Rahmen der Prüfung zum

**Bachelor of Science (B.Sc.)**

des Studienganges Wirtschaftsinformatik

an der Dualen Hochschule Baden-Württemberg Karlsruhe

von

**Tom Wolfrum**

- Sperrvermerk -

| | |
|---|---|
| Abgabedatum: | 4. September 2023 |
| Bearbeitungszeitraum: | 05.06.2023 - 03.09.2023 |
| Kurs: | WWI22B5 |
| Ausbildungsfirma: | SAP SE<br>Dietmar-Hopp-Allee 16<br>69190 Walldorf, Deutschland |
| Betreuer der Ausbildungsfirma: | Steven Rösinger |
| Gutachter der Dualen Hochschule: | Paul Peitz |

# Sperrvermerk

Die nachfolgende Arbeit enthält vertrauliche Daten der:

SAP SE
Dietmar-Hopp-Allee 16
69190 Walldorf, Deutschland

Der Inhalt dieser Arbeit darf weder als Ganzes noch in Auszügen Personen ausserhalb des Prüfungs- und Evaluationsverfahrens zugänglich gemacht werden, sofern keine anders lautende Genehmigung des Dualen Partners vorliegt.

# Selbstständigkeitserklärung

Ich versichere hiermit, dass ich die vorliegende Projektarbeit 1 mit dem Thema:

> *Darstellung und Vergleich mehrerer Möglichkeiten zur Umsetzung eines sequentiellen HR-Prozesses im RESTful API-Umfeld*

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Karlsruhe, den 12. Juli 2023

---

Wolfrum, Tom

# Inhaltsverzeichnis

# Abkürzungsverzeichnis

| | |
|---|---|
| **SaaS** | Software-as-a-Service |
| **AIS** | Application Innovation Services |
| **HCM** | Human Capital Management |
| **API** | Application Programming Interface |
| **REST** | Representational State Transfer |
| **URI** | Uniform Resource Identifier |
| **URL** | Uniform Resource Locator |
| **RAP** | Restful Application Programming Model |
| **ABAP** | Advanced Business Application Programming |
| **CDS** | Core Data Service |
| **BO** | Business Object |
| **UI** | User Interface (Benutzeroberfläche) |

# Abbildungsverzeichnis

# Tabellenverzeichnis

# 1 Einleitung

## 1.1 Unternehmensprofil und Anwendungsbezug

SAP ist ein börsennotierter Softwarekonzern mit Sitz in Walldorf. Das Unternehmen wurde 1972 von 5 IBM-Mitarbeitern, darunter Hasso Plattner und Dietmar Hopp gegründet. Das Hauptgeschäft ist die Entwicklung von Unternehmenssoftware zur Abwicklung von Geschäftsprozessen, unter anderem in den Berichen Buchführung, Controlling, Vertrieb, Einkauf, Produktion, Lagerhaltung, Transport und Personalwesen. Für das Unternehmen arbeiten heute 105.000 Mitarbeiter an Standorten in 157 Ländern und erwirtschaften einen Umsatz von ca. 29,5 Mrd. €. Erfolgreich wurde das Unternehmen mit seinem Standardsoftwarepaket SAP R/2 für Großrechnersysteme und später mit SAP R/3 für Client-Server-Systeme. Die Vorstellung der Mittelstandslösung SAP ByDesign im Jahr 2007 als Cloud-Produkt läutete die bis heute andauernde Transformation der gesamten Produktpallette in Richtung Cloud/ SaaS ein, die 2015 mit der Einführung von S/4 HANA als Hauptprodukt noch einmal verstärkt wurde.

Die Abteilung AIS HCM ist Teil des Unternehmensbereichs Product Engineering und zuständig für 2nd-Level-Support und Eigenentwicklungen für die on-premise Variante der SAP Personallösung HCM. Die Kunden der Abteilung sind Unternehmen die HCM verwenden und zusätzlich Wartungsverträge mit der SAP abgeschlossen haben. Somit ist das Hauptgeschäft tiefergehende Probleme, die durch den 1st-Level Product-Support nicht gelöst werden können, zu beheben und kleinere Features für Kunden mit speziellen Anforderungen umzusetzen. Zudem stellt die Abteilung mehrere SAP Fiori Apps als Self-Service für Mitarbeiter z. B. um Urlaub zu beantragen und Manager z. B. um Urlaubsanträge zu bearbeiten, bereit. Auch diese Fiori Apps sind vom technologischen Wandel der SAP betroffen und dadurch ergeben sich hier relevante Änderungen, die im nachfolgenden Kapitel noch genauer beschrieben werden. Durch das hohe Nutzungsvolumen dieser Apps und der somit großen betriebswirtschaftlichen Relevanz, sind diese und auch der Untersuchungsgegenstand dieser Arbeit für das Produkt HCM von großer Bedeutung.

## 1.2 Motivation und Problemstellung

Im folgenden Kapitel soll dargestellt werden, welche Probleme sich durch gewisse technische Veränderungen der SAP Produkten ergeben und sich somit für eine wissenschaftliche Untersuchung im Rahmen dieser Arbeit anbieten.

Die von der Abteilung betriebenen Fiori Apps, die schon im Zusammenhang der Einleitung angesprochen wurden, sind auf Basis des Frameworks SAP UI5 auf HTML5 Basis für ein älteres Produkt - SAP ERP - entwickelt worden. In diesem Produkt sind viele relevante Funktionen, die mit der neuen ERP-Lösung S/4 HANA eingeführt werden, nicht vorhanden. Da der Support für diese Anwendung 2027 ausläuft, ist der Umstieg von auf S/4 HANA für viele Kunden ein relevantes Thema. Im neuen S/4 HANA System ("S/4" abgekürzt) finden somit auch die S/4-Design-Guidelines Anwendung. Das sind Vorgaben die gewisse Aspekte, wie z. B. Oberflächen-Design oder Technologien, die bei der Entwicklung in diesem System verwendet werden müssen, festlegt. Die in den Richtlinien festgelegten neueren Technologien stimmen jedoch nicht mit denen überein, die zur Entwicklungszeit von HCM verwendet wurden.

Dieser Umstand war ursprünglich kein Problem, da vom Management die strategische Entscheidung getroffen wurde, dass HCM nicht in S/4 HANA integriert wird und durch die neuere Personallösung SuccessFactors ersetzt werden soll. Aufgrund fehlender Funktionalitäten in SuccessFactors und hoher verlässlicher Einnahmen durch Wartungsverträge für HCM wurde diese Entscheidung revidiert und HCM ist Bestandteil von S/4 als "HCM for S/4 HANA" bzw. "H4S4". Somit finden jetzt auch auf die Fiori Apps für HCM die Design-Guidelines von S/4 Anwendung und jegliche neue Apps, die entwickelt werden müssen die Technologie Fiori Elements verwenden. Da es einen erheblichen Entwicklungsaufwand darstellen würde, alle bestehenden Apps, die somit auch Fiori Elements verwenden müssten dahingehend umzubauen, dürfen aus Praktikabilitätsgründen bereits existierende Apps mit den notwendigen Anpassunge auf Basis der älteren Fiori Freestyle Technologie in S/4 weiterbetrieben werden.

Diese Situation sorgt für ein Problem für manche Geschäftsprozesse, die über solche Apps abgebildet werden sollen. Das Framework Fiori Elements generiert das gesamte Front-End der Anwendung selbstständig. Das erleichtert auf der einen Seite die Entwicklung der Apps, da lediglich die benötigten Daten bereitgestellt und je nach Anwendungsfall aggregiert und auf bestimmte Art und Weise dargestellt werden müssen. Auf der anderen

Seite kann dadurch jedoch keine eigene Programmlogik mehr im Front-End eingebaut werden. Zudem wird die Kommunikation mit dem Back-End über eine RESTful API, die zustandslos angelegt ist, abgewickelt. Auch wenn eine RESTful-API viele Vorteile mit sich bringt, birgt die angesprochene Zustandslosigkeit jedoch den Nachteil, dass Anwendungen, deren Prozesse asynchrone Kommunikation benötigen nur noch schwer abbildbar sind.

Es ist dennoch wichtig die Möglichkeit zu haben solche asynchronen Geschäftsprozesse abzubilden, da z. B. ein Urlaubsantrag eines Mitarbeiters nicht direkt persistent in der Datenbank gespeichert werden soll, sondern erst wenn er zeitlich asynchron vom jeweiligen Manager genehmigt wurde. Diese Situation ist ein Beispiel für die Limitation einer zustandslosen API, da die Genehmigung des Urlaubs zeitlich versetzt als eigene, vom Urlaubsantrag getrennte Anfrage an die API geschickt würde und der Prozess sich so nicht abbilden lieSSe. Dieses Beispiel ist eines von von vielen, weshalb eine Lösung für asynchrone Prozesse in diesem Umfeld dringend benötigt wird.

In der vorliegenden Arbeit soll nun untersucht werden, wie sich solche asynchronen Prozesse, trotz den eben dargelegten Einschränkungen trotzdem im neuen S/4 HANA Umfeld mit den neueren Technologien umsetzen lassen.

## 1.3 Aufbau und Ziel der Arbeit

Im Folgenden wird der Aufbau und das Ziel der Arbeit thematisiert.

Als erstes wird im einleitenden Kapitel die SAP und die Abteilung AIS HCM, in der die Praxisphase absolviert wurde, kurz vorgestellt und somit der Anwendungsbezug der Arbeit hergestellt. Danach wird mit der Motivation und Problemstellung der Untersuchungsgegenstand und die Bedeutung der Arbeit für die Abteilung und die Kunden erläutert. Danach soll die Arbeit klar von verwandten Themen abgegrenzt werden, um einen klaren Rahmen für die Untersuchung zu schaffen. Im methodischen Vorgehen werden dann abschlieSSend für die Einleitung noch auf die wissenschaftlichen Methoden, die verwendet wurden, um die Untersuchungsergebnisse zu erhalten, eingegangen. Der Hauptteil der Arbeit besteht aus zwei Teilen: Im ersten Teil werden die theoretischen Grundlagen der Arbeit gelegt. Hier werden die Designprinzipien einer RESTful API, wie diese im RESTful Application Programming Model der SAP eingesetzt werden und die Technologie Fiori Elements näher beleuchtet. Der praktische Hauptteil stellt drei Ansätze

vor, wie das in der Problemstellung thematisierte Problem gelöst werden kann. Hierfür werden die Technologien Business Workflows, Business Events und das Background Processing Framework vorgestellt und im Bezug auf Stärken und Schächen sowie Effizienz und Robustheit verglichen. Diese Ergebnisse werden dann in einer Entscheidungsmatrix dargestellt. Das Ziel soll es sein, dass diese Entscheidungsmatrix klare Tendenzen gibt, welche der untersuchten Technologien sich im konkreten Anwendungsfall für das Abbilden von asynchronen Prozessen im RESTful API Umfeld anbietet. Im Schlussteil werden die Ergebnisse der Arbeit nochmals zusammengefasst, eine konkrete Handlungsempfehlung für die Lösung dieses Problems gegeben und die Ergebnisse abschlieSSend kritisch Reflektiert und ein Ausblick auf zukünftige Entwicklungen gegeben.

## 1.4 Abgrenzung

In diesem Kapitel soll die Arbeit klar von verwandten Themen abgegrenzt werden um einen klaren Rahmen für die Untersuchung zu schaffen. Der Zweck der vorliegenden Arbeit ist es, die drei vorgestellten Technologien vergleichend zu bewerten und je nach Anwendungsfall eine Handlungsempfehlung im Bezug auf eine sich anbietende Technologie zu geben. Über diese drei Technologien hinaus werden keine anderen Möglichkeiten asynchrone Prozesse abzubilden, wie z. B. im Cloud Application Programming Model (CAP) behandelt. Nicht behandelt werden die Technologien HIER TECHNOLOGIEN EINFÜGEN Außerdem findet aufgrund des beschränkten Umfangs der Arbeit lediglich ein Vergleich der Technologien statt und keine direkte Implementierung dieser in einem konkreten Anwendungsfall. Hierfür sei auf die offizielle Dokumentation der SAP für die respektiven Technologien verwiesen.

## 1.5 Methodisches Vorgehen

Wie werden die Ansätze betrachtet? Wie kommt die Handlungsempfehlung zustande?

# 2 Theoretische Grundlagen

Im Folgenden sollen die theoretischen Grundlagen für die nachfolgende vergleichende Darstellung der Umsetzung sequentieller Prozesse im REST-Umfeld gelegt werden. Zuerst wird grundsätzlich erklärt, was eine RESTful-API ist und danach wird auf die Umsetzung von REST in ABAP näher erleutert. Abgeschlossen wird der theoretische Teil der Arbeit mit einer Darstellung von Fiori Elements, dem Framework zur Entwicklung von Fiori Apps.

## 2.1 RESTful Application Programming Interface

Eine API ist eine Schnittstelle, über die verschiedene Softwareanwendungen miteinander kommunizieren können. Die API definiert die Methoden, Protokolle und Tools, die für den Zugriff auf die Funktionen und Daten einer Softwareanwendung verwendet werden können. Somit standardisiert eine API die Kommunikation verschiedener Anwendungen und ermöglicht den Zugriff auf bereitgestellte Daten ohne dass die zugreifende Anwendung die interne Logik oder Implementierung der anderen Anwendung kennen muss.

Eine RESTful-API ist eine spezielle Schnittstelle, die den Designkonventionen nach REST folgt.

Das erste Prinzip ist die Client-Server-Architektur. Das bedeutet, dass die Benutzeroberfläche von den gespeicherten Daten getrennt wird. Die Benutzeroberfläche und Sitzung existiert nur auf dem Client und die gespeicherten Daten oder zur verfügung gestellten Funktionen existieren nur auf dem Server. Somit wird die Portierbarkeit und Skalierbarkeit des Gesamtsystems verbessert. Zudem wird die Möglichkeit einer unabhängigen Weiterentwicklung der verschiedenen Komponenten sichergestellt.

Zudem soll eine RESTful-API zustandslos angelegt sein. Das heißt im Genaueren, dass die Kommunikation der verschiedenen Parteien zustandslos sein muss. Es muss für den Server somit möglich sein, die Anfrage des Clients vollständig zu verstehen und zu verarbeiten, ohne zusätzlich auf vergangene Anfragen zugreifen zu müssen. Auf der

anderen Seite bedeutet das auch, dass der Client jede Antwort des Servers ohne zusätzliche Inforamtionen, die eventuell zu einem früheren Zeitpunkt angefordert wurden verstehen können muss. Das heißt, dass in jeder Anfrage immer alle notwendigen Informationen mitgeschickt werden müssen und von keinem "Vorwissen" ausgegangen werden darf. Das hat wiederum zur Folge, dass Sitzungsinformationen ausschließlich auf dem Client gespeichert werden. Durch diese Bedingung verbessert sich die Skalierbarkeit weiter, da der Server Ressourcen, die ansonsten für die Speicherung der Stati der Requests benötigt würden, nicht freihalten muss. Zudem steigt die Zuverlässigkeit der Schnittstelle, da bei einem Fehler immer nur eine Request betrachtet werden muss. Somit ist ein Fehler einfacher behebbar und hat keine Auswirkungen auf andere Anfragen. Damit einher geht auch ein vereinfachtes Monitoring, da immer nur eine Request betrachtet werden muss und nicht erst eine Kette zusammenhängender Anfragen nachvollzogen werden muss.

Die dritte Designkonvention besagt, dass auf der Client Seite ein Cache vorhanden sein muss. Durch das implizite oder explizite Markieren von Daten als cache-fähig dürfen die Anfrage-Daten vom Client für spätere identische Requests wiederverwendet werden. Durch dieses Caching von Daten ist es möglich manche Client-Server Interaktionen teilweise oder ganz zu vermeiden, wodurch die Netzwerkauslastung und Skalierbarkeit verbessert wird. Jedoch birgt die Verwendung eines Caches das Risiko, dass die Daten im Cache im Vergleich zu den auf dem Server gespeicherten Daten schon veraltet sind, was gegebenenfalls zu Fehlern in der weiteren Verarbeitung führen könnte.

Das vierte Prinzip und zentrales Unterscheidungsmerkmal von REST ist das einheitliche Interface zwischen den verschiedenen Komponenten. Hierdurch wird die Systemarchitektur durch das Prinzip der Generalität vereinfacht. Die Schnittstelle ist einfacher benutzbar. Zudem wird eine unabhängige Weiterentwicklung der verschiedenen kommunizierenden Kompoenenten gewährleistet, da die Implementierung der einzelnen Komponenten von den angebotenen Services getrennt wird. Jedoch entsteht durch die einheitliche Schnittstelle auch ein Effizienzverlust, da diese nicht an die Bedürfnisse einer speziellen Anwendung angepasst werden kann. Um ein einheitliches Interface zu erreichen, finden mehrere Beschränkungen auf die Schnittstelle Anwendung: Ressourcen der Schnittstelle sollen eindeutig identifizierbar sein. Eine Ressource ist eine vom Interface bereitgestellte Information, die eindeutig über einen URI identifizierbar ist. Die Informationen, die durch die Ressourcen repräsentiert werden können statisch festgelegt sein, oder sich auch im Zeitablauf verändern. Zudem kann eine Ressource auch existieren, ohne das die Informa-

tion schon existiert. Das erleichtert die Verarbeitung verschiedener Informationsarten, da auf abstrakter Ressourcenebene nicht zwischen bestimmten Typen unterschieden wird. AuSSerdem kann so die benötigte Information auch noch zu einem späten Zeitpunkt, je nach Inhalt der Anfrage, festgelegt werden. Zudem hat jeder Service, der nach den REST Prinzipien entworfen ist eine URL, also eine eindeutige Adresse. Durch diese URL ist der Zugriffsweg zum Webservice standardisiert. Durch diese eindeutig identifizierbaren Ressourcen und Services wird zudem die Kombinierbarkeit verschiedener Ressourcen eines Services bzw. von verschiedenen Services in einem gröSSerem System erleichtert. Eine weitere Beschränkung für die Schnittstelle ist die Verwendung von Repräsentationen zur Veränderung von Ressourcen. Eine Repräsentation ist eine Folge von Bytes, die eine Ressource in einer bestimmten Darstellung und zugehörige Metadaten abbildet. Somit kann eine Ressource vom Server in verschiedenen Repräsentationen, je nach Anfrage, zurückgegeben werden. Zudem werden mit der Repräsentation alle Informationen, wie die Ressource verändert werden kann, mitgeschickt. Veränderungen der Ressource finden nur über die Repräsentation statt. Des weiteren sollen Antworten des Servers auf Anfragen selbsterklärend sein. Dass heiSSt, das Standard-Methoden und -Datentypen verwendet werden, um die Ressource zu verändern oder Informationen auszutauschen. Diese Standard-Methoden sind zwar in REST selbst nicht festgelegt, werden aber normalerweise durch die Verwendung des Protokolls auf der Anwendungsschicht definiert. Für das meistens im Internet verwendete HTTP-Protokoll sind diese z. B. : GET (gewünschte Ressource vom Server anfordern), POST (neue Ressource unterhalb angegebener Ressource einfügen) oder PUT (angegebene Ressource anlegen bzw. ändern). Die letzte Beschränkung wird als ”Hypermedia as the Engine of Application State” bezeichnet. Hiermit ist gemeint, dass die Interaktion mit einer API dynamisch über Hypermedien abläuft. Somit ist auf der Client-Seite nur Basiswissen über Hypertext und fast kein Wissen über die Interaktion mit der spezifischen Schnittstelle nötig. Somit können Client und Server voneinander entkoppelt werden, da der Server dem Client neben den angeforderten Informationen dynamisch mögliche Interaktionen zurückgibt.

Die Systemarchitektur soll zudem in Schichten aufgebaut sein. Das heißt, dass eine Schicht jeweils nur die nächste darunter- und darüberliegende Schicht sehen und mit ihr interagieren kann. Durch diese Architektur wird die Komplexität des Gesamtsystems reduziert und die unabhängige Weiterentwicklung der einzelnen Schichten gefördert. Zudem können veraltete Dienste abgekapselt werden und und neue somit von diesen getrennt

werden. Durch die Aufteilung der Architektur in Schichten kann zudem redundante oder selten benutzte Funktionalität in eine "Zwischenschicht" ausgelagert werden. Durch diese Aulagerung verbessert sich zudem die Skalierbarkeit des Systems, da load-balancing, also die Lastverteilung eines Services auf mehrere Netzwerke oder Prozessoren, ermöglicht wird. Dennoch bringt die eine Schichtenarchitektur auch Nachteile mit sich. Durch die Kapselung der Dienste und Funktionalitäten in Schichten steigt der Verwaltungs- und Wartungsaufwand des Gesamtsystems. Zudem sinkt auch die Geschwindigkeit, mit der Daten verarbeitet werden, da die Anfrage im Verarbeitungsprozess wesentlich mehr Schnittstellen passieren muss. Dieser Nachteil kann jedoch durch die Verwendung von geteilten Caches in den Zwischenschichten kompensiert werden, da durch diese Caches die Anzahl der Schnittstellen, die die Anfrage passieren muss, reduziert werden kann. Ein weiterer Vorteil der Schichtenarchitektur ist, dass die Anfragen selektiv von den einzelnen Schichten verändert werden können, da der Inhalt dieser selbst-beschreibend und die Bedeutung der Nachricht für die Zwischenschichten sichtbar ist.

Die sechste (optionale) Designkonvention von REST besagt, dass wenn nötig Code in Form von Skripten oder Apps über die Schnittstelle vom Client heruntergeladen und ausgeführt werden kann. Dies vereinfacht die Programmlogik des Clients, da weniger Programme schon im Voraus vorhanden sein müssen. Zudem wird dadurch die Erweiterbarkeit eines Systems verbessert, da auch nach dem initalen Installieren eines Systems, dieses noch durch das Bereitstellen von Code über die Schnittstelle erweitert werden kann.

## 2.2 ABAP Restful Application Programming Model

Im Folgenden wird das Restful Application Programming Model für ABAP vorgestellt.

ABAP RAP ist ein Programmiermodell der SAP auf Basis von ABAP, das die Architektur für die Entwicklung von OData-Services, die für die HANA-Datenbank optimiert sind, definiert. Es basiert auf den Designprinzipien nach REST. Mit diesem Modell können sowohl Web APIs veröffentlicht und Business Events erzeugt als auch Fiori Apps entwickelt werden. RAP ist nach dem "Classic ABAP Programming" und dem "ABAP Programming Model for SAP Fiori" die dritte Evolutionsstufe des ABAP Programming Model. RAP kann sowohl in Cloud-, als auch in on-premise Systemen eingesetzt werden.

RAP baut im Allgemeinen auf drei Säulen auf: Anders als in den vorhergehenden Programmiermodellen, wo mehrere Tools zum entwickeln von z. B. einer Fiori App nötig waren, sind sollen in RAP Implementierungsaufgaben in einer Entwicklungsumgebung integriert werden um einen standardisierteren Entwicklungsprozess zu gewährleisten und diesen für den Entwickler zu vereinfachen. Die zweite Säule ist die Programmiersprache ABAP: Durch Erweiterungen und Anpassungen ist es möglich diese für die Entwicklung mit RAP zu verwenden. Hierbei kommen Technologien wie CDS-Views zum Einsatz um aussagekräftige Daten-Modelle zu definieren. Zudem können vorgefertigte APIs für generische Entwicklungsaufgaben verwendet werden. Die dritte Säule sind umfassende Frameworks, die dem Entwickler helfen, effizient und in kurzer Zeit eine Anwendung zu entwickeln, da einzelne Bausteine automatisch generiert werden können und an bestimmten Stellen noch anwendungsspezifische Logik eingefügt werden kann.

Die Architektur eines OData-Services unter Verwendung von RAP wird nachfolgend beschrieben.

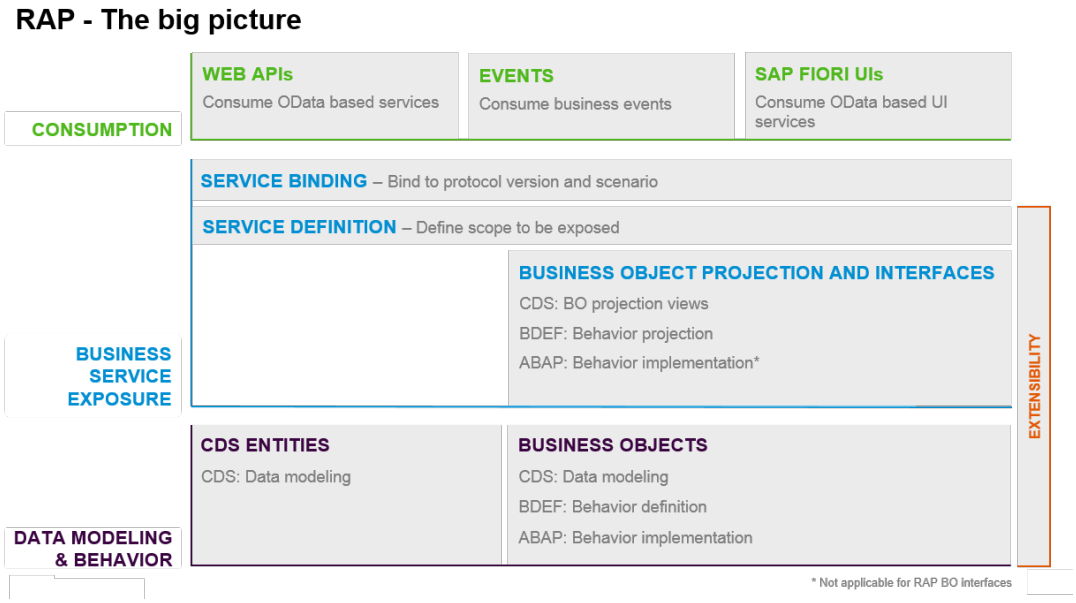


Abbildung 1: RESTful Application Programming Model Architektur

Auf der untersten Ebene werden die benötigen Daten und das beabsichtigte Verhalten modelliert. Dies kann entweder durch CDS-Views oder durch Business Objects geschehen. Core Data Services sind ein Framework um das Datenmodell, basierend auf der HANA-Datenbank, zu definieren und zu organisieren. Es können selektiv die benötigten Daten

aus den Datenbanktabellen ausgewählt werden. Genauer werden alle benötigten Spalten aus einer oder mehreren Tabellen ausgewählt und diese bei Bedarf mit Annotationen versehen, um speziellere Anforderungen zu erfüllen. Es kann bei Bedarf auch noch nach Datensätzen, die gewisse Bedingungen erfüllen gefiltert werden. Somit strukturiert und gruppiert ein CDS-View die benötigten Daten. Die SQL-Abfrage, um diese Daten von der Datenbank abzurufen ist in dem CDS-View integriert. Der Zweck eines CDS-Views ist jedoch lediglich das Lesen und Strukturieren der Daten; es können hiermit keine Daten verändert werden.

Eine andere Möglichkeit ein Datenmodell zu erzeugen sind Business Objects. Diese bieten zudem die Implementierung von Behaviors (Verhalten) und einer Laufzeit. Ein BO ist aus struktureller Sicht ein hierarchisch aufgebauter Baum aus mehreren Knoten, die gewissen Daten entsprechen und durch Eltern-Kind-Beziehungen (sogenannte Kompositionen) miteinander verknüpft sind. Diese Knoten werden durch CDS Entitäten dargestellt. Der hierarchisch oberste Wurzelknoten stellt dabei die Repräsentation des BO an sich dar. Um das Verhalten eines BO zu spezifizieren, muss eine ”Business Object Behavior definition” (”Behavior definition” abgekürzt) angelegt werden. Dieses ABAP Objekt beschreibt das gewünschte Verhalten des BO in RAP. Die Behavior definition bezieht sich immer auf die Wurzel-CDS-Entität eines BO. Diese Behavior definition wird als ABAP Klasse implementiert. Ein Verhalten beschreibt welche Operationen und Feldeigenschaften für ein BO verfügbar sein sollen. Es besteht auSSerdem noch aus der ”Behavior characteristic”, die zusätzliche Eigenschaften, wie z. B. Autorisierungen für Operationen festlegt. Operationen sind z. B. create() für das Erstellen, update() für das Aktualisieren und delete() für das Löschen eines Datensatzes. Diese modify-Operationen verändern im Gegensatz zu den read-Operationen auch die tatsächlichen Daten auf der Datenbank. Die Laufzeit eines BO besteht aus 2 Teilen: In der Interaktionsphase werden durch das Ausführen von Operationen Daten gelesen und/ oder verändert. Diese Veränderungen werden zunächst in einem ”transactional buffer” (Pufferspeicher) gespeichert und nachdem alle Änderungen durchgeführt wurden in der sog. ”save sequence” auf der Datenbank persistiert.

Die zweite Ebene sorgt für das projizieren und von BOs und die Erstellung sowie Veröffentlichung von Business Services. Ein Business Service ist in RAP ein RESTful Service, der Repräsentationen von Ressourcen veröffentlicht, die dann von Konsumenten abgerufen werden können. Die Bestandteile eines Business Services werden später noch genauer erläutert. Die Projektion eines BO ist notwendig, um es flexibel konsumieren

zu können, da dieses an sich komplett unabhängig vom OData-Service ist. Das BO an sich stellt die maximal möglichen Funktionen und Daten bereit, die service-unabhängig implementiert und ggf. durch die Projektion auf die für den Service relevanten Aktionen und Daten eingeschränkt werden. Zudem können genauere Anpassungen z. B. im Bezug auf die Darstellung auf einer Benutzeroberfläche über UI-Annotationen erfolgen, die aber nicht Teil des Datenmodells sein sollen. Eine zusätzliche Projektions-Schicht hat mehrere Vorteile: Zum einen kann das zugrundeliegende BO angepasst und erweitert werden, ohne dass der darauf aufbauende Service davon betroffen ist. Zum anderen können verschiedene Projektions-Views für verschiedene Anforderungen erstellt werden, die alle dasselbe BO wiederverwenden. Zudem können die Daten und Funktionen eines Services für eine Fiori App oder Web API veröffentlicht werden. Des Weiteren können Services auch rollenbaisert veröffentlicht werden, sodass unterschiedliche Daten und Funktionen für unterschiedliche Anwender bereitgestellt werden können. Um eine solche Projektionsschicht zu erstellen, muss zusätzlich eine CDS Projection View erstellt werden, um die speziellen Daten einer Projektion darzustellen. Dieser basiert auf der CDS View des BO und erzeugt selbst keine neue SQL-View, sondern nur eine Repräsentation der dargestellten Entitäten. Um eine CDS-Entität eines BO zu projizieren müssen die Wurzel-Entität sowie alle Eltern-Entitäten ebenfalls projiziert sein. Zudem wird auch eine Projection Behavior Definition benötigt, die alle Verhalten, die für einen speziellen Service veröffentlicht werden sollen projiziert.

Nachdem Teile des BO projiziert wurden und die zugehörigen Artefakte erstellt wurden, muss in der zweiten Ebene ein Service definiert werden. In einer ”business service definition” (abgekürzt ”Service Definition”) wird festgelegt, welche CDS Entitäten eines Datenmodells, also welche Daten, in einen bestimmten Service veröffentlicht werden sollen. Die Service Definition stellt eine protokoll-unabhängige und Konsumenten-spezifische Sichtweise auf das Datenmodell dar. Es können auch mehrere CDS-Entitäten oder eine komplette BO Struktur in einem Service veröffentlicht werden. Dafür muss in der Service Definition die hierarchisch höchste Entität des BO, die veröffentlicht werden soll, markiert werden. Diese dient dann als Einstiegspunkt für den Service.

Als letzter Schritt in der zweiten Ebene muss noch das Kommunikationsprotokoll des Service im Service Binding definiert werden. Ein häufiges Beispiel wäre hierbei OData, für das Bereitstellen von Daten in einer Fiori App. Ein Service Binding bezieht sich immer direkt auf eine oder mehrere Service Definitions. Es können auch mehrere Service

Bindings basierend auf einer Service Definition erstellt werden. Das ist z. B. hilfreich, wenn derselbe Service mit unterschiedlichen Kommunikationsprotokollen veröffentlicht werden soll, da durch die Trennung von Service Definition und Binding das Protokoll von der Geschäftslogik getrennt wird. Somit kann der Entwicklungsaufwand für einen Service erheblich reduziert werden. Ein Service kann für die Protokolle OData Version 2 und 4 veröffentlicht werden, wenn das Ziel ist, die Daten in einer Fiori App darzustellen. OData ermöglicht auSSerdem das Erstellen von HTTP-basierten Services, deren Ressourcen über URIs identifizierbar sind und über HTTP-Nachrichten abgerufen und modifiziert werden können. Dies korrespondiert wiederum mit den Designkonventionen nach REST aus dem vorhergehenden Kapitel. Zudem stehen noch die Protokolle Information Access für Analysezwecke und SQL zur Verfügung. Ein Service kann grundlegend auf zwei Arten veröffentlicht werden: Entweder als UI Service mit den Protokollen OData oder Information Access, indem man dem durch UI-Annotationen eine Fiori Elements oder andere Benutzeroberfläche hinzufügt. Für alle anderen Anwendungsfälle wird der Service als Web API veröffentlicht, die von Clients über das Web mit OData konsumiert werden kann. Wenn ein Service veröffentlicht ist, kann er jedoch im Standard nur innerhalb des Entwicklungssystems abgerufen werden. Ein Service kann zudem in mehreren Versionen existieren. Dies geschieht durch das Hinzufügen oder Entfernen von zusätzlichen Service Definitions zu einem Service Binding. Damit kann ein Service geändert oder erweitert werden.

Die dritte und oberste Ebene der RAP Architektur ist für das Konsumieren der Daten eines BO oder CDS Views verantwortlich. Es gibt zwei Möglichkeiten, diese Daten durch einen Service zu konsumieren. Die erste Möglichkeit ist durch eine Web API. Die Metadaten eines so veröffentlichten Services enthalten keine Informationen über eine Benutzeroberfläche für die Darstellung der Informationen. Der Zugriff auf die bereitgestellten Daten erfolgt über eine öffentliche Schnittstelle des OData Services. Eine weitere Möglichkeit einen Service zu konsumieren ist innerhalb einer Fiori Elements Anwendung als UI Service. Hier werden die Konfigurationen für die Benutzeroberfläche und das Front-End der Anwendung, die im Back-End als Annotationen in den CDS Entitäten festgelegt wurden, über die Metadaten mitgegeben. Somit kann das Fiori Elements Framework aus diesen Metadaten direkt eine fertige UI generieren. Als letzte Möglichkeit ein BO zu konsumieren sind noch Business Events zu nennen. Diese werden hier nur kurz genannt und in einem späteren Kapitel detaillierter beschreiben.

## 2.3 SAP Fiori Elements

Fiori Elements ist ein Framework zum entwickeln benutzerfreundlicher und ansprechender Anwendungen. Apps werden auf Basis von OData-Services und UI-Annotationen in den CDS Entitäten fast automatisch generiert. Somit ist im Gegensatz zur älteren SAP UI5 Freestyle Technologie kein JavaScript Coding mehr nötig um das Front-End zu programmieren. Elements benutzt vordefinierte Layouts und Controller für Aktionen der App.

Fiori Elements bietet drei zentrale Vorteile: Es soll dabei helfen, dass sich Entwickler auf die spezifische Geschäftsprozess-Logik und die Back-End Entwicklung fokussieren und somit weniger Zeit für die Programmierung der Benutzeroberfläche benötigen, was insgesamt zu einer verkürzten Entwicklungszeit von Apps und somit auch für niedrigere Entwicklungskosten sorgt. Zudem wird die Kontinuität des UI über alle Fiori Apps hinweg und die Übereinstimmung der Apps mit den SAP Designkonventionen sichergestellt. Die Benutzer der Apps haben so eine einheitliche Benutzungserfahrung (Layout, Navigation, Suche, ...) über alle Apps hinweg. Zudem stellt das zentrale Framework auch sicher, dass der Programmcode für die Benutzeroberflächen der Apps immer sofort funktioniert und bietet auSSerdem weitere Funktionen wie Übersetzungen und Unterstützung für mobile Entgeräte. Diese Vorteile tragen wiederum zu einer reduzierten Entwicklungszeit und somit einem Kostenersparnis bei.

Apps sind in Fiori Elements immer auf vordefinierten Layouts aufgebaut. Diese können am Anfang der Entwicklung einer App ausgewählt werden und geben das generelle Aussehen und Funktionen vor.

# 3 Praktischer Teil

Im Folgenden werden die verschiedene praktische Lösungsansätze vorgestellt und anhand verschiedener Kriterien gegeneinander abgewogen, sodass am Ende eine Handlungsmatrix erstellt werden kann.

## 3.1 Lösungsansätze

Jetzt werden 3 verschiedene Lösungsansätze vorgestellt, wie man trotzdem sequentielle Prozesse/ asynchrone Kommunikation umsetzen kann

### 3.1.1 Business Workflows

Das ist die bisherige alte/ ineffiziente Lösung. Diese wird zwar noch vorgestellt, soll aber überdacht werden.

### 3.1.2 Business Events

Eine Option wären die Verwendung von Business Events. Hier auch ggf. auf Probleme mit Event-Mesh (Cloud- bzw. BTP-Komponente) für onPremise-Systeme eingehen -> lokale Verarbeitung der Business Events?

### 3.1.3 Background Processing Framework

Andere Option wäre das Background Processing Framework über Background remote function calls.

## 3.2 Entscheidungsmatrix

Hier soll eine Entscheidungsmatrix entwickelt werden, welchen Lösungsansatz man in Abhängigkeit von mehreren Faktoren am besten verwenden soll (ersetzt auch weng mit die Zusammenfassung)

# 4 Schlussbetrachtungen

Im folgenden sollen die wichtigsten Ergebnisse noch einmal zusammengefasst, bewertet und eingeordnet werden. Zudem soll eine Handlungsempfehlung verfasst und die Arbeit einmal kritisch reflektiert werden. Abgeschlossen wird mit einem Ausblick auf weitere Entwicklungen.

## 4.1 Zusammenfassung

Hier nochmal die ganze Arbeit zusammenfassen

## 4.2 Handlungsempfehlung

Konkrete Handlungsempfehlung für die Praxis abgeben

## 4.3 Reflexion der Arbeit und Ausblick

Ergebnisse reflektieren und einen Ausblick auf zukünftige Entwicklungen geben