edv-buchversand.de

SAP Fiori Elements

Grundlagen, Entwicklung und Erweiterung

» Hier geht's direkt zum Buch

DIE LESEPROBE

Diese Leseprobe wird Ihnen von www.edv-buchversand.de zur Verfügung gestellt.

Kapitel 4
Programmiermodelle

SAP Fiori Elements kann in den verschiedenen modernen Programmiermodellen von SAP als UI-Technologie verwendet werden. In diesem Kapitel stellen wir Ihnen die beiden wichtigsten vor.

Bereits seit mehreren Jahrzehnten bietet SAP für seine On-Premise-Kunden die Möglichkeit an, eigene Anwendungen oder Erweiterungen in ABAP auf Basis des *SAP NetWeaver Application Servers (AS) ABAP* zu entwickeln. Um die Jahrtausendwende trug die Popularität von Java dazu bei, dass Java auch bei SAP als weitere Programmiersprache Einzug hielt. Dazu wurde der *SAP NetWeaver AS Java* veröffentlicht. Als erste umfangreiche Applikationen setzten unter anderem SAP NetWeaver Portal (heute SAP Enterprise Portal) und SAP NetWeaver Exchange Infrastructure (SAP XI, heute SAP Process Orchestration) auf diesem Applikationsserver auf.

Nicht nur in den Anfangszeiten, sondern über mehr als ein Jahrzehnt hinweg setzte SAP im Java-Umfeld auf ein eigenes Komponentenmodell für die Entwicklung von Anwendungen. In den letzten Jahren hat sich jedoch einiges verändert. Die traditionellen On-Premise-Systeme haben Konkurrenz durch Cloud-Lösungen erhalten. Plötzlich war es Standard, das Software immer und überall zur Verfügung steht. Einer der Pioniere im Bereich der Cloud-Software war Salesforce (und ist es heute immer noch). Die Software dieses Anbieters kann in der Cloud bezogen werden, ohne eine eigene Infrastruktur für das Hosting der Server betreiben zu müssen.

Doch nicht nur in technischer Hinsicht, sondern auch aus wirtschaftlicher Sicht haben hat sich die Situation fundamental geändert. Daraus erwuchs für SAP die Notwendigkeit, Alternativen zur klassischen Entwicklung auf dem SAP NetWeaver AS ABAP und Java anzubieten und statt auf eigene Komponentenmodelle auf offene Standards zu setzen. Es wurden neue Programmiermodelle entwickelt, die sich auch für die Entwicklung und Erweiterung von Cloud-Anwendungen eignen.

In diesem Kapitel stellen wir Ihnen diese Programmiermodelle vor. Nach einem kurzen Rückblick in die Entstehungsgeschichte der neuen Programmiermodelle in Abschnitt 4.1 führen wir Sie in Abschnitt 4.2 in das *SAP Cloud Application Programming Model* ein. Anschließend erhalten Sie in Abschnitt 4.3 eine Einführung in das *ABAP RESTful Application Programming Model*. Beide Modelle setzen für die Frontend-Entwicklung auf das Framework SAP Fiori Elements.

4.1 Warum braucht es neue Programmiermodelle für die Cloud?

CapEx vs. OpEx

Mit der zunehmenden Bedeutung von Cloud-Software ging der Trend in den Unternehmensführungen weg von den *Capital Expenditures* (CapEx) hin zu *Operational Expenditures* (OpEx):

- Capital Expenditures, zu Deutsch Investitionsausgaben, umfassen alle längerfristigen Investitionen in die Vermögenswerte eines Unternehmens. Diesen Investitionen sind auch einmalige Zahlungen für Infrastruktur und Softwarelizenzen zuzuordnen. Diese Investitionen müssen über die gesamte Nutzungsdauer der Software hinweg abgeschrieben werden.
- Operational Expenditures hingegen, zu Deutsch Betriebsausgaben, umfassen alle Ausgaben, die nötig sind, um einen funktionsfähigen Geschäftsbetrieb operativ zu ermöglichen und auch kontinuierlich zu gewährleisten. Dabei handelt es sich um wiederkehrende Ausgaben, die in der Regel monatlich, quartalsweise oder jährlich bezahlt werden. Sie werden jeweils der Bilanzierungsperiode zugerechnet, in der die Kosten tatsächlich angefallen sind, und in dieser Periode auch in voller Höhe bilanziert.

Gründe für Cloud-Software

Wenn Sie diese beiden Investitionstypen einander gegenüberstellen, können Sie die Motivation für die Einführung von Cloud-Software sowohl aufseiten der Anbieter als auch aufseiten der Kunden verstehen. Cloud-Software lässt sich üblicherweise schneller einführen. Die Metriken der kommerziellen Modelle sind einfach zu verstehen, da sie meist auf der Anzahl der Benutzer-Accounts basieren. Mit Ausnahme der Endgeräte sind keine Investitionen in die Infrastruktur erforderlich. Cloud-Verträge haben meist kürzere Kündigungsfristen. Aufseiten der Anbieter lässt sich außerdem einfacher ein Skalierungseffekt erreichen.

Erste Cloud-Lösungen von SAP

Als Reaktion auf diesen Trend stellte SAP 2007 nach mehrjähriger Entwicklung als erstes Cloud-Produkt *SAP Business ByDesign* vor. Die Lösung galt als kleiner Bruder bzw. kleine Schwester der SAP Business Suite. Zielgruppe waren kleine und mittelständische Unternehmen. SAP konnte damit sehr viel wertvolle Erfahrungen in der Entwicklung und dem Betrieb von Cloud-Lösungen gewinnen. Die Lösung wurde einige Jahre später einem vollständigen Redesign unterzogen, um die Erwartungen der Nutzer*innen an die Performance zu erfüllen.

SAP Business ByDesign verwendete erstmals auch ein eigenes Erweiterungsmodell. In den ersten Versionen setzte die Lösung noch nicht auf SAPUI5 als UI-Technologie, da sie vor der Einführung von SAPUI5 und SAP Fiori entstand. Seit geraumer Zeit setzen aber auch SAP Business ByDesign und das zugehörige Erweiterungskonzept auf SAP Fiori.

Side-by-Side-Erweiterungen

Side-by-Side-Erweiterungen sind Erweiterungen, die parallel zur erweiternden Anwendung auf einer anderen Plattform laufen. Sie müssen nicht in die eigent-

liche Applikation eingreifen, um solche Erweiterungen zu entwickeln. Für Cloud-Lösungen ist es wichtig, dass SAP einen stabilen Kern bereitstellen kann, der nicht durch Erweiterungen der Kunden beeinflusst wird.

Side-by-Side-Erweiterbarkeit ist daher das technologische Herzstück der *Clean-Core-Strategie*. Erweiterungen werden physisch getrennt vom ERP-Kern entwickelt und betrieben – typischerweise auf der *SAP Business Technology Platform* (SAP BTP). Durch diese strikte Entkopplung bleibt der Standardcode im SAP-S/4HANA-System unangetastet, was die Wartbarkeit verbessert und Upgrades erleichtert. Da die Kommunikation ausschließlich über stabile, versionierte APIs und Events erfolgt, lassen sich moderne Technologien wie Agentic AI oder cloudnative Microservices in Sprachen wie Node.js oder Python nutzen, ohne die Stabilität oder Update-Fähigkeit des geschäftskritischen Kernsystems zu gefährden.

Events und APIs

Side-by-Side-Erweiterungen setzen auf zwei wesentliche technologische Konzepte: Events und APIs. Die zu erweiternde Applikation löst *Events* aus, um potenzielle Empfänger wie die Erweiterung darüber zu informieren, dass ein bestimmtes Ereignis eingetreten ist. Solche Ereignisse können beispielsweise die Änderung der Stammdaten eines Kunden oder die Änderung des Status einer Bestellung sein. In der Erweiterung werden darauf aufbauend die relevanten Daten über ein *Application Programming Interface* (API) gelesen und verarbeitet. Zusätzliche Entitäten und Felder werden in der Erweiterung abgelegt. Die APIs aller SAP-Standardprodukte können im *SAP Business Accelerator Hub* (*https://api.sap.com/*) eingesehen werden.

Erweiterung von Benutzeroberflächen

Die Benutzeroberflächen werden ebenfalls in der Erweiterung bereitgestellt, können aber über das SAP Fiori Launchpad in die zu erweiternde Applikation eingebunden werden. Damit haben Anwender*innen das Gefühl, dass es sich um dieselbe Anwendung handelt. Die Erweiterung wird durch dieses Konzept isoliert und verwendet dedizierte Ressourcen. Sollten Sie sich für die Nutzung des Side-by-Side-Erweiterungsmodells entscheiden, können Sie die Entwicklungsmodelle für die ABAP- und Cloud-Entwicklung verwenden, die in den folgenden Abschnitten vorgestellt werden.

SAP Business Technology Platform

Als Technologieplattform für solche Erweiterungen stellt SAP die SAP Business Technology Platform (SAP BTP) bereit. Diese Plattform wurde 2012 erstmals unter dem Namen *SAP HANA Cloud Platform* vorgestellt und erhielt nach einer zwischenzeitlichen Umbenennung in *SAP Cloud Platform* Anfang 2021 ihren aktuellen Namen. Doch mit der Umbenennung wurde nicht nur alter Wein in neuen Schläuchen bereitgestellt, sondern es wurden auch gravierende technologische Änderungen vorgenommen. SAP setzt nun noch mehr auf ereignisgesteuerte Erweiterungen in der Cloud, da sich damit viele Probleme von Kundenerweiterungen lösen lassen und ein stabiler Betrieb des Kerns ermöglicht

wird. Ganz nach dem Motto »Keep the Core clean« werden Side-by-Side-Erweiterungen mit der SAP BTP heute als Mittel der Wahl propagiert.

Cloud vs. on premise

Der Weg in die Cloud war jedoch an anderen Stellen etwas holprig. Als Anbieter hybrider Lösungen, wahrscheinlich auch aus leidvoller eigener Erfahrung, hat SAP mittlerweile erkannt, dass sich Kunden von der Fragestellung Cloud oder on premise rasch verunsichern lassen. Hier und jetzt eine nachhaltige Technologieentscheidung zu treffen und dabei gleichzeitig bestehende Investitionen zu schützen, ist nicht immer einfach.

Programmiermodell für SAP HANA

Ob es da eine glückliche Fügung, Zufall oder eine brillante strategische Entscheidung von SAP war, dass der Vorgänger der SAP BTP zeitlich mit SAP HANA an den Start ging, können wir nicht beurteilen. Aufgrund der initialen Namensgebung SAP HANA Cloud Platform könnte man vermuten, dass es kein Zufall war. SAP entschied sich bei der neuen In-Memory-Plattform für ein Programmiermodell, das sowohl in der Cloud als auch on premise angewendet werden konnte.

Die auf den *SAP HANA Extended Application Services* (SAP HANA XS), d. h. dem in SAP HANA integrierten Applikationsserver, entwickelten Anwendungen basierten noch auf einem teilweise proprietären JavaScript-Framework und Java, basierend auf dem Webserver *Apache Tomcat*. Sie konnten ohne eine einzige Änderung am Quellcode sowohl in der Cloud als auch on premise ausgeführt werden. Mit dem Framework SAP HANA XS JavaScript (*XSJS*) hatte man anfangs vielleicht noch auf das falsche Pferd gesetzt, diese Entscheidung wurde jedoch mit der Einführung der *SAP HANA Extended Application Services, Advanced Model* korrigiert. Fortan setzte SAP ganz auf ein offenes Framework, und zwar *Node.js*.

Sowohl mit Java als auch mit Node.js, dem genauso wie XSJS ebenfalls JavaScript zugrunde liegt, adressiert SAP auch viele Entwickler*innen, die nicht aus dem SAP-Kosmos stammen. Anders, als das bei früheren proprietären SAP-Technologien der Fall war, sind diese Entwickler*innen heute ohne SAP-Spezialwissen in der Lage, SAP-Erweiterungen und eigenständige Lösungen zu entwickeln. Und durch künstliche Intelligenz werden die Karten noch einmal neu gemischt. Unterstützt durch Joule, Claude, ChatGPT, Gemini und Co. ist es nun sogar Nicht-SAP-Entwickler*innen möglich, SAP-Anwendungen zu entwickeln.

SAP Cloud Application Programming Model

Auch mit der Einführung des *SAP Cloud Application Programming Models* (CAP) im Jahr 2019 hatte SAP vor allem Nicht-SAP-Entwickler*innen im Fokus. Dem Mythos, dass SAP-Technologie kompliziert, schwergewichtig und kaum zu erlernen ist, hat SAP etwas entgegengesetzt. In Abschnitt 4.2 führen wir in die Aspekte des SAP Cloud Application Programming Models ein, die für die SAP-Fiori-Entwicklung relevant sind.

ABAP RESTful Application Programming Model

SAP hat aber nach wie vor eine Millionengefolgschaft klassischer SAP-Entwickler*innen, die SAP-Produkte auf Basis der Programmiersprache *ABAP* erweitern und anpassen. Diese Entwicklergemeinde darf SAP in der modernen Cloud-Welt natürlich nicht vernachlässigen, sofern man sie nicht verlieren möchte. Eine Umschulung auf Java oder JavaScript ist in vielen Fällen nicht zielführend, da sie mit hohen Kosten und einer längeren Einarbeitungsphase verbunden wäre. Mit dem *ABAP RESTful Application Programming Model*, inoffiziell auch RAP abgekürzt, adressiert SAP diese Zielgruppe. Das Modell kann sowohl für Cloud- als auch für On-Premise-Erweiterungen angewendet werden. Auf die für die SAP-Fiori-Entwicklung relevanten Aspekte dieses Programmiermodells gehen wir in Abschnitt 4.3 ein.

Sowohl mit dem SAP Cloud Application Programming Model als auch mit dem ABAP RESTful Application Programming Model können Sie aktuell nichts falsch machen. Beiden Modellen liegen *Core Data Services* (CDS) als Technologie zugrunde. Eine weitere Gemeinsamkeit ist, dass Anwendungen beider Modelle ohne Änderungen am Quellcode sowohl auf der SAP BTP als auch in einer On-Premise-Landschaft betrieben werden können. Ihre Investitionen bleiben also geschützt. Auch wenn Sie zunächst einmal mit der On-Premise-Entwicklung starten, können Sie Ihre Entwicklungen nach einer Cloud-Transformation noch weiterverwenden.

4.2 SAP Cloud Application Programming Model

Das SAP Cloud Application Programming Model stellt die fachlichen Anforderungen an die Anwendungsentwicklung in den Vordergrund. Wie diese Entwicklung umgesetzt wird, also die zugrunde liegenden Technologien, wird durch das Programmiermodell abstrahiert. Dadurch wird auch die Zusammenarbeit zwischen den Experten*innen aus den Fachbereichen und den Entwickler*innen erleichtert. Eine wesentliche Rolle spielt dabei CDS als leistungsfähige Sprache zur Erfassung von bereichspezifischen Datenmodellen.

Übergeordnete Konzepte und Best Practices

Mit der sich schnell verändernden Welt der Cloud-Technologien und -Plattformen Schritt zu halten, stellt eine große Herausforderung dar – sowohl auf Hersteller- als auch auf Kundenseite. Heute aktuelle Technologien können übermorgen bereits überholt sein. SAP vermeidet eine einseitige Fokussierung beim SAP Cloud Application Programming Model durch übergeordnete Konzepte und APIs. Letzten Endes haben Entwicklungs- und Architekturteams die Kontrolle darüber, welche Tools oder Technologien sie wählen oder welche Architekturmuster sie befolgen.

Framework

Das SAP Cloud Application Programming Model ist ein Framework aus Programmier- und Designsprachen, Bibliotheken und Werkzeugen für die Entwick-

lung von Services und Anwendungen im professionellen Unternehmensumfeld. Es gibt Ihnen Best Practices an die Hand und stellt eine Vielzahl sofort einsetzbarer Lösungen für wiederkehrende Aufgaben bereit. Das SAP Cloud Application Programming Model ermöglicht Ihnen die Entwicklung von *Full-Stack-Applikationen*. Darunter versteht man alle Komponenten einer Applikation von der *Persistenzschicht*, über die *Geschäftslogik* und *Servicebereitstellung* bis hin zur Benutzeroberfläche. Das Framework besteht aus einer Mischung aus bewährten sowie weit verbreiteten Open-Source-Technologien und SAP-Technologien.

Unterstützte Sprachen

Mit diesem Programmiermodell entwickelte Anwendungen werden entweder in JavaScript oder in Java entwickelt – beides Open-Source-Frameworks. Bei der JavaScript-Entwicklung wird auf *Node.js* in Kombination mit dem *Express Framework* gesetzt. Node.js ist der De-facto-Standard für die clientseitige JavaScript-Entwicklung. Das Express Framework ermöglicht eine einfache Entwicklung und Bereitstellung von Anwendungen auf Basis von Node.js.

Bei der Java-Entwicklung wird das Framework *Spring* verwendet. Darauf aufbauend verwendet das SAP Cloud Application Programming Model CDS für die Modellierung der Persistenzschicht und der Serviceschicht. Dabei handelt es sich um eine domänenspezifische Sprache, die von SAP entworfen wurde. Zusätzlich werden Service Software Development Kits (SDKs) eingesetzt, um beispielsweise auf SAP S/4HANA Cloud zuzugreifen.

Lokale oder Cloud-Entwicklung

Mit dem SAP Cloud Application Programming Model können Sie sowohl lokal als auch in der SAP BTP entwickeln. Sollten Sie die lokale Entwicklung bevorzugen, müssen Sie Node.js und den zugehörigen Node Package Manager (npm) auf Ihrem Rechner installieren. Anschließend können die Anwendungen auf der SAP BTP bereitgestellt werden.

Datenbanken

Die mit dem Programmiermodell entwickelten Applikationen verwenden entweder SQLite oder SAP HANA als Datenbank für die Speicherung der Daten. *SQLite* ist eine minimalistische Datenbank, die als Open-Source-Software bereitgestellt wird. Während der Entwicklung kann zusätzlich die In-Memory-Datenbank *H2* verwendet werden. H2 ist eine in Java entwickelte relationale Datenbank. Dabei werden die Daten jedoch nach jedem Neustart der Anwendung gelöscht. Die SAP-HANA-Datenbank kann sowohl in der Cloud als auch on premise verwendet werden. Die beliebte Open-Source-Datenbank *Postgres* kann ebenfalls verwendet werden.

Einsatzszenarien

Das SAP Cloud Application Programming Model wird im Wesentlichen für zwei Szenarien eingesetzt:

- Zum einen steht die Entwicklung neuer Applikationen im Fokus. Mit dem Programmiermodell können Sie im Handumdrehen Applikationen erstellen, die Sie in der Cloud und on premise bereitstellen können. Auch die Mandantenfähigkeit, die SAP als Anbieter von ERP-Lösungen nahezu unschlagbar

gemacht hat, wird vom SAP Cloud Application Programming Model standardmäßig unterstützt. Damit haben Sie die Möglichkeit, Anwendungen für verschiedene Mandanten unabhängig und vollständig isoliert voneinander bereitzustellen.

- Das zweite Einsatzszenario ist die Erweiterung von SaaS-Anwendungen. Diese werden üblicherweise ereignisgesteuert entwickelt. In der SaaS-Anwendung wird ein Ereignis ausgelöst, beispielsweise die Änderungen der Daten eines Kunden. Die Erweiterung registriert sich auf dieses Ereignis und führt bestimmte Funktionen aus, sobald es eintritt.

SAP überlässt im SAP Cloud Application Programming Model den Entwickler*innen die Entscheidung, welche UI-Technologie sie verwenden möchten. Die Empfehlung geht aber ganz eindeutig in Richtung SAP Fiori Elements. Das SAP Cloud Application Programming Model unterstützt alle für SAP-Fiori-Elements-Applikationen erforderlichen Annotationen.

Beispielapplikation

Im Folgenden möchten wir Ihnen anhand eines Beispiels zeigen, wie das SAP Cloud Application Programming Model angewendet wird. Die Beispielanwendung haben wir lokal in *Visual Studio Code* erstellt (siehe auch Abschnitt 5.2.2). Mit dieser IDE lässt sich die Effizienz der Entwicklung im Vergleich zum SAP Business Application Studio unserer Erfahrung nach signifikant steigern.

Projekt anlegen

Im ersten Schritt initialisieren Sie mit dem Befehl `cds init` ein neues CAP-Projekt über die Kommandozeile:

```
cds init clouddna
```

Die Struktur des Projekts wird durch das SAP Cloud Application Programming Model vorgegeben (siehe Abbildung 4.1). Datenmodell, Servicebereitstellung und Benutzeroberfläche werden getrennt voneinander vorgehalten.

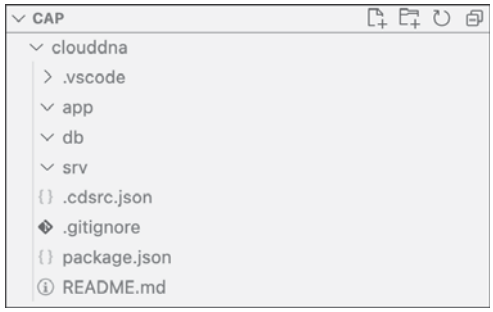

Abbildung 4.1: Projektstruktur im SAP Cloud Application Programming Model

Datenmodell

Das Datenmodell wird im Verzeichnis *db* angelegt. Als Dateierweiterung wird *.cds* verwendet. Der Name der Datei spielt dabei keine Rolle. In unserem Beispiel haben wir eine Datei namens *model.cds* angelegt (siehe Abbildung 4.2).

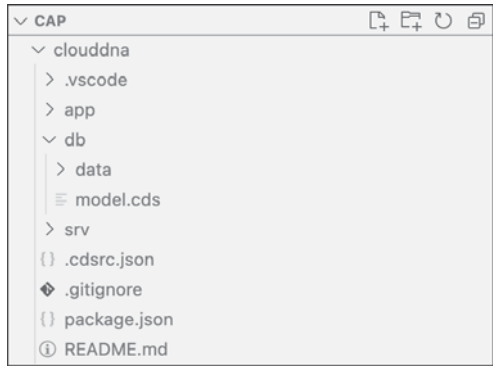

Abbildung 4.2: Datenmodell innerhalb des Verzeichnisses »db«

In dieser Datei muss zunächst ein *Namensraum* (Namespace) deklariert werden, um Namenskonflikte mit anderen eigenen Projekten oder Projekten von SAP zu vermeiden. In unserem Beispiel legen wir die Entitäten »Kunde« und »Adresse« im Namensraum `at.clouddna.cap` an (siehe Listing 4.1). Beide Entitäten werden um den Aspekt `managed` erweitert. Ein *Aspekt* beinhaltet bestimmte, häufig benötigte Eigenschaften. Dabei handelt es sich um ein von SAP bereitgestelltes Artefakt, das über die Anfrage `using` referenziert werden muss.

```
using { managed, sap } from '@sap/cds/common';
namespace at.clouddna.cap;

entity Kunde: managed {
    key ID:         Integer;
        vorname:    String(50);
        nachname:   String(50);
        email:      String(100);
        telefon:    String(30);
        adresse:    Association to Adresse;
}

entity Adresse: managed {
    key ID:         Integer;
        land:       String(50);
        region:     String(50);
        plz:        String(5);
        ort:        String(100);
        strasse:    String(100);
        kunde:      Association to one Kunde;
}
```

Listing 4.1: CDS-Datenmodell einer CAP-Anwendung

Über das Artefakt `managed` werden die Entitäten mit den zusätzlichen Attributen `createdAt`, `createdBy`, `modifiedAt` und `modifiedBy` angereichert. Diese Felder werden zur Laufzeit automatisch befüllt und geben Ihnen die Möglichkeit, festzustellen, welcher Benutzer oder welche Benutzerin zu welchem Zeitpunkt einen Eintrag in der darunterliegenden Datenbank angelegt oder geändert hat.

Den einzelnen Attributen einer Entität müssen Datentypen zugeordnet werden. Zusätzlich besteht die Möglichkeit, Fremdschlüsselbeziehungen abzubilden. In unserem Beispiel gibt es eine solche Beziehung (`Association to`) zwischen den Entitäten Kunde und Adresse.

Sie müssen das Modell nicht in einer einzigen Datei erstellen, sondern können es auf mehrere Dateien verteilen. Das vereinfacht es, in Teams zu arbeiten, und die Modelle können besser strukturiert werden.

Anwendung starten

Nachdem das Datenmodell angelegt wurde, kann die Applikation bereits über den folgenden Befehl gestartet werden:

```
cds serve
```

Dabei werden Ihnen gegebenenfalls Syntaxfehler angezeigt.

Beim Start der Applikation haben Sie auch die Möglichkeit, über einen *CSV-Import* Daten in die Datenbank einzuspielen. Dazu müssen Sie im Verzeichnis *db* oder in einem Unterverzeichnis eine Datei mit der Endung *.csv* anlegen. Dabei muss eine bestimmte Namenskonvention eingehalten werden. Der Dateiname muss wie folgt aufgebaut sein: *<Namespace>-<Entität>.csv* (siehe Abbildung 4.3).

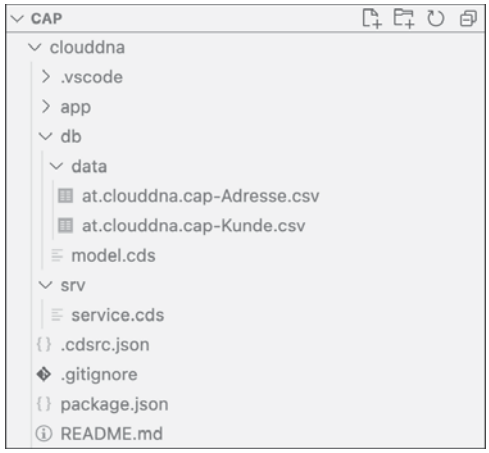

Abbildung 4.3: CSV-Dateien für einen Datenimport

OData-Service

Nach der Anlage des Datenmodells in CDS können Sie im nächsten Schritt den OData-Service erstellen. Dazu ist keinerlei Programmiererfahrung erforderlich.

Die Bereitstellung des Service erfolgt in Form von Dateien, die Sie mit der Erweiterung *.cds* im Verzeichnis *srv* oder einem Unterverzeichnis anlegen müssen (siehe Abbildung 4.4).

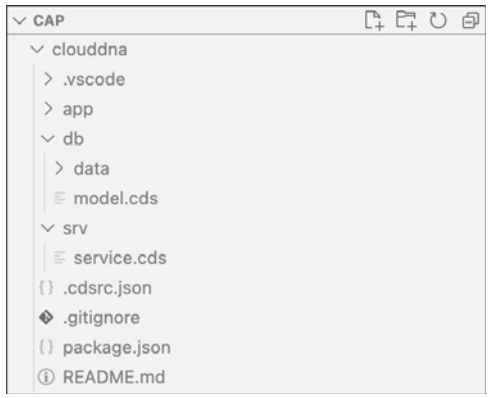

Abbildung 4.4: OData-Service im Verzeichnis »srv«

Für unser Beispiel haben wir einen Service in der Datei *service.cds* angelegt. Sie können Ihren Service auch auf verschiedene Dateien aufteilen. Das SAP Cloud Application Programming Model arbeitet im Standard mit OData-Services der Version V4. Für die Bereitstellung der im Datenmodell definierten Entitäten ist es lediglich erforderlich, eine *Projektion* zu erstellen. Dazu müssen Sie das zuvor definierte Datenmodell über die Anfrage `using` referenzieren und unter einem Alias zur Verfügung stellen. Im Beispiel in Listing 4.2 wurde der Alias `cap` verwendet.

```
using { at.clouddna.cap as cap } from '../db/model';

service UserService {
    entity Kunde as projection on cap.Kunde;
    entity Adresse as projection on cap.Adresse;
}
```

Listing 4.2: OData-Service über eine Projektion generieren

Standardmäßig werden Ihnen alle CRUDQ-Methoden für jede Entität bereitgestellt. Mit diesen Methoden können Sie Funktionen für das Anlegen, Ändern, Lesen und Löschen von Daten bereitstellen, ohne eine einzige Zeile Code zu programmieren (siehe auch Abschnitt 3.2, »CRUDQ-Methoden«).

Nachdem Sie die Anwendung noch einmal mit dem Befehl `cds serve` gestartet haben, ist diese auf dem `localhost` unter dem Port 4004 erreichbar. Dort werden alle Webapplikationen und die Serviceendpunkte des Projekts mit dem zugehörigen Metadatendokument bereitgestellt (siehe Abbildung 4.5).

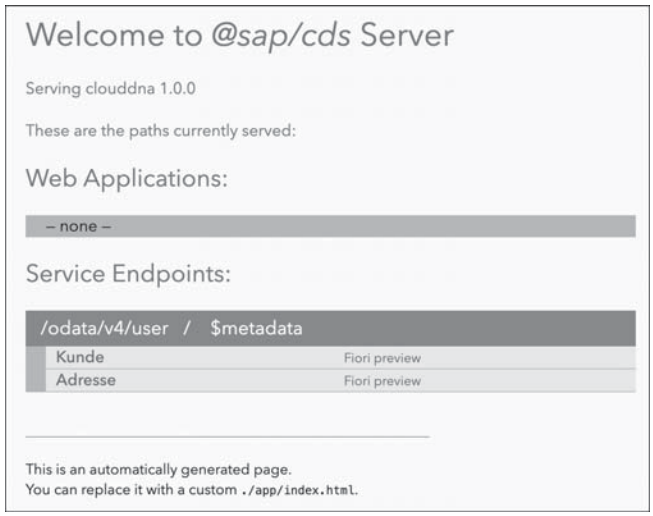

Abbildung 4.5: Startseite der CAP-Anwendung

In Listing 4.3 sehen Sie, wie über einen zusätzlichen Service ausschließlich lesender Zugriff auf die Entität erlaubt wird.

```
service SupportService @(path:'/support') {
    @readonly entity Kunde as SELECT from cap.Kunde { * }
        excluding { createdBy, modifiedBy };
}
```

Listing 4.3: Bereitstellung einer Entität mit Lesezugriff

SAP-Fiori-Benutzeroberfläche

Auf der Startseite der CAP-Anwendung wird für jede Entität die Vorschau einer SAP-Fiori-Benutzeroberfläche (**Fiori Preview**) bereitgestellt. Dazu wird dynamisch eine SAP-Fiori-Elements-Applikation erzeugt. Diese automatisch generierte Anwendung sollte lediglich zu Testzwecken verwendet werden.

UI-Annotationen

Das Aussehen der Anwendung wird über Annotationen gesteuert (siehe Abschnitt 3.5, »OData- und CDS-Annotationen«). Diese können Sie in der Servicedatei mit der Endung *.cds* oder in einer eigenen *.cds*-Datei im Verzeichnis *srv* einsehen und anpassen. In Listing 4.4 sehen Sie, wie die Entität `Kunde` mit Annotationen aus dem UI-Vokabular versehen wurde. Das Beispiel beschränkt sich auf zwei Annotationen: `SelectionFields` und `LineItem`. Mit den Werten der Annotation `SelectionFields` steuern Sie, welche Filter angezeigt werden. Mit den Werten der Annotation `LineItem` steuern Sie, welche Felder in einer Tabelle angezeigt werden.

```
annotate UserService.Kunde with @(
    UI: {
        SelectionFields: [ ID, nachname, vorname ],
```

```
        LineItem: [
            {   Value: ID, Label: 'Kundennummer' },
            {   Value: vorname, Label: 'Vorname'},
            {   Value: nachname, Label: 'Nachname' },
            {   Value: email, Label: 'Email Adresse' },
            {   Value: telefon, Label: 'Telefonnummer' }
        ]
    }
);
```

Listing 4.4: UI-Annotationen im Datenmodell einer CAP-Anwendung

Vorschau der Anwendung

Änderungen, die Sie an diesen Annotationen vornehmen, spiegeln sich direkt in der Vorschau der Anwendung wider (siehe Abbildung 4.6).

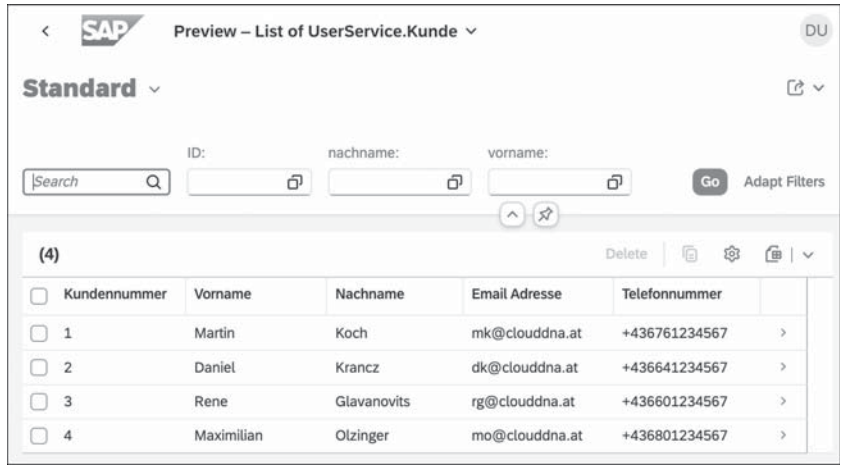

Abbildung 4.6: SAP-Fiori-Elements-Vorschau der CAP-Applikation

Mit der Anzeige der Namen der Filter in der Vorschau in Abbildung 4.6 sind wir noch nicht zufrieden. Anstelle des Textes »ID« möchten wir beispielsweise den Text »Kundennummer (Filter)« für das erste Filterelement anzeigen.

Durch die Verwendung attributspezifischer Annotationen, hier der Annotation `@titel`, können Sie den auf der Benutzeroberfläche angezeigten Text beeinflussen, wie in Listing 4.5 gezeigt.

```
annotate UserService.Kunde with {
    ID @title : 'Kundennummer (Filter)';
    nachname @title : 'Nachname (Filter)';
    vorname @title : 'Vorname (Filter)';
}
```

Listing 4.5: Annotationen für bestimmte Attribute

Die geänderten Annotationen spiegeln sich dann in der Vorschau wider (siehe Abbildung 4.7).

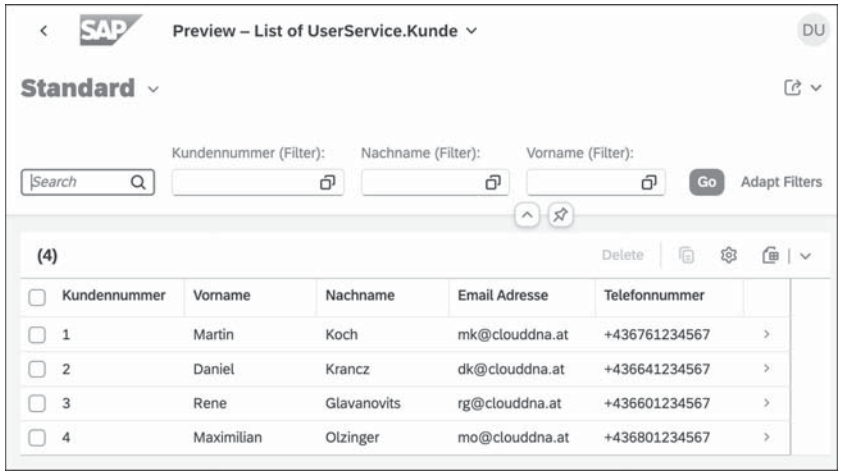

Abbildung 4.7: Vorschau der CAP-Applikation nach Ergänzung der Title-Annotationen

Template Wizard

Letztendlich ist es das Ziel, die SAP-Fiori-Benutzeroberfläche gemeinsam mit der CAP-Anwendung auszuliefern. Das SAP Cloud Application Programming Model ermöglicht es, eine oder mehrere SAP-Fiori-Applikation(en) im Verzeichnis *app* auszuliefern. Für jede Anwendung muss dazu ein eigenes Unterverzeichnis angelegt werden. Die SAP-Fiori-Applikationen können Sie mithilfe eines Assistenten erstellen, dem *Template Wizard*, der über ein Plug-in in Visual Studio Code integriert wird (siehe Abbildung 4.8).

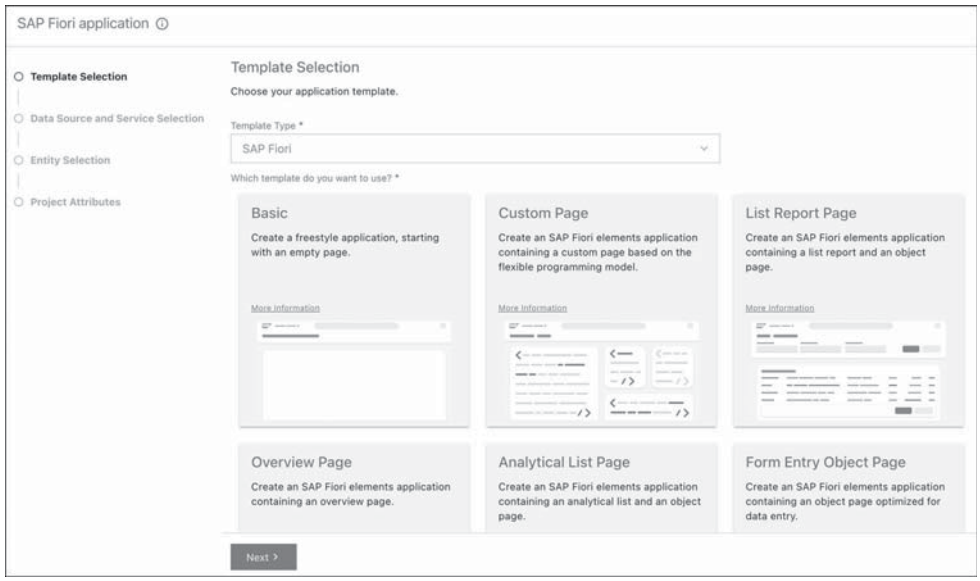

Abbildung 4.8: Template Wizard zur Generierung von SAP-Fiori-Elements-Applikationen

Hier können Sie verschiedene SAP-Fiori-Elements-Floorplans für den Aufbau der Seiten auswählen, die wir Ihnen in Teil II, »Entwicklung mit Floorplans«, noch ausführlich vorstellen werden.

Dieser Wizard erstellt eine vollständige Verzeichnisstruktur für Ihre SAP-Fiori-Elements-Applikation im Verzeichnis *app* (siehe Abbildung 4.9).

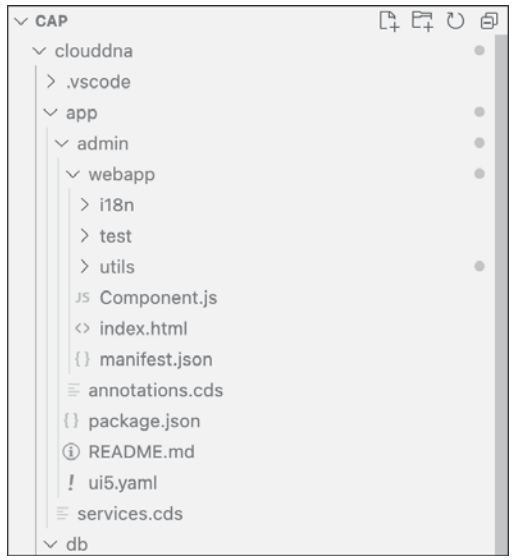

Abbildung 4.9: Verzeichnisstruktur einer generierten SAP-Fiori-Elements-Applikation

Starten Sie die CAP-Applikation nun erneut, wird der Absprung zur SAP-Fiori-Elements-Applikation auf der Startseite angeboten (siehe Abbildung 4.10).

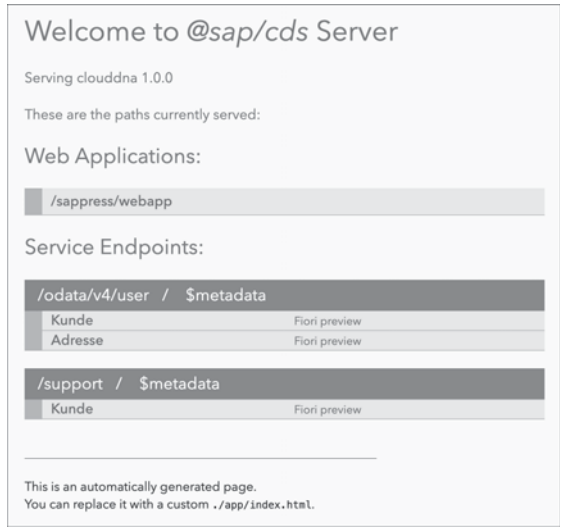

Abbildung 4.10: Startseite mit dem Absprung auf die SAP-Fiori-Elements-Applikation

Wir haben Ihnen an dieser Stelle bewusst nur einen kleinen Teil des SAP Cloud Application Programming Models gezeigt, um Ihnen einen Eindruck zu vermitteln, was mit diesem Programmiermodell möglich ist. Unsere Beispielanwendung konnten wir in wenigen Minuten vollständig erstellen. Das SAP Cloud Application Programming Model ist daher aus unserer Sicht ein Game-Changer, wenn es um die Entwicklung moderner SAP-Applikationen geht. Sie müssen kein SAP-Vorwissen mitbringen, um SAP-Anwendungen zu entwickeln. Damit adressiert das Programmiermodell auch eine potenziell sehr große Zielgruppe.

4.3 ABAP RESTful Application Programming Model

Aus der Perspektive von SAP erscheint es sinnvoll, die Cloud-Entwicklung auch für ABAP-Entwickler*innen zu ermöglichen. So kann man die neue Technologie mit der Erfahrung der großen ABAP-Entwicklergemeinde zusammenbringen. Bestehendes Know-how kann auch in der Cloud weiter angewendet werden. SAP hat dazu das ABAP RESTful Application Programming Model entwickelt. Applikationen, die mit diesem Programmiermodell entwickelt werden, können sowohl in der SAP BTP als auch auf SAP-S/4HANA-Systemen on premise ausgeführt werden. Ähnlich wie mit dem SAP Cloud Application Programming Model hat SAP damit gleich zwei Fliegen mit einer Klappe geschlagen.

Evolution der Programmiermodelle

Das ABAP RESTful Application Programming Model hat mit der klassischen ABAP-Entwicklung wenig zu tun. Vielmehr handelt es sich um eine modernes, vollständig auf CDS und SAP Fiori basierendes Programmiermodell. In der Evolution der ABAP-Programmiermodelle stellt das ABAP RESTful Application Programming Model aktuell die letzte Stufe dar. Das klassische ABAP-Programmiermodell, das sich über Jahrzehnte hinweg bewährt hatte, wurde 2012 zunächst durch das *ABAP-Programmiermodell für SAP Fiori* abgelöst. Dessen Nachfolger wiederum ist das ABAP RESTful Application Programming Model. Da dieses Programmiermodell nur in der Cloud und auf SAP S/4HANA ausgeführt werden kann, behalten die Vorgängermodelle für ältere Releasestände aber weiterhin ihre Gültigkeit.

Säulen des Programmiermodells

Das ABAP RESTful Application Programming Model hat die Entwicklung von ABAP-Anwendungen auf der Basis von OData-Services vereinheitlicht. Es basiert auf drei Säulen, die Ihnen die Entwicklung erleichtern:

- **Werkzeuge**
  Mit den in die Entwicklungsumgebung *Eclipse* integrierten *ABAP Development Tools* (ADT) können alle Implementierungsaufgaben durchgeführt werden. Damit lässt sich der Arbeitsablauf in Entwicklungsprojekten optimieren. Neu eingeführte Entwicklungsartefakte unterstützen Sie dabei, die Entwicklung zu standardisieren. Zu diesen Artefakten zählen beispielsweise

CDS Views, Metadatenerweiterungen oder ABAP Managed Database Procedures (AMDP).

- **Sprache**
  Die ABAP-Programmiersprache wurde für das ABAP RESTful Application Programming Model angepasst, modernisiert und erweitert. Ziel war dabei, CDS zu unterstützen und die Erweiterungen von SAP-Cloud-Anwendungen zu ermöglichen, ohne die Stabilität des Anwendungskerns zu gefährden. Befehle, die im Cloud-Kontext nicht sinnvoll sind, können mit dem ABAP RESTful Application Programming Model daher nicht mehr verwendet werden. Sie verwenden typisierte APIs für Standardimplementierungsaufgaben und profitieren von Funktionen wie Autovervollständigung, statischen Codeprüfungen und Elementinformationen.
- **Frameworks**
  Leistungsfähige und moderne Frameworks bilden die dritte wichtige Säule des Programmiermodells. Diese Frameworks, zu denen beispielsweise SAP Gateway und das Orchestration Framework zählen, unterstützen Sie bei Standardimplementierungsaufgaben.

Architektur

Abbildung 4.11 stellt die Architektur des ABAP RESTful Application Programming Models dar. Die unterste Ebene bildet die Datenmodellierung auf Basis von Business-Objekten. Als Zwischenschicht dient die Servicebereitstellung in Form von OData-Services. Auf Ebene der Servicekonsumierung können SAP-Fiori-Elements-Applikationen oder API-basierte Webapplikationen verwendet werden. Im Folgenden gehen wir genauer auf die einzelnen Komponenten dieser Architektur ein.

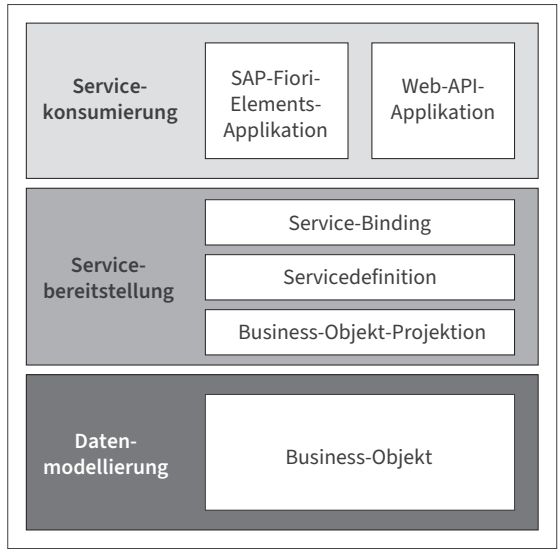

Abbildung 4.11: Architektur des ABAP RESTful Application Programming Models

Business-Objekte

Ein *Business-Objekt* (BO) ist ein gängiges Artefakt in der Entwicklung von Unternehmensanwendungen. Es bildet ein Objekt in der Unternehmenswelt ab, beispielsweise eine Bestellung, ein Produkt oder einen Geschäftspartner. Strukturell gesehen besteht ein Business-Objekt aus einem hierarchischen Baum, der aus mehreren Knoten besteht. So enthält eine Bestellung beispielsweise mehrere Bestellpositionen, die sich wiederum aus mehreren Produkten aufbauen. Die Knoten dieser Struktur werden durch eine spezielle Art von Assoziationen, einer sogenannten *Komposition* miteinander verbunden. Jeder Knoten dieses Kompositionsbaumes wird als CDS-Entität modelliert. Die Wurzelentität repräsentiert das Business-Objekt nach außen. Es stellt den obersten Knoten innerhalb der Hierarchie dar.

Behavior

Für jedes Business-Objekt kann ein Verhalten (*Behavior*) definiert werden. Damit werden die Operationen und Feldeigenschaften eines einzelnen Business-Objekts definiert. Eine *Behavior Definition* enthält also Verhaltenscharakteristika und eine Reihe von Operationen für jede Entität entlang des Kompositionsbaumes eines Business-Objekts. Das Verhalten wird mit der *Behavior Definition Language* (BDL) modelliert.

Projektionsschicht

Das Business-Objekt ist grundsätzlich dienstunabhängig. Das bedeutet, dass es unabhängig von jeglicher OData-Service-Schicht erstellt wird. Das Business-Objekt umfasst daher die maximale Bandbreite an Funktionen, die von einem spezifischen OData-Service, der dieses Business-Objekt bereitstellt, d. h. *projiziert*, genutzt werden können.

Die *Projektionsschicht* ist die erste servicespezifische Schicht, die im Verlauf des Entwicklungsprojekts erstellt wird. Mit der Projektion des Business-Objekts definieren Sie sein reales Verhalten, und zwar in einem OData-Service. Durch diese Zwischenschicht sind Sie sehr flexibel darin, wie Sie das Business-Objekt in verschiedenen OData-Services verwenden wollen. Das zugrunde liegende Business-Objekt kann beispielsweise erweitert werden, ohne dass sich dies auf bereits bestehende OData-Services auswirkt.

Interface View

Die Projektion wird mithilfe eines CDS Views erstellt. Üblicherweise wird dazu mit Interface und Consumption Views gearbeitet. Ein *Interface View* blickt als direkte Projektion auf eine darunterliegende Datenbanktabelle. Er beinhaltet keine Annotationen aus dem UI-Vokabular. Listing 4.6 zeigt ein Beispiel für einen Interface View aus einer unserer Demo-Anwendungen.

```
@AbapCatalog.sqlViewName: 'ZMOXIS_DEMO_CDSS'
@AbapCatalog.compiler.compareFilter: true
@AbapCatalog.preserveKey: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'Moxis Demo'
@Metadata.allowExtensions: true
define view entity ZI_MOXIS_DEMO_CDS as select from /moxis/doc
```

```
{
    key guid as Guid,t,
    order_text as OrderText,
    moxis_job_descr as MoxisJobDescr,
    moxis_order_id as MoxisOrderId,
    created_by as CreatedBy,
    created_date as CreatedDate,
    moxis_job_status as MoxisJobStatus,
    document_name as DocumentName
}
```

Listing 4.6: CDS-Projektion mit einem Interface View

Consumption View

Ein *Consumption View* basiert auf mehreren Interface Views. Er beinhaltet Annotationen aus dem UI-Vokabular und ist für die Verwendung in einer bestimmten SAP-Fiori-Applikation bestimmt. Listing 4.7 zeigt ein Beispiel für einen Consumption View.

```
@AbapCatalog.sqlViewName: 'ZC_MOXIS_DEMO'
@AbapCatalog.compiler.compareFilter: true
@AbapCatalog.preserveKey: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'Moxis Demo Consumption View'
define view ZC_MOXIS_DEMO_CDS as select from ZI_MOXIS_DEMO_CDS
{

    @UI.hidden: true
    key Guid,

    @UI.lineItem: [{position:10, importance: #HIGH,
                    label: 'Job Beschreibung'}]
    OrderText,

    @UI.selectionField: [{position:20}]
    @UI.lineItem: [{position:20, importance: #MEDIUM,
                    label: 'Moxis Status'}]
    MoxisJobStatus,

    @UI.selectionField: [{position:40}]
    @UI.lineItem: [{position:20, importance: #MEDIUM,
                    label: 'Erstellt am'}]
    CreatedDate,

    @UI.selectionField: [{position:30}]
    @UI.lineItem: [{position:30, importance: #HIGH,
```

```
                    label: 'Erstellt von'}]
    CreatedBy
}
```

Listing 4.7: Consumption View mit verschiedenen UI-Annotationen

Servicedefinition

Auf diesen CDS Views setzt die Business-Service-Schicht auf. Die *Servicedefinition* beschreibt, welche CDS-Entitäten eines Datenmodells exponiert werden sollen, damit eine bestimmte Geschäftsanforderung umgesetzt werden kann. Bei der Servicedefinition handelt es sich um ein ABAP-Repository-Objekt, das die konsumentenspezifische Sicht auf ein Datenmodell beschreibt. Die Servicedefinition ist protokollunabhängig, d. h. nicht von der Version oder dem Typ des verwendeten Protokolls abhängig. Ein Beispiel für eine Servicedefinition finden Sie in Listing 4.8.

```
@EndUserText.label: 'Moxis Demo'
define service ZMOXIS_UI_DEMO {
  expose ZC_MOXIS_DEMO_CDS;
}
```

Listing 4.8: Servicedefinition

Service-Binding

Das *Service-Binding* ist ein weiteres ABAP-Repository-Objekt, das dazu dient, die Servicedefinition an ein Client-Server-Kommunikationsprotokoll zu binden. Im SAP-Kontext handelt sich bei dem Kommunikationsprotokoll üblicherweise um OData. Wie jedes andere ABAP-Repository-Objekt nutzt das Service-Binding die bewährte Infrastruktur der ABAP Workbench, einschließlich der Transportfunktionalität. Das Service-Binding wird unter Verwendung der ADT erstellt. In Form des Binding-Typs geben Sie an, mit welcher OData-Version und welchem Konsumierungstyp ein Service bereitgestellt werden soll (siehe Abbildung 4.12). Aktuell werden OData V2 und V4 unterstützt.

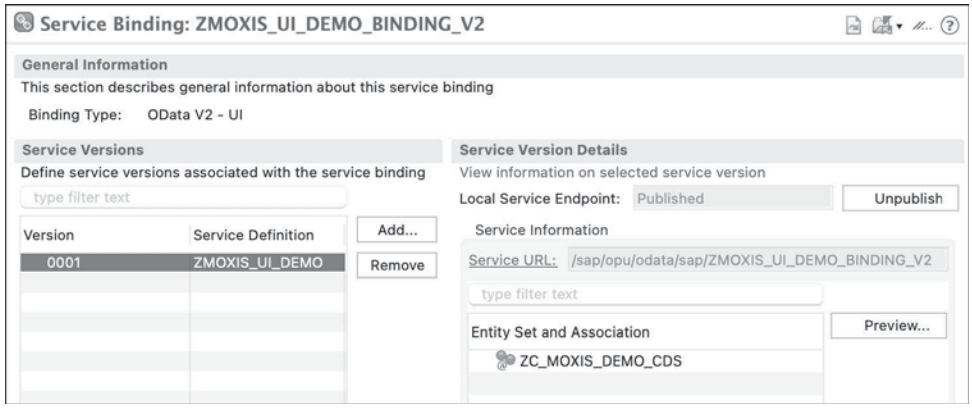

Abbildung 4.12: Service-Binding in den ABAP Development Tools erstellen

Service-konsumierung

Die bereitgestellten OData-Services werden entweder als *UI-Services* für die Konsumierung in SAP-Fiori-Applikationen oder als *Web-APIs* für die Konsumierung in jedem beliebigen OData-Client bereitgestellt. Jede Frontend-Konfiguration, die in Form von Annotationen aus dem UI-Vokabular im Backend durchgeführt wird, ist in den Metadaten des OData-Service enthalten.

Eine SAP-Fiori-Elements-Applikation liest die Metadaten-Informationen, interpretiert sie und erstellt daraus abgeleitet die passenden Benutzeroberflächen. Die Metadaten eines OData-Service, der als Web-API bereitgestellt wird, enthalten keine benutzeroberflächenspezifischen Informationen.

Vorschau auf die Benutzeroberfläche

Um die von Ihnen im Consumption View definierten Annotationen zu testen, können Sie bei der Erstellung des Service-Bindings in den ADT eine Vorschau im Standardbrowser Ihres Betriebssystems starten. Klicken Sie dazu auf den Button **Preview**. Die Vorschau für das in diesem Abschnitt vorgestellte Beispiel sehen Sie in Abbildung 4.13.

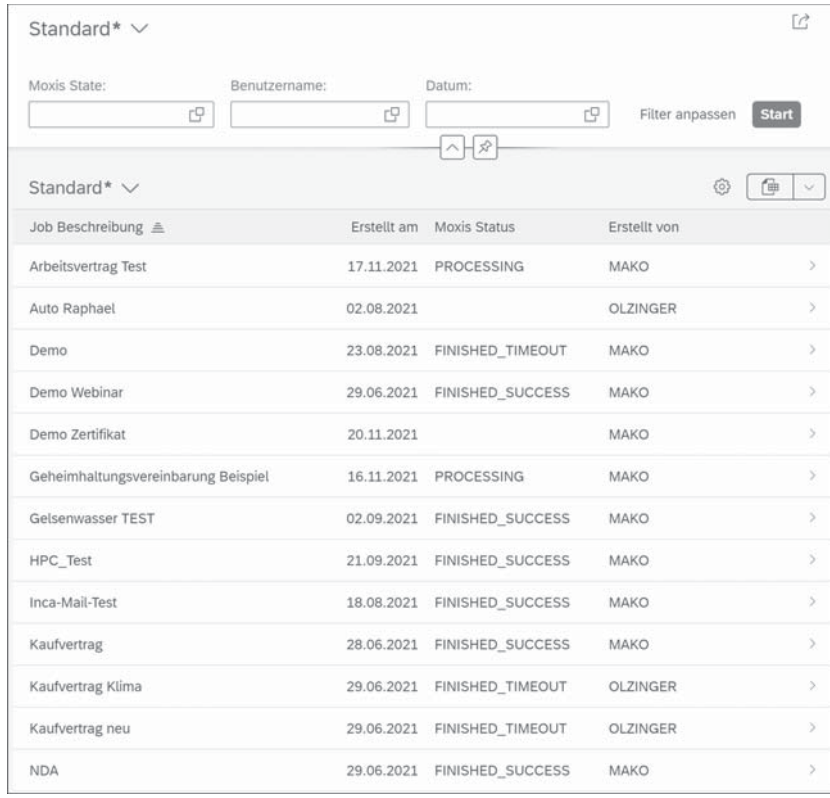

Abbildung 4.13: Vorschau auf die Benutzeroberfläche der SAP-Fiori-Elements-Applikation in den ABAP Development Tools

Wie Sie erkennen können, gibt es sehr viele Parallelen zwischen dem SAP Cloud Application Programming Model und dem ABAP RESTful Application Programming Model. Das macht den Umstieg von einem Modell auf das andere oder die Arbeit mit beiden Modellen für Entwickler*innen sehr einfach. Wir haben in diesem Kapitel lediglich einen Überblick über die Funktionsweise der beiden Programmiermodelle gegeben, da der Fokus dieses Buches auf der Konsumierung der Services in SAP Fiori Elements liegt. Für die Entwicklung mit dem SAP Cloud Application Programming Model sollten Sie sich mit JavaScript oder Java und für die Entwicklung mit dem ABAP RESTful Application Programming Model mit ABAP auskennen, wenn Sie komplexere Anforderungen umsetzen möchten.