



Application Development with SAP® Business Technology Platform

- Build and deploy cloud applications on SAP BTP using Java, Node.js, Python, and ABAP
- Work with SAP Business Application Studio, SAP Cloud Application Programming Model, SAP Joule for Developers, and more
- Manage, monitor, and secure your applications

2nd edition, updated and expanded

Bajaj • Bhattacharya • Bhowmik
Dhar • Ghosh • Acharya • Lahiri



Rheinwerk
Publishing

Contents

Foreword by Dr. Tom Janoshalmi	19
Preface	21

PART I Getting Started

1 Introduction to SAP Business Technology Platform	29
1.1 Transformation Flywheel	30
1.2 Cloud Platforms	31
1.3 Pillars of SAP Business Technology Platform	33
1.3.1 Agent and Application Development	34
1.3.2 Integration	36
1.3.3 Artificial Intelligence and Data Foundation	37
1.3.4 Deployment Models	39
1.4 SAP Business AI Platform	39
1.5 Summary	40
2 Development Environment	41
2.1 Introduction	41
2.1.1 SAP Business Technology Platform Architecture	43
2.1.2 Development Environments	45
2.2 SAP BTP, Cloud Foundry Environment	46
2.2.1 Characteristics of Cloud Foundry	47
2.2.2 How SAP Business Technology Platform Has Adopted Cloud Foundry	49
2.2.3 Cloud Foundry Services Offered by SAP Business Technology Platform	49
2.2.4 SAP Joule for Developers	52
2.3 SAP BTP ABAP Environment	53
2.3.1 Multitenancy in SAP BTP ABAP Environment	55
2.3.2 Key Cloud-Specific Features	55
2.3.3 SAP Joule for Developers	56

2.4	SAP BTP, Kyma Runtime	57
2.4.1	Overview	57
2.4.2	Creating Services	58
2.4.3	Creating Functions	59
2.5	Deployment Based on Development Environment	60
2.5.1	Deployment on SAP BTP, Neo Environment	60
2.5.2	Deployment on SAP BTP, Cloud Foundry Environment	62
2.5.3	Deployment on SAP BTP, Kyma Runtime	64
2.5.4	Deployment on SAP BTP ABAP Environment	64
2.6	Use Cases Based on Development Environment	65
2.6.1	Application Development Use Cases Supported by SAP BTP, Neo Environment	65
2.6.2	Application Development Use Cases Supported by SAP BTP, Cloud Foundry Environment	66
2.6.3	Application Development Use Cases Supported by SAP BTP ABAP Environment	66
2.6.4	Application Development Use Cases Supported by SAP BTP, Kyma Runtime	67
2.7	Estimating the Cost of SAP BTP Projects	67
2.8	Summary	71

PART II Backend and Frontend Cloud Application Development

3	Designing Cloud Applications for SAP	75
3.1	Cloud-Native Applications	76
3.1.1	What Are Cloud-Native Applications?	77
3.1.2	Microservice Architectures	77
3.1.3	Container Architectures	79
3.2	Clean Core for SAP BTP	80
3.2.1	Concept	81
3.2.2	Mechanism	82
3.2.3	Benefits	84
3.3	Microservices	84
3.3.1	What Are Microservices?	85
3.3.2	How Does SAP Business Technology Platform Enable Microservices?	86
3.3.3	Finding the Right Microservice	91

3.3.4	Consuming Microservices	92
3.3.5	Microservices in SAP BTP, Kyma Runtime	93
3.4	Integrating Cloud-Native Applications	93
3.4.1	Introduction to SAP Integration Suite	94
3.4.2	SAP Data Intelligence	108
3.4.3	SAP Business Accelerator Hub	110
3.4.4	Migrating SAP Process Integration or SAP Process Orchestration Content to SAP Integration Suite	110
3.4.5	Scenario-Based Recommendations for Integration Tools	112
3.5	Summary	114
4	SAP Business Application Studio	115
4.1	Capabilities	116
4.2	Setting Up SAP Business Application Studio	117
4.2.1	Plans	117
4.2.2	Setup Instructions	118
4.2.3	Types of Dev Spaces	118
4.2.4	Creating Dev Spaces in SAP Business Application Studio	135
4.2.5	Workspaces Inside Dev Spaces	140
4.3	Developing Your First Cloud Application Using SAP Business Application Studio	142
4.3.1	Creating a Project Using the Wizard	142
4.3.2	Creating a Project Using the Terminal	143
4.3.3	Creating a Project in a Local Workspace by Cloning a Repository	144
4.3.4	Creating a Project by Importing a Project	149
4.4	Leveraging SAP Build Code and Joule	150
4.4.1	What Is SAP Build Code?	150
4.4.2	Set Up SAP Build Code on SAP BTP	150
4.4.3	Create a Project Using SAP Build Code	153
4.5	Migrating Projects from SAP Web IDE to SAP Business Application Studio	158
4.6	Summary	161
5	SAP HANA Cloud	163
5.1	Features of SAP HANA Cloud	163
5.2	Medallion Data Architecture	165
5.3	SAP HANA Cloud Central	167

- 5.4 SAP HANA Cloud Knowledge Graph Engine 168**
 - 5.4.1 Knowledge Graphs 168
 - 5.4.2 Features and Capabilities of SAP HANA Cloud Knowledge Graph Engine 169
- 5.5 SAP HANA Cloud Vector Engine 171**
 - 5.5.1 Journey from Unstructured Data to Vectors 172
 - 5.5.2 Benefits and Use Cases of SAP HANA Cloud Vector Engine 174
- 5.6 Summary 174**
- 6 SAP Cloud Application Programming Model 175**
 - 6.1 Overview of SAP Cloud Application Programming Model 175**
 - 6.2 Project Setup 176**
 - 6.3 Understanding Domain Modeling 178**
 - 6.4 Using the Default SQLite Database 183**
 - 6.5 Using the SAP HANA Cloud Database 186**
 - 6.6 Generating SAP Fiori User Interfaces 189**
 - 6.7 Deploying Applications to SAP BTP, Cloud Foundry Environment 193**
 - 6.8 Consuming External Services 195**
 - 6.9 Additional Topics 197**
 - 6.9.1 Localization of SAP Fiori Apps 198
 - 6.9.2 Authentication and Authorization 199
 - 6.9.3 Annotations 205
 - 6.9.4 SAP Cloud Application Programming Model LLM Plugin 206
 - 6.10 Summary 207**
- 7 Developing Node.js Applications 209**
 - 7.1 Creating a Project Using SAP Business Application Studio 209**
 - 7.2 Connecting to SAP BTP, Cloud Foundry Environment 213**
 - 7.3 Installing Node.js Modules 216**
 - 7.4 Creating Core Data Services Resources for Our Sample Project 218**
 - 7.4.1 Creating Files for Domain Models 218
 - 7.4.2 Creating Files for Service Definitions 219
 - 7.4.3 Creating Data Files 220
 - 7.5 Creating SAP HANA Cloud Artifacts 221**
 - 7.5.1 Binding Resources to the HDI Container Service 222
 - 7.5.2 Deploying Artifacts in SAP Business Application Studio 223
 - 7.5.3 Viewing Database Artifacts Using SAP HANA Database Explorer 224

7.6	Deploying Applications to SAP BTP, Cloud Foundry	225
7.7	Generating Artifacts Using Joule	228
7.8	Summary	233
8	Developing Java Applications	235
8.1	Creating a Project from a Template	236
8.2	Creating a Java Application	239
8.3	Creating a Service Instance and Binding It to an Application	244
8.3.1	Creating an SAP HANA Deployment Infrastructure Container Service Instance	245
8.3.2	Deploying Artifacts in a Backend Database in SAP Business Application Studio	246
8.3.3	Viewing Database Artifacts Using the SAP HANA Database Explorer	246
8.4	Creating Core Data Services Domain Models and Services Based on SAP HANA Artifacts	247
8.4.1	Creating a Calculation View	247
8.4.2	Creating a Domain Model Based on a Calculation View	249
8.4.3	Creating a Service Definition	250
8.5	Deploying Applications to SAP BTP, Cloud Foundry	250
8.6	Summary	254
9	Developing ABAP Applications	255
9.1	SAP BTP ABAP Environment Architecture and Overview	255
9.2	Provisioning an SAP BTP ABAP Environment Trial	257
9.3	ABAP Development Tools for Eclipse	261
9.3.1	Installation and Setup	261
9.3.2	SAP Joule for Developers	265
9.3.3	ABAP AI Capabilities	266
9.4	ABAP RESTful Application Programming Model	268
9.4.1	Data Modeling and Behavior	269
9.4.2	Business Service Provisioning	270
9.4.3	Service Consumption	271
9.5	Developing a List Report App	271
9.5.1	Business Requirements	271
9.5.2	Creating an ABAP Package and Database Table	272
9.5.3	Building an ABAP RESTful Application Programming Model Application	276

9.6	Managed Versus Unmanaged Transactional Apps	286
9.7	Developing a Managed App	287
9.8	Adding Draft Capabilities to Managed Apps	294
9.9	Developing an Unmanaged App	298
9.10	Developing a Web API	310
9.11	Transport Management	312
9.12	Summary	318
10	Developing Python Application Using SAP Business Application Studio	319
10.1	Python Overview and Libraries	319
10.2	Jupyter Notebook	321
10.3	Creating a Python Application	323
10.3.1	Setup	323
10.3.2	Create a Project	328
10.3.3	Install Python Extension	330
10.3.4	Install Flask Library	332
10.3.5	Deployment	336
10.4	Summary	341
11	Developing Applications in SAP Business Technology Platform, Kyma Runtime	343
11.1	SAP BTP, Kyma Runtime	344
11.2	Use Cases	346
11.3	Building with Kyma	347
11.3.1	Prerequisite Configuration	347
11.3.2	Configuring SAP BTP, Kyma Runtime, and Service Deployment	349
11.3.3	Creating Pairing and Application Programming Interface Rules	357
11.4	Summary	358
12	Developing SAP Fiori Applications	359
12.1	Overview and Architecture	360
12.2	SAP Fiori Tools in SAP Business Application Studio	361
12.3	Freestyle SAP Fiori versus SAP Fiori Elements	364

12.4	Developing Complex Freestyle SAP Fiori Applications	366
12.4.1	Developing List-Detail Applications	367
12.4.2	Basic Files in SAPUI5 Applications	375
12.5	Developing Complex SAP Fiori Elements Applications	375
12.5.1	Determining Floorplans	376
12.5.2	Determining Data Sources and Other Details	378
12.5.3	Guided Development	379
12.6	Deploying an Application	386
12.7	Extending Freestyle Applications	387
12.8	Extending SAP Fiori Elements Applications	388
12.9	Summary	390

PART III Operating Applications

13	DevOps Implementation	393
13.1	DevOps Fundamentals	394
13.1.1	Core Tenets of DevOps	394
13.1.2	DevOps Inner Circles	398
13.1.3	Continuous Integration/Continuous Delivery	399
13.2	Implementing Tooling for DevOps	401
13.2.1	Continuous Everything	401
13.2.2	Requirements Management	403
13.2.3	Collaborative Development Tools	407
13.2.4	Deployment Options in the Cloud	409
13.2.5	Quality Assurance	414
13.2.6	Monitoring and Feedback	417
13.2.7	Continuous Delivery Options in SAP BTP	418
13.3	Best Practices for DevOps in SAP	423
13.4	DevOps Tooling Integration for SAP and Hybrid Landscapes	425
13.5	Summary	428
14	Security	431
14.1	SAP BTP Authentication Options	431
14.1.1	Identity Authentication	432
14.1.2	Solution Overview for User Authentication via the SAP BTP User Store	434

14.1.3	Solution Overview for User Authentication Using a Corporate User Store	435
14.1.4	Solution Overview for User Authentication Using Your Own Identity Provider	436
14.2	SAP BTP Single Sign-On	437
14.2.1	Application-to-Application Single Sign-On	437
14.2.2	Assertion SSO	438
14.2.3	Principal Propagation	438
14.3	Access Management on SAP BTP	439
14.3.1	Key Concepts in User Authentication in SAP BTP	439
14.3.2	User Account and Authentication Service	440
14.3.3	Web Access	440
14.4	Authorizations for Applications on SAP BTP, Cloud Foundry Environment	442
14.4.1	Access Control	442
14.4.2	Security Artifacts for Authorizations	443
14.4.3	Authorization Enforcement	443
14.4.4	Scopes, Role Templates, and Attributes	444
14.5	Web Security for Applications in SAP BTP	444
14.5.1	Protection Against Cross-Origin Resource Sharing	445
14.5.2	Protecting Against Cross-Site Request Forgery	445
14.5.3	Protecting Against Click-Jacking Attempts	445
14.5.4	Protecting Against Cross-Site Scripting	446
14.5.5	Setting Up Timeouts for Application User Sessions	446
14.5.6	Securing Node.js Applications	446
14.5.7	Securing Your Java Applications	446
14.6	Joule Security	447
14.7	Summary	448
15	Application Monitoring	449
15.1	Monitoring Java Applications	449
15.2	Monitoring Database Systems	452
15.3	Monitoring Cloud Foundry Applications	455
15.3.1	Activating Application Log	456
15.3.2	Visualizing Application Log	456
15.4	Monitoring APIs	458
15.4.1	REST APIs for Java Applications	458
15.4.2	REST APIs for Database Systems and SAP HANA Extended Application Services	461

15.5 Monitoring Applications with Notifications 461

15.6 SAP Automation Pilot 462

 15.6.1 Concepts 462

 15.6.2 How SAP Automation Pilot Works 464

15.7 Summary 465

PART IV Supporting Tools and Services

16 SAP Workflow Management 469

16.1 Overview 470

 16.1.1 Key Capabilities 470

 16.1.2 Limitations and Considerations 471

 16.1.3 Process Model 471

 16.1.4 User Interfaces 472

 16.1.5 Configuring Workflow Steps 473

 16.1.6 Triggering the Workflow 474

 16.1.7 Configuring the Rules 475

 16.1.8 Process Visibility 477

16.2 Integrations 478

 16.2.1 Integration Specific to Workflow Steps 479

 16.2.2 Integrating Intelligent Services 479

16.3 Model Architecture for Workflow on SAP Business Technology Platform 480

16.4 Workflow Package 481

16.5 Workflow Monitoring 482

16.6 Summary 483

17 SAP Build Work Zone 485

17.1 Introduction to SAP Build Work Zone 486

17.2 Key Concepts 486

17.3 SAP Build Work Zone Editor 488

 17.3.1 Spaces, Pages, Cards, Widgets, and Content Packages 489

 17.3.2 Users, Roles, and Permissions 491

 17.3.3 Content Integration 492

17.4 Summary 494

18	SAP Build Process Automation	495
18.1	Basics of SAP Build Process Automation Design	495
18.1.1	Tenant Setup for SAP Build Lobby	496
18.1.2	Projects	500
18.1.3	Agents	505
18.1.4	Recorder	506
18.1.5	Packages	507
18.1.6	Deployment	508
18.2	Orchestrating Automation Processes	508
18.3	Attended and Scheduled Bots	510
18.4	Imbuing Intelligence in Bots	510
18.5	Integration	512
18.6	Summary	514
19	Joule	517
19.1	Introduction to Joule	517
19.2	Capabilities	519
19.3	Key Process and Components	523
19.4	Joule Extensions and Agents	524
19.5	Summary	528
20	SAP Business Data Cloud	529
20.1	Architecture	529
20.2	Capabilities	531
20.3	SAP Business Data Cloud Cockpit	533
20.4	Data Products	534
20.5	Intelligent Applications	536
20.6	SAP Databricks	538
20.7	Use Cases	540
20.7.1	Modernizing SAP Business Warehouse with SAP Business Data Cloud	540
20.7.2	Supporting SAP Business AI	542
20.8	Summary	543

21 Conclusion and Roadmap 545

21.1 The Autonomous Enterprise 545

21.2 Cloud Platforms 547

21.3 Digital Innovation 550

21.4 SAP BTP Roadmap 552

21.5 Summary 553

The Authors 555

Index 559

Chapter 6

SAP Cloud Application Programming Model

SAP Cloud Application Programming Model is an application development approach for creating cloud applications. The model provides developers with tools, languages, and libraries that can help reduce development overhead by simplifying the development process. This chapter will introduce you to the model and its use in detail.

In this chapter, we'll explain the concepts behind SAP Cloud Application Programming Model and its unique capabilities. We'll describe the technical components available in SAP Cloud Application Programming Model and then go deeper into various implementation concepts through simple examples. At the end of this chapter, you should have a fair understanding of this application development approach and its related steps for building enterprise-grade applications. Let's begin our journey by understanding the concepts behind SAP Cloud Application Programming Model.

6.1 Overview of SAP Cloud Application Programming Model

A *programming model* is one level higher than programming languages and describes best practices when building applications in a standardized way. A programming model comprises of one or more programming languages, tools, libraries, templates, and more. A well-defined programming model will help you develop your applications quickly and help you test, debug, and troubleshoot. SAP Cloud Application Programming Model guides the developer down a golden path to developing applications based on best practices through the use of out-of-the-box content. The key differentiator of SAP Cloud Application Programming Model is its *domain models* that focus on domains instead of technical disciplines. Before we describe domain models in detail in subsequent sections, let's first explore the various technical components of SAP Cloud Application Programming Model, as shown in Figure 6.1.

As shown in Figure 6.1, SAP Cloud Application Programming Model is based on many open-source technologies (e.g., Java, Node.js, etc.). However, it also contains many SAP specific technologies, such as core data services (CDS), SAP Fiori, SAP HANA, and more. CDS is the definition language for building domain models and for defining services. SAP Fiori is the

set of design principles for achieving a better user experience (UX). SAP HANA is the database where all database artifacts are created behind the scenes. You can also use any other UX technology and other databases as well; however, building SAP Cloud Application Programming Model-based applications always starts with domain modeling using CDS.

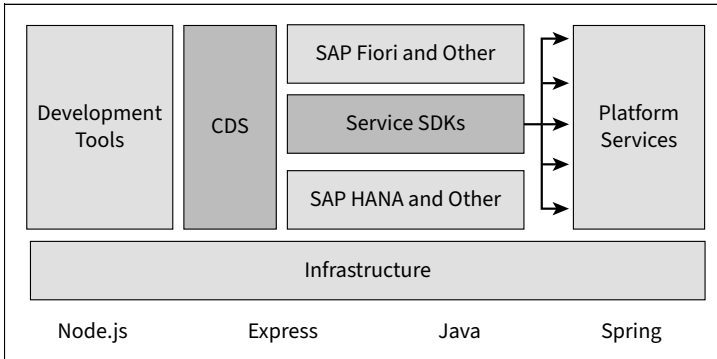


Figure 6.1: SAP Cloud Application Programming Model: Technical Components

SAP Cloud Application Programming Model provides two runtime choices (Java and Node.js), and its service software development kits (SDKs) provide various platform services to handle repetitive tasks in a platform-agnostic way. As mentioned earlier, SAP Cloud Application Programming Model also contains best practices out of the box. You also have multiple options when selecting development tools as it has dedicated tools support for SAP Business Application Studio, Visual Studio Code (VS Code), and Eclipse. All behavioral aspects of SAP Cloud Application Programming Model are based on services and events. Services are stateless, declared using CDS, and can be accessed by other services using application programming interfaces (APIs). Services react to various events.

Now, you have some basic ideas about SAP Cloud Application Programming Model and its technical components. Before we dive into further details and build an application, we'll shift our focus to development tools. Even though multiple tool choices are available, we'll primarily focus on SAP Business Application Studio. Since Chapter 4 already covered the setup of SAP Business Application Studio and creating your project, we'll only cover a few SAP Cloud Application Programming Model-specific pieces of project setup in the next section.

6.2 Project Setup

As soon as your project is created, it's visible in SAP Business Application Studio, as shown in Figure 6.2.

You can now open the project workspace by clicking on the project's name. Once the workspace is open, you'll find three primary folders (**app**, **db**, and **srv**), as shown in Figure 6.3.

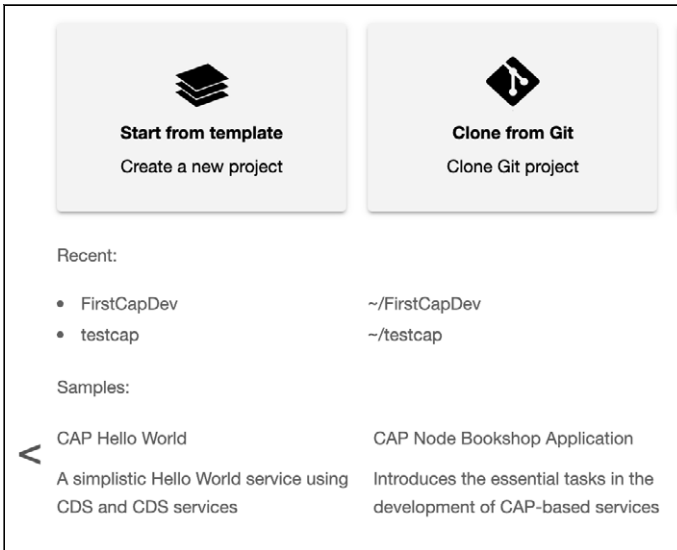


Figure 6.2: SAP Cloud Application Programming Model Project Available in SAP Business Application Studio

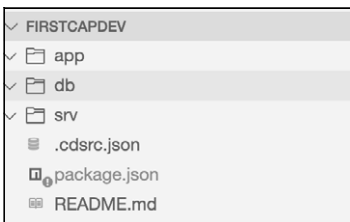


Figure 6.3: Folder Structure of an SAP Cloud Application Programming Model Project

You user interface (UI) content should be stored in the **app** folder; the domain model and database-related artifacts should go to the **db** folder, and the service definition and implementation should be maintained in the **srv** folder. The *package.json* file is your project's description in JavaScript Object Notation (JSON). We'll explore the significance and usage of all these folders and files in the following sections.

While you're in your project workspace, you can open the terminal window by following the menu path **Terminal • New terminal**. You should see the current folder is your project name, as shown in Figure 6.4.

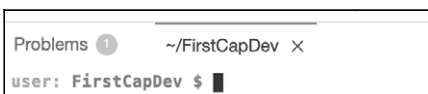


Figure 6.4: Terminal Showing Project Name as the Current Folder

Now, you can enter the command `cds env get folders` to review your current folder structure, as shown in Figure 6.5. This folder structure can be changed via configuration in the `package.json` file, but we'll keep it simple and go with the default folder structure. Keeping this default project layout will also help in the easy execution of other `cds` commands.

```
user: FirstCapDev $ cds env get folders
{ db: 'db/', srv: 'srv/', app: 'app/' }
```

Figure 6.5: Command to View the Folders

Our project has now been created with the basic structure, and further code can be added under various folders. Although mentioned earlier, let's now explain domain modeling in greater detail since this primary concept will be used throughout the rest of this chapter.

6.3 Understanding Domain Modeling

A domain model is a key element when building SAP Cloud Application Programming Model-based applications, where the focus is on the domain level, defined in a business-centric way, rather than defining the finer details of a technical implementation. The domain model is the starting point when building any application using SAP Cloud Application Programming Model. As shown in Figure 6.6, database artifacts are created from a domain model either directly or through a persistence model. Domain models can also be exposed as services for UIs. Domain models can also be reused by other SAP Cloud Application Programming Model projects.

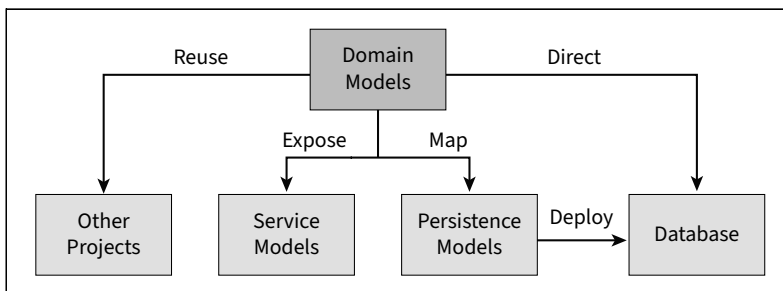


Figure 6.6: Central Role of Domain Model

The purpose of domain modeling is to enhance collaboration between developers and domain experts by providing a shared language. Technically, a domain model is like an entity relationship model, with many more additional features.

Let's now create our first domain model and add various features to it. Your domain models must be created in your defined `db` folder. Open SAP Business Application Studio and open your previously created project workspace, right-click on the project workspace, and select **New File** from the context menu. Create a file with the name "schema.cds" under the `db` folder, as shown in Figure 6.7.

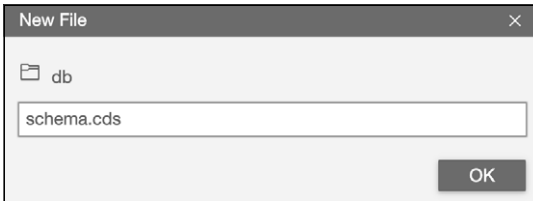


Figure 6.7: Creating a `schema.cds` File to Define the Domain Model

Once the file is created, you can add the code for the domain model. Listing 6.1 shows some basic CDS code that describes one entity for a sales order with a bunch of attributes.

```
namespace company.orders;

entity Header {
  key ordernum : Integer;
  customer: String(10);
  salesorg: String(4);
  distchan: String(2);
  division: String(2);
}
```

Listing 6.1: Initial Code for Building a Header Entity

Listing 6.1 shows just one entity with the name `Header`, which is the order's header information with five different attributes. The key for this entity is `ordernum`. The namespace is optional, but normally, you would add a namespace directive on top of your domain model (e.g., the entity `Header` can be referred to as `company.orders.Header` outside of your domain model).

Let's now try to understand entity association with the enhanced source code shown in Listing 6.2.

```
namespace company.orders;

entity Header {
  key ordernum : Integer;
  itemno: Association to many Items;
  customer: Association to Customers on customer.customerID =
CustomerID;
  salesorg: String(4);
  distchan: String(2);
  division: String(2);
  CustomerID: String;
```

```

}

entity Customers {
    key customerID: String(10);
    name: String(40);
    parent: Association to Header;
}

entity Items {
    itemno: Integer;
    product: Association to Products;
    qty: Integer;
    price: Decimal(15,2);
    parent: Association to Header;
}

entity Products {
    productno: String(40);
    name: String(50);
}

```

Listing 6.2: Entities with Associations

Associations define the relationships between entities. As shown in Listing 6.2, the `Header` entity is associated with the entities `Items` and `Customers`. Similarly, the `Items` entity is further associated with products and so on. When you define a service on top of this model, the service can generate complex join conditions based on these defined associations to fulfill any query operation. These associations can also be used to create or update complex hierarchical data. If you carefully look into the nature of the associations for the `Header` entities, you'll find two different patterns of association: one with the `on` clause and another without the `on` clause. Any association with the `on` clause is called an *unmanaged association* because you can use any arbitrary elements to join two entities. *Managed associations*, on the other hand, always use the primary key of the associated entity and thus don't need the `on` condition. Additionally, as shown in Listing 6.2, some associations have one-to-many relationships. Maintaining many-to-many relationship is also possible but is not shown in Listing 6.2.

After defining the domain model, let's define a service to expose this model as an OData service. Our service should ideally be created in the `srv` folder, as shown in Figure 6.8.

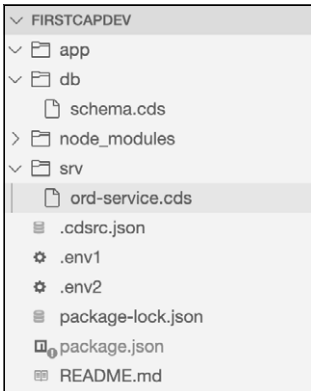


Figure 6.8: Service File *ord-service.cds*

Listing 6.3 shows the source code of the *ord-service.cds* file, where we've exposed all four entities as a part of the OData service.

```
using { company.orders as ord } from '../db/schema';
```

```
service CatalogService {

    entity Orders as SELECT from ord.Header {*,
        customer.name as name
    } ;
    entity Orderitems as select from ord.Items{*};
    entity Customers as select from ord.Customers;
    entity Products as select from ord.Products;

}
```

Listing 6.3: Service Exposure of the Domain Model

We'll now open a new terminal window and run the command `cds watch`, as shown in Figure 6.9.

```
[cds] - model loaded from 2 file(s):
./db/schema.cds
./srv/ord-service.cds

[cds] - connect using bindings from: { registry: '~/cds-services.json' }
[cds] - connect to db > sqlite { database: ':memory:' }
/> successfully deployed to sqlite in-memory db

[cds] - serving CatalogService { at: '/catalog' }

[cds] - server listening on { url: 'http://localhost:4004' }
[cds] - launched at 4/12/2022, 4:21:42 AM, in: 2.336s
[cds] - [ terminate with ^C ]
```

Figure 6.9: *cds watch*

As shown in Figure 6.9, by default, SAP Cloud Application Programming Model connects to an in-memory SQLite database to create all the necessary database artifacts. In later sections, you'll learn how to switch this connection to an actual SQLite database and an SAP HANA database separately. However, let's now explore the exposed service. The `cds watch` command will display the popup window shown in Figure 6.10.

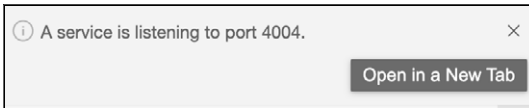


Figure 6.10: Popup Window for Testing Your Service for the Domain Model

Clicking on **Open in a New Tab** will open a new browser tab where you can see your service details, as shown in Figure 6.11.

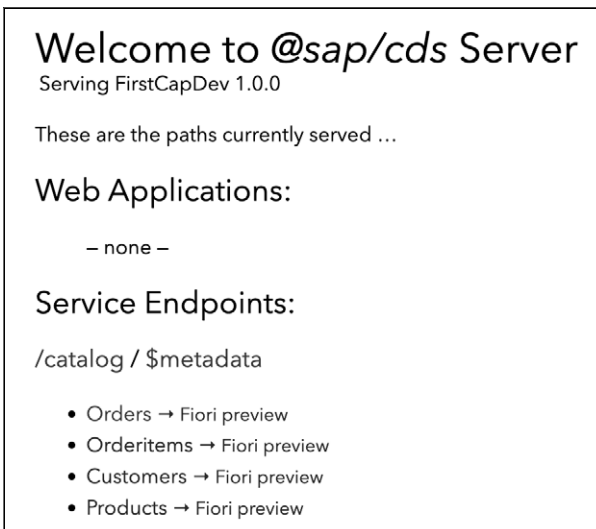


Figure 6.11: Exposing Service and Entity Set Details

Clicking on **\$metadata** will open the OData metadata that you can explore further to understand your service, its entity sets, its associations, and its elements as defined by your domain model and as exposed by your service definition CDS. Figure 6.12 shows a glimpse of the metadata file, in the Entity Data Model in XML format (EDMX) file format.

Now, you've created a domain model with various elements, key attributes, and associations and have exposed this domain model as an OData service. As mentioned earlier, domain models serve as the primary design element for building a persistence model (i.e., database artifacts) and your service model. In the following sections, you'll learn how to add different databases (e.g., SQLite and SAP HANA) to a project so that database artifacts can be created persistently in the database. So, let's start with using SQLite as a database as the persistence medium for our project.

```

▼<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx" Version="4.0">
  ▼<edmx:DataServices>
    ▼<Schema xmlns="http://docs.oasis-open.org/odata/ns/edm" Namespace="CatalogService">
      ▼<EntityContainer Name="EntityContainer">
        ▼<EntitySet Name="Orders" EntityType="CatalogService.Orders">
          <NavigationPropertyBinding Path="itemno" Target="Orderitems"/>
          <NavigationPropertyBinding Path="customer" Target="Customers"/>
        </EntitySet>
        ▼<EntitySet Name="Orderitems" EntityType="CatalogService.Orderitems">
          <NavigationPropertyBinding Path="product" Target="Products"/>
          <NavigationPropertyBinding Path="parent" Target="Orders"/>
        </EntitySet>
        ▼<EntitySet Name="Customers" EntityType="CatalogService.Customers">
          <NavigationPropertyBinding Path="parent" Target="Orders"/>
        </EntitySet>
        <EntitySet Name="Products" EntityType="CatalogService.Products"/>
      </EntityContainer>
      ▼<EntityType Name="Orders">
        ▼<Key>
          <PropertyRef Name="ordernum"/>
        </Key>
        <Property Name="ordernum" Type="Edm.Int32" Nullable="false"/>
        <NavigationProperty Name="itemno" Type="Collection(CatalogService.Orderitems)"/>
        ▼<NavigationProperty Name="customer" Type="CatalogService.Customers">
          <ReferentialConstraint Property="CustomerID" ReferencedProperty="customerID"/>
        </NavigationProperty>
        <Property Name="salesorg" Type="Edm.String" MaxLength="4"/>
      </EntityType>
    </Schema>
  </DataServices>
</Edmx>

```

Figure 6.12: OData Metadata in an EDM File

6.4 Using the Default SQLite Database

Earlier, you saw how an SAP Cloud Application Programming Model project by default provides an SQLite in-memory database if you do not want to persist the data while restarting the service. Thus, every time you load the service or a related app, the data will be reset to its initial state. This feature is a good way to check and verify your initial service or application without connecting to a database. If you want this database connection to change to an SQLite database, run the command `cds deploy --to sqlite`, as shown in Figure 6.13.

```

user: FirstCapDev $ cds deploy --to sqlite
/> successfully deployed to ./sqlite.db

/> updated ./package.json

```

Figure 6.13: Deploying CDS Artifacts to an SQLite Database

As shown in Figure 6.13, the CDS command also updated the `package.json` file. Your project's `package.json` should now contain the code shown in Listing 6.4.

```

"cds": {
  "requires": {
    "db": {
      "kind": "sqlite",
      "credentials": {
        "database": "sqlite.db"
      }
    }
  }
}

```

```

    }
  }
}

```

Listing 6.4: package.json Content for Deployment to the SQLite Database

You can now run the command `cds deploy --to sqlite: memory`, and you'll see your `package.json` file has been updated, as shown in Listing 6.5.

```

"cds": {
  "requires": {
    "db": {
      "kind": "sqlite",
      "credentials": {
        "database": "memory"
      }
    }
  }
}

```

Listing 6.5: package.json Content for Deployment to In-Memory SQLite

So, now you understand how to select an SQLite in-memory database using the `cds` command. You may also need some default data for your database so that you can quickly test your application. SAP Cloud Application Programming Model provides an easy way to update your database from CSV files. The first step is to create a folder called **data** under the **db** folder, as shown in Figure 6.14.

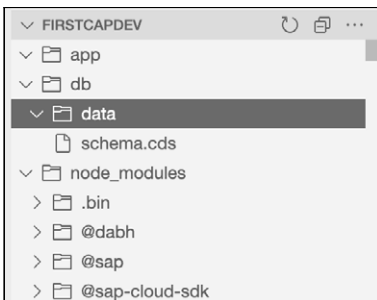
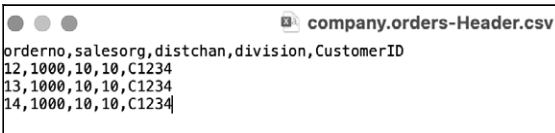


Figure 6.14: Test Data Folder Under the db Folder

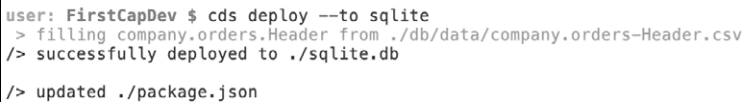
Create a CSV file with the correct element name as the column header, and your test data in the rows shown in Figure 6.15. Then, drop the CSV file into the data folder. The name of this file must be in the format `<namespace>-<Entity Name>.csv`.



```
orderno,salesorg,distchan,division,CustomerID
12,1000,10,10,C1234
13,1000,10,10,C1234
14,1000,10,10,C1234
```

Figure 6.15: Sample Test Data Content for the Header Entity

Once your file is loaded, run the command `cds deploy --to sqlite` again to check for the successful loading of the data, as shown in Figure 6.16. In our case, we deployed the data for just one entity, but you can create other CSV files for other entities and load them together.

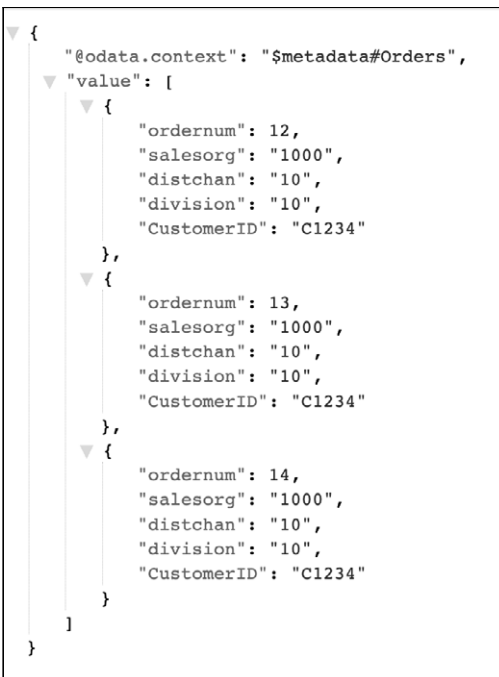


```
user: FirstCapDev $ cds deploy --to sqlite
> filling company.orders.Header from ./db/data/company.orders-Header.csv
/> successfully deployed to ./sqlite.db

/> updated ./package.json
```

Figure 6.16: Deploying an SQLite Database with Test Data

Now, use the `cds watch` command again and then open the `Orders` entity to review its test data, as shown in Figure 6.17.



```
{
  "@odata.context": "$metadata#Orders",
  "value": [
    {
      "ordernum": 12,
      "salesorg": "1000",
      "distchan": "10",
      "division": "10",
      "CustomerID": "C1234"
    },
    {
      "ordernum": 13,
      "salesorg": "1000",
      "distchan": "10",
      "division": "10",
      "CustomerID": "C1234"
    },
    {
      "ordernum": 14,
      "salesorg": "1000",
      "distchan": "10",
      "division": "10",
      "CustomerID": "C1234"
    }
  ]
}
```

Figure 6.17: Test Data for the Order Entityset of Service

You should now be comfortable adding SQLite databases to your applications and loading test data. Test data can also be loaded in a similar way for an in-memory SQLite database.

However, we generally use an SAP HANA database as the persistence layer for enterprise-grade applications, and thus, you might be curious about switching this SQLite database for an SAP HANA database. Let's tackle that scenario next!

6.5 Using the SAP HANA Cloud Database

Since the option of adding an SQLite database is provided by default with any SAP Cloud Application Programming Model project, getting set up doesn't require much effort. But remember that the SQLite database is part of your project and can be seen in the project structure. The database will be deployed to SAP BTP, Cloud Foundry environment together with the other artifacts. If you want to develop an enterprise-grade application, however, you should use an SAP HANA Cloud database provisioned in SAP BTP, Cloud Foundry environment. Chapter 5 already covered SAP HANA Cloud in some detail, so we'll only walk through the steps specific to using SAP HANA as your database for your SAP Cloud Application Programming Model application here.

Before you can add SAP HANA as the database to your application, you'll need to log on to SAP BTP, Cloud Foundry environment, via SAP Business Application Studio by following the menu path **View • Find Command**. Once logged on to SAP BTP, Cloud Foundry environment, click the **Cloud Foundry** link in the toolbar on the left in SAP Business Application Studio, which should display details about your SAP BTP, Cloud Foundry environment, along with your SAP HANA Cloud instance, as shown in Figure 6.18.

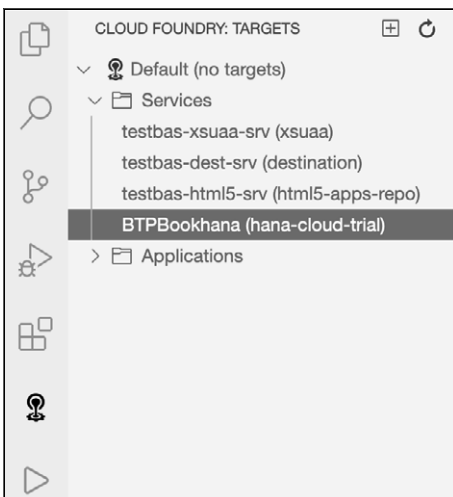


Figure 6.18: SAP HANA Cloud Instance in SAP BTP, Cloud Foundry Environment from SAP Business Application Studio

Now, go back to the project explorer and open the terminal window again. If you created your project previously using the terminal window, you may be lacking the required *multi-target application (MTA)*-specific features in your project. In this case, use the command

`cds add mta`. This step is not required if you created a SAP Cloud Application Programming Model project using the wizard and selected the **MTA Based SAP Business Technology Platform Deployment** option. After optionally adding *mta* features, you can now run the command `cds add hana` in your terminal window, which will perform two primary changes to your *package.json* file:

- The command will add an `hdb` driver for SAP HANA as a dependency.
- A data source of type `sql` will be added to the `cds.requires.db` block.

Use the command `cds deploy --to hana` to deploy the database artifacts to your SAP HANA Cloud instance, as shown in Figure 6.19.

```
user: FirstCapDev $ cds deploy --to hana
[cds.deploy] - Starting deploy to SAP HANA ...

[cds.deploy] - Creating build tasks
[cds.deploy] - Running build

[cds.deploy] - Using container FirstCapDev-db

[cds.deploy] - Running 'cf create-service hana hdi-shared FirstCapDev-db' with
Creating service instance FirstCapDev-db in org 69fcc01ctrial / space dev as ga
OK

Create in progress. Use 'cf services' or 'cf service FirstCapDev-db' to check
Showing info of service FirstCapDev-db in org 69fcc01ctrial / space dev as ga
```

Figure 6.19: Deploying the Project's Database Artifacts to SAP HANA

Deployment triggers the creation of an instance of **SAP HANA Schemas & HDI Containers**, as shown in Figure 6.20. Service keys will be created, and then all the SAP HANA database tables/views in the SAP HANA deployment infrastructure (HDI) container will be deployed.

Instance	Service	Plan
BTPBookhana	SAP HANA Cloud	hana
FirstCapDev-db	SAP HANA Schemas & HDI Containers	hdi-shared

Figure 6.20: SAP HANA Deployment Infrastructure Schemas in SAP BTP, Cloud Foundry Environment

Note

In our example, we added an extra record to our test data file(s) before deploying to SAP HANA, as shown in Figure 6.21.

```
db > data > company.orders-Header.csv
1  ordernum,salesorg,distchan,division,CustomerID
2  12,1000,10,10,C1234
3  13,1000,10,10,C1234
4  14,1000,10,10,C1234
5  15,1000,10,10,C1234
```

Figure 6.21: Extra Record in the Test Data File Before Its Deployment to SAP HANA



If you want to run your application from SAP Business Application Studio but need to see data from your SAP HANA database, you must change the `kind` property of the `package.json` file to `hana`, as shown in Figure 6.22, and then run `cds watch`.

```

    },
    "cds": {
      "requires": {
        "db": {
          "kind": "hana",
          "credentials": {
            "database": "sqlite.db"
          }
        }
      }
    }
  }
}

```

Figure 6.22: Changing the `kind` Property to `hana`

Now, when viewing your order data, you'll see the extra record is now available because the data is coming from the SAP HANA database, as shown in Figure 6.23.

```

{
  "@odata.context": "$metadata#Orders",
  "value": [
    {
      "ordernum": 12,
      "salesorg": "1000",
      "distchan": "10",
      "division": "10",
      "CustomerID": "C1234"
    },
    {
      "ordernum": 13,
      "salesorg": "1000",
      "distchan": "10",
      "division": "10",
      "CustomerID": "C1234"
    },
    {
      "ordernum": 14,
      "salesorg": "1000",
      "distchan": "10",
      "division": "10",
      "CustomerID": "C1234"
    },
    {
      "ordernum": 15,
      "salesorg": "1000",
      "distchan": "10",
      "division": "10",
      "CustomerID": "C1234"
    }
  ]
}

```

Figure 6.23: Additional Record Coming from the SAP HANA Database

Switching the `kind` property to `sql` will show only three records, which means your SQLite database has three records, but SAP HANA has four records for the `Header` entity (i.e., `Order Entityset`) of the OData service.

6.6 Generating SAP Fiori User Interfaces

SAP Cloud Application Programming Model provides out-of-the-box support for SAP Fiori elements applications, driven by annotations. Figure 6.24 shows links to several SAP Fiori elements applications, listed under **Service Endpoints**.

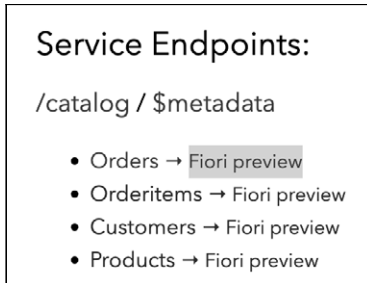


Figure 6.24: SAP Fiori Preview

These links are for previews of possible SAP Fiori applications, as shown in Figure 6.25.

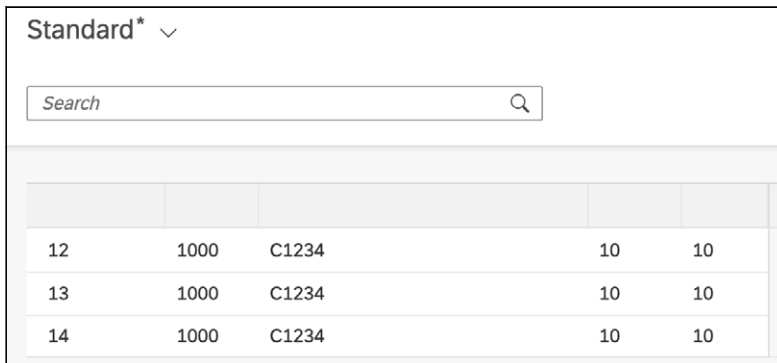


Figure 6.25: SAP Fiori App in Preview Mode

To build a full-fledged SAP Fiori app, you'll need to add annotations for your domain model or your service model. Annotations are additional metadata that you can use to instruct the SAPUI5 framework how the UI should look. Based on your defined annotations at runtime, the SAPUI5 framework will derive the necessary UIs for you. Thus, through annotations, you can create low-code/no-code SAP Fiori applications. We recommend adding separate annotation files to the **app** folder, but you can technically put annotations anywhere in a model.

SAP Business Application Studio provides another powerful feature, SAP Fiori tools, for building SAP Fiori applications on top of your SAP Cloud Application Programming Model service. To create an SAP Fiori app, follow the menu path **File • New Project From Template** and then select **SAP Fiori Application**. On the next screen, select **SAP Fiori Elements** and **List report object page**. Then, you'll select an SAP Cloud Application Programming Model project by its project path and select a service, as shown in Figure 6.26. Once you complete

these steps in the project, your SAP Fiori-related files should be visible in the **app** folder, as shown in Figure 6.27.

Figure 6.26: SAP Cloud Application Programming Model Project Details

Figure 6.27 shows a separate annotation file added to the **app • orders** folder.

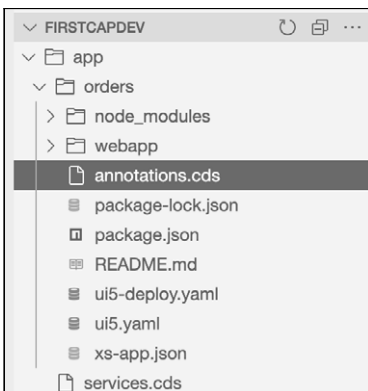


Figure 6.27: Annotation File in the app/orders Folder

Some annotation features include the following:

- Annotations start with @; for instance, @UI.LineItem is for table fields.
- You can annotate service entities, entity elements, and more.
- Annotations support internationalization (i.e., i18n for supporting multiple languages).
- Through annotations, you can define the draft functionality for editable entities.

SAP Business Application Studio supports code completion for annotations so that you can create annotations more quickly. Listing 6.6 shows a generated annotation file with several different types of annotations created.

```
using CatalogService as service from '../..srv/ord-service';
annotate service.Orders with @(
  UI.LineItem : [
    {
      $Type : 'UI.DataField',
```

```

        Label : 'ordernum',
        Value : ordernum,
    },
    {
        $Type : 'UI.DataField',
        Label : 'salesorg',
        Value : salesorg,
    },
    {
        $Type : 'UI.DataField',
        Label : 'distchan',
        Value : distchan,
    },
    {
        $Type : 'UI.DataField',
        Label : 'division',
        Value : division,
    },
    {
        $Type : 'UI.DataField',
        Label : 'CustomerID',
        Value : CustomerID,
    },
    ],
);
annotate service.Orders with @(
    UI.FieldGroup #GeneratedGroup1 : {
        $Type : 'UI.FieldGroupType',
        Data : [
            {
                $Type : 'UI.DataField',
                Label : 'ordernum',
                Value : ordernum,
            },
            {
                $Type : 'UI.DataField',
                Label : 'salesorg',
                Value : salesorg,
            },
            {
                $Type : 'UI.DataField',
                Label : 'distchan',
                Value : distchan,
            },
        ],
    },
);

```

```

        {
            $Type : 'UI.DataField',
            Label : 'division',
            Value : division,
        },
        {
            $Type : 'UI.DataField',
            Label : 'CustomerID',
            Value : CustomerID,
        },
    ],
},
UI.Facets : [
    {
        $Type : 'UI.ReferenceFacet',
        ID : 'GeneratedFacet1',
        Label : 'General Information',
        Target : '@UI.FieldGroup#GeneratedGroup1',
    },
]
);

```

Listing 6.6: Annotations for SAP Fiori Apps

The `UI.LineItem` annotation is responsible for creating table fields in the list report, while `UI.FieldGroups` is responsible for displaying fields on the object page, and `UI.Facets` is responsible for organizing object page fields. Many other different UI annotations are available, which you can explore at cap.cloud.sap. If you again enter the `cds watch` command, you'll see that your test screen has been changed to display the newly created app, as shown in Figure 6.28.

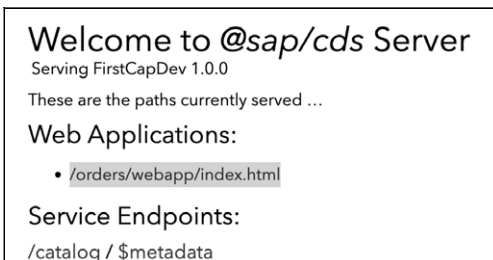


Figure 6.28: Test Screen Updated to Test the Newly Created SAP Fiori App

At this point, our SAP HANA database is available, the domain model is complete, a service has been exposed, and a UI will be generated through SAP Fiori elements. Now, everything can be tested via SAP Business Application Studio, but we'll deploy our application to SAP BTP, Cloud Foundry environment.

6.7 Deploying Applications to SAP BTP, Cloud Foundry Environment

An SAP Cloud Application Programming Model application can be deployed in various ways, depending on your use case, for instance, in the following ways:

- Deploy to SAP BTP, Cloud Foundry environment: For simple use cases.
- Deploy to SAP BTP, Cloud Foundry environment as a multitenant software as a service (SaaS) application: For applications where a cloud service provider will deploy the app, and multiple customers will use the app with their own data.
- Deploy to the continuous integration (CI)/continuous delivery (CD) pipeline: For deployment automation.
- Simple SAP BTP, Cloud Foundry environment deployment: Using Cloud Foundry's native `cf push` command.

In this book, we'll focus on the first option, deploying to SAP BTP, Cloud Foundry environment. First, however, let's explore some key tools essential for deployment:

- Cloud MTA Build Tool: This tool creates the `.mtar` file, which is an MTA archive file, from the deployment descriptor (i.e., `mtad.yaml`) of the SAP Cloud Application Programming Model application.
- The Cloud Foundry CLI plugin
- The MTA plugin for the Cloud Foundry CLI plugin: This tool is used to deploy the `.mtar` file created by the Cloud MTA Build Tool to SAP BTP, Cloud Foundry environment.

With these tools in place, to execute the deployment process, follow these steps:

1. If you created the project using the wizard in SAP Business Application Studio, then the Cloud MTA Build Tool and the Cloud Foundry CLI with the MTA plugin have already been installed for your project. Run the commands `mbt` and `cf` to ensure these tools are available in your project.
2. If you used an SQLite database for testing and now you want to switch to an SAP HANA database for deployment, run the command `cds add hana --for production`. In this context, `production` means the production landscape that has the SAP HANA database as a persistence.
3. Now, add extended services for user account and authentication (XSUAA)-based authentication using the command `cds add xsuaa --for production`. You'll learn more about XSUAA authentication in Section 6.9.2, but for now, simply run the command.
4. Add the *AppRouter* as a gateway. You can make your app available to users in two ways after deployment: using the AppRouter as the entry point or using the SAP Fiori launchpad. We'll use the AppRouter option in this scenario, so we'll run the command `cds add approuter --for production`.

5. If you previously used the `cds add mta` command, then your project folder already has the required deployment descriptor; otherwise, run this command to add the *mta.yaml* file to your project structure.
6. You should now freeze dependencies using the command `npm update --package-lock-only`. Freezing dependencies will create a file called *package-lock.json*. This file may also be present in your project hierarchy depending on how you created the project or how you added some other tools.
7. Use the Cloud MTA Build Tool to build and assemble everything into an *mta.tar* archive. Use the command `mbt build -t gen --mta mta.tar` to create the *mta.tar* file under the **gen** folder.
8. Make sure your SAP HANA database instance is up and running, which is particularly important if you're using a trial account because this instance is stopped automatically when not used.
9. The final step is to deploy the *mta.tar* file to SAP BTP, Cloud Foundry environment. You can now run the command `cf deploy gen/mta.tar` to deploy the *mta.tar* file. Make sure you are logged on to SAP BTP, Cloud Foundry environment, using the command **CF:Login to Cloud Foundry** via the menu path **View • Find Command** in SAP Business Application Studio prior to running this command.

Once you complete the deployment process, your app should be visible under the **Routes** menu of the SAP BTP, Cloud Foundry environment space you selected before deployment, as shown in Figure 6.29.

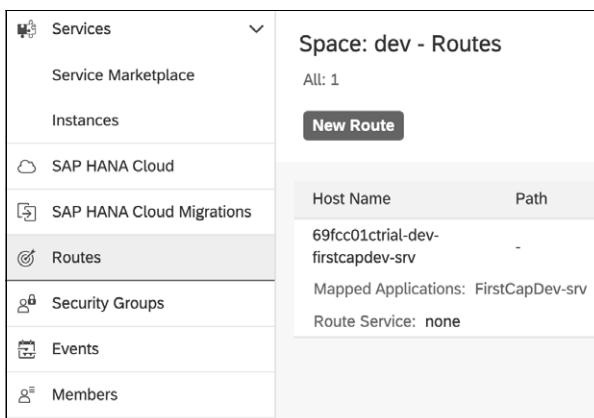


Figure 6.29: Deployed App in an SAP BTP, Cloud Foundry Environment Space

At this point, you've seen how a complete full-stack SAP Cloud Application Programming Model application, with all its database artifacts, service components, and SAP Fiori UIs, can be built and deployed to SAP BTP, Cloud Foundry environment. However, you may also encounter a situation where you need to consume external services for your application (e.g., consuming SAP S/4HANA services to build a side-by-side application for your core ERP system). We'll cover the steps for consuming external services next.

6.8 Consuming External Services

So far, we've described the steps for building an application from scratch and running its database artifacts either on local SQLite database or on SAP BTP/SAP HANA Cloud. However, in many situations, you may need to consume external services from one or more on-premise and/or cloud-based systems, for instance, from SAP S/4HANA, SAP S/4HANA Cloud, SAP Sales Cloud, SAP SuccessFactors, and others. Your requirement might be a simple consumption step or might be complex enough to extend a service with additional fields for calculation, aggregation, etc. In this section, we'll walk you through the steps for consuming an external service, starting from design to deployment, as shown in Figure 6.30.

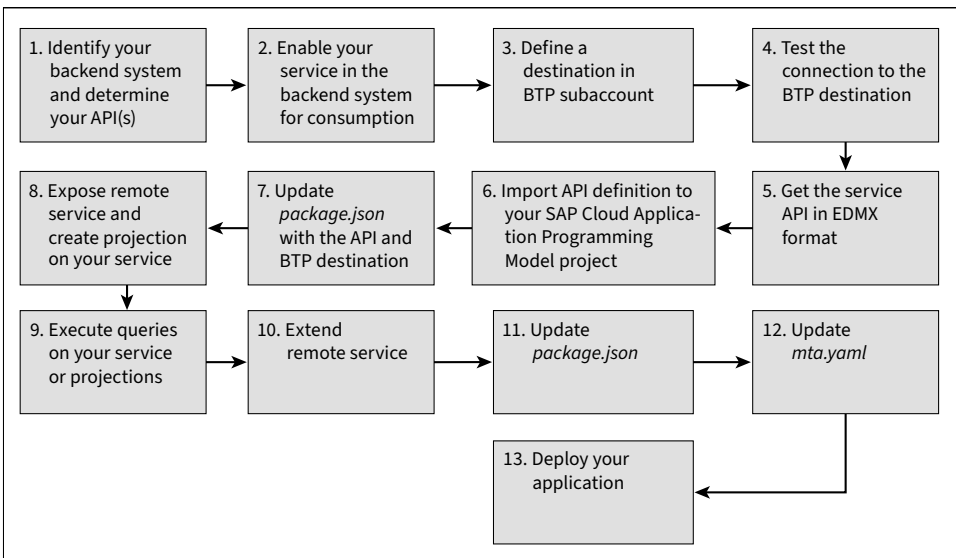


Figure 6.30: Steps for Consuming External Services

Let's briefly go over each step next:

1. The first step is simple: identify the related backend system(s) that will provide the required services. Note that your backend service can be from an on-premise SAP system (e.g., SAP S/4HANA); an on-premise non-SAP system; an SAP cloud system like SAP Ariba or SAP Sales Cloud; or even a non-SAP cloud system like Salesforce. Depending on the type of system, your configuration will be slightly different. For example, you can use the cloud connector to consume any services from SAP S/4HANA on-premise.
2. The next step is to identify one or more APIs and related services from the backend system(s). You may need to consume different services from multiple systems and mash up the data in your SAP Cloud Application Programming Model application. If your backend system is SAP S/4HANA, then you can explore the available APIs from SAP Business Accelerator Hub at <https://api.sap.com>.

3. Enable this service in your backend system. For example, you must define a communication arrangement for that specific communication scenario for SAP S/4HANA Cloud, and for SAP S/4HANA, you'll need to activate the OData service (OData API).
4. In the next step, define an SAP BTP destination in your predefined subaccount to connect to the previously enabled backend system APIs.
5. Test the connection to the SAP BTP destination and make sure it's reachable.
6. Get the API's service definition in EDMX format. For SAP APIs, you can export the EDMX file from SAP Business Accelerator Hub. SAP Cloud Application Programming Model also works with OData v4 export.
7. Import the API service definition using the `cds import` command.
8. You can now update the `package.json` file of your SAP Cloud Application Programming Model project, as shown in Listing 6.7.

```

    "cds": {
      "requires": {
        "API_BUSINESS_PARTNER": {
          "kind": "odata",
          "model": "srv/external/API_BUSINESS_PARTNER",
          "[production]": {
            "credentials": {
              "destination": "S4HANA",
              "path": "/sap/opu/odata/sap/API_BUSINESS_PARTNER"
            }
          }
        }
      }
    }
  }
}

```

Listing 6.7: Updating the package.json File with External Service Details

As shown in Listing 6.7, you can use destination for production testing after deployment or use the local testing option without the destination.

9. Next, you'll expose, extend, and project your service, as shown in Listing 6.8, which is similar to exposing a local service.

```

using { API_BUSINESS_PARTNER as bupa } from '../srv/external/API_
BUSINESS_PARTNER';

extend service RiskService with {
  entity BusinessPartners as projection on bupa.A_BusinessPartner;
}

```

Listing 6.8: Exposing, Extending, and Projecting Your Service

10. You can now use Node.js or Java code to query your external service for calculating additional fields and logic.
11. Update your *mta.yaml* file to add details under the `resources` section, as shown in Listing 6.9.

```
- name: cpapp-uaa
  type: org.cloudfoundry.managed-service
  parameters:
    service: xsuaa
    service-plan: application
    path: ./xs-security.json

- name: cpapp-destination
  type: org.cloudfoundry.managed-service
  parameters:
    service: destination
    service-plan: lite
```

Listing 6.9: Resources in the mta.yaml File

12. If you're connecting to an on-premise system, you must add a connectivity service as well, as shown in Listing 6.10.

```
- name: cpapp-connectivity
  type: org.cloudfoundry.managed-service
  parameters:
    service: connectivity
    service-plan: lite
```

Listing 6.10: Connectivity Service for On-Premise Systems

13. Now, you can deploy your application in the usual way and test your remote service or additional query logic.

We've now covered both the provisioning for and the service consumption of an SAP Cloud Application Programming Model application. Before we wrap up this chapter, we'd like to explore four additional topics that are critical for developing enterprise-grade SAP Cloud Application Programming Model applications—localization, authentication and authorization, annotations, and the SAP Cloud Application Programming Model LLM plugin for building SAP Fiori applications.

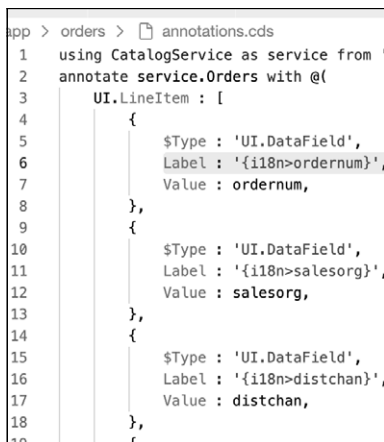
6.9 Additional Topics

We're almost at the end of this chapter explaining how to build SAP Cloud Application Programming Model applications and running them on SAP BTP, Cloud Foundry environment. To enhance this chapter, we'll go further, starting with localization and showing

you how to make an app available in multiple languages. Then, we'll describe XSUAA-based authentication, followed by discussion about some common annotations required for building fully functional SAP Cloud Application Programming Model applications. Last, but not least, we'll discuss the SAP Cloud Application Programming Model LLM plugin that lets you create advanced SAP Cloud Application Programming Model applications powered by AI/ML services.

6.9.1 Localization of SAP Fiori Apps

The localization of an app is nothing but the translation of various texts in your application into target languages. In the localization process, you must indicate which texts need to be translated. This step can be achieved by removing hard-coded texts and externalizing your texts as i18n texts, as shown in Figure 6.31.



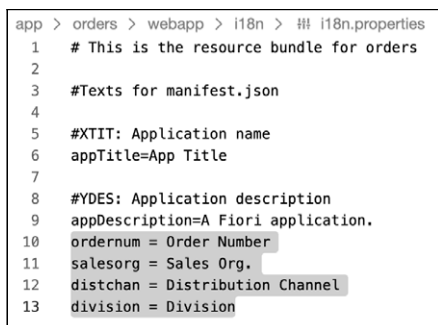
```

app > orders > annotations.cds
1  using CatalogService as service from '
2  annotate service.Orders with @(
3    UI.LineItem : [
4      {
5        $Type : 'UI.DataField',
6        Label : '{i18n>ordernum}',
7        Value : ordernum,
8      },
9      {
10       $Type : 'UI.DataField',
11       Label : '{i18n>salesorg}',
12       Value : salesorg,
13     },
14     {
15       $Type : 'UI.DataField',
16       Label : '{i18n>distchan}',
17       Value : distchan,
18     },
19   ]
20 )

```

Figure 6.31: i18n Texts for Translations

As shown, these texts are indicated by the `i18n>` prefix in curly braces, and each word next to that prefix serves as a key to uniquely identify that text. Next, you'll create i18n files to contain the translations for each key. A typical i18n file is shown in Figure 6.32, with default language (English) texts.



```

app > orders > webapp > i18n > ## i18n.properties
1  # This is the resource bundle for orders
2
3  #Texts for manifest.json
4
5  #XTIT: Application name
6  appTitle=App Title
7
8  #YDES: Application description
9  appDescription=A Fiori application.
10 ordernum = Order Number
11 salesorg = Sales Org.
12 distchan = Distribution Channel
13 division = Division

```

Figure 6.32: i18n File Content (Default)

When you created the SAP Fiori elements module for your SAP Cloud Application Programming Model project, a default `i18n` file might have already been created with the name `i18n.properties`, in the `app/<app name>/webapp/i18n` folder. You can edit this file further to add additional texts, as shown in Figure 6.32.

Other options for storing `i18n` files include in the `_i18n` folder or in the `_assets/i18n_` folder. Using the property `cds.i18n.folders` in the `package.json` file, you can change the folder path as needed. In this way, you can create multiple files for different languages, as shown in Figure 6.33.

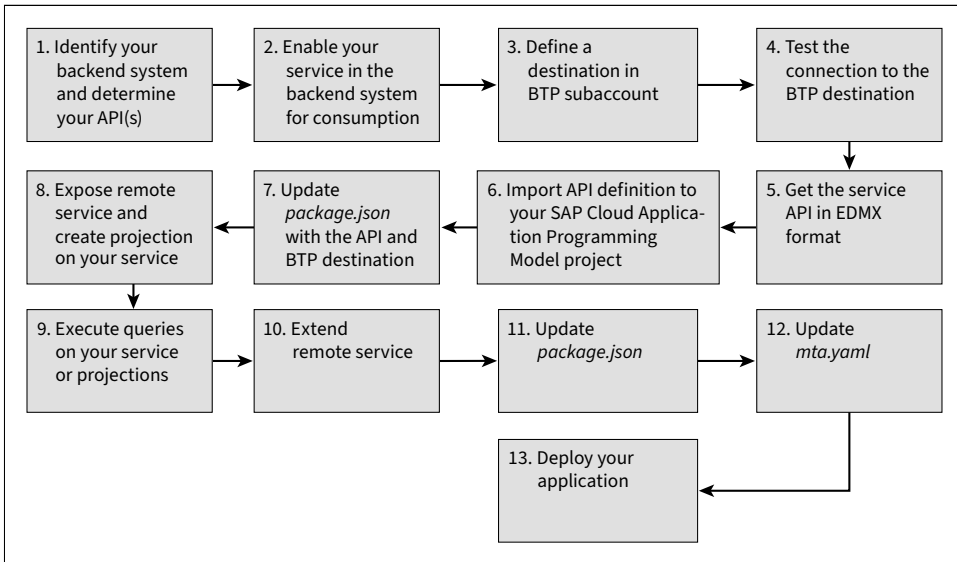


Figure 6.33: `i18n` Files for Different Languages

Figure 6.33 shows three variations of `i18n` files: The file without any language specification will be applied first and then language-specific `i18n` files will be applied depending on the user's preferred language. The user's language is determined in multiple ways (e.g., browser settings, `sap-language` URL parameter, HTTP request header's `accept-language` parameter, default language configured at app level, and so on).

We hope you have a good sense of how the localization concept can be applied for your SAP Cloud Application Programming Model projects. Let's now shift our focus to another area—XSUAA-based authentication and authorization.

6.9.2 Authentication and Authorization

SAP provides various authentication strategies for use with SAP Cloud Application Programming Model applications. By default, SAP Cloud Application Programming Model uses `mocked-auth` in development (in SAP Business Application Studio) and `jwt-auth` in production (in SAP BTP). You can override these settings by specifying a different authentication strategy in the `package.json` file. Some options include the following:

- Dummy authentication: This type of authentication creates a user that passes all authorization checks.
- Basic authentication: Authentication with user ID and password that can be used for testing.
- JSON Web Token (JWT)-based authentication: Authentication is performed using tokens (JWT) in the authorization header of the incoming HTTP request.
- XSUAA-based authentication: This option is also based on JWT and additionally offers SAML attributes. This choice is commonly used for SAP Cloud Application Programming Model applications.
- Custom authentication: You can implement your own authentication using a JavaScript file and referencing that file in your *package.json* file.

In most cases, you'll probably use XSUAA-based authentication for SAP Cloud Application Programming Model development. Figure 6.34 shows the steps for implementing authentication and authorization using XSUAA to provide a high-level overview of authorization concepts.

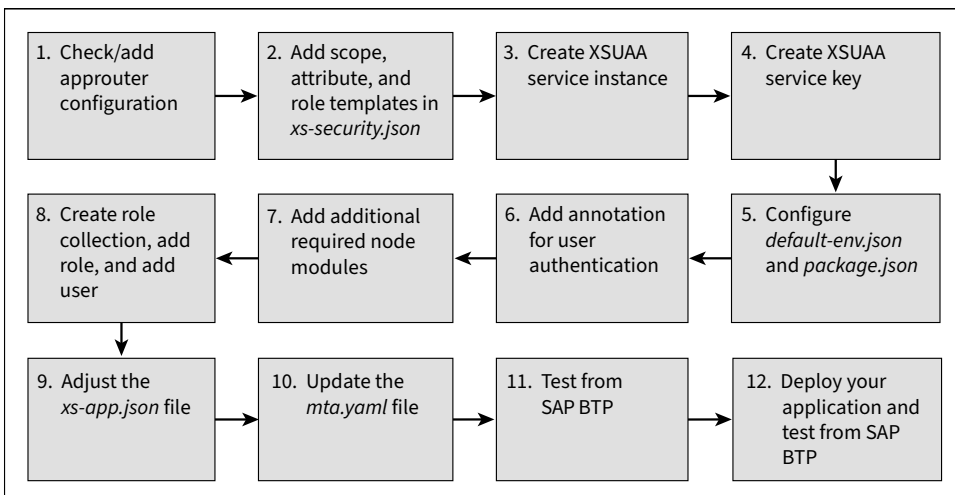


Figure 6.34: Steps for Setting Up Authentication and Authorization

Let's go over each step shown in Figure 6.34 in sequence:

1. Add the AppRouter, which is the central entry point through which a calling component can access your SAP Cloud Application Programming Model application. If you haven't added an AppRouter to your project, doing so is the first step you should perform. The AppRouter can be added either using the command `cds add approuter --for production` or using the wizard available via the menu path **View • Find Command** and selecting the option **MTA Module from Template**. Opening the wizard will display the AppRouter option, as shown in Figure 6.35.

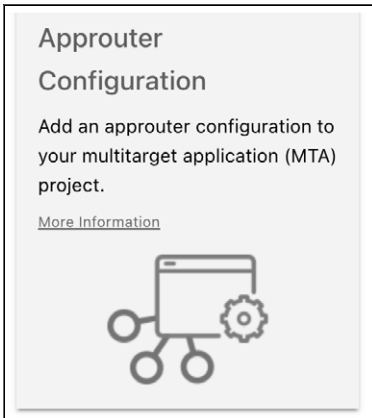


Figure 6.35: Adding an AppRouter Using a Wizard

Using the wizard will help you create the AppRouter for your SAP Cloud Application Programming Model project.

2. When you add an AppRouter in the first step, the `xs-security.json` file is created in the project's root folder. Using the wizard will also add some sample content to this file, as shown in Listing 6.11.

```
{
  "xsappname": "firstcapdev",
  "tenant-mode": "dedicated",
  "description": "Security profile of called application",
  "scopes": [
    {
      "name": "uaa.user",
      "description": "UAA"
    }
  ],
  "role-templates": [
    {
      "name": "Token_Exchange",
      "description": "UAA",
      "scope-references": [
        "uaa.user"
      ]
    }
  ]
}
```

Listing 6.11: Default Content of the `xs-security.json` File

The `scopes` element enables various authorization concepts and implements them using Node.js or Java. You can update the name of the role template and its description to meet your requirements. To test your security setup in SAP Business Application Studio, before deploying the application to SAP BTP, you'll need to append the content shown in Listing 6.12 to your `xs-security.json` file. This step will allow a request from SAP Business Application Studio to be processed as a wildcarded SAP Business Application Studio URL maintained as the `redirect-uris` parameter value.

```
"oauth2-configuration": {
  "redirect-uris": [
    "https://*.applicationstudio.cloud.sap/**"
  ]
}
```

Listing 6.12: Redirect URI to Test from SAP Business Application Studio

3. In the next step, create an XSUAA service instance by using the command `cf create-service xsuaa application firstcapdev-uaa -c xs-security.json`.
4. Now, create the service key for the XSUAA service instance using the command `cf create-service-key firstcapdev-uaa firstcapdev-key`. Here, the `firstcapdev-key` is the service key of the service instance `firstcapdev-uaa`, created in the previous step. After successful creation of your service instance and service key, you can view them from your SAP BTP cockpit.
5. Next, add the content shown in Listing 6.13 to your `default-env.json` file.

```
"xsuaa": [{
  "name": "firstcapdev-uaa",
  "label": "xsuaa",
  "tags": ["xsuaa"],
  "credentials": {
    [...]
  }
}],
```

Listing 6.13: Initial Content for default-env.json

6. Then, go to the SAP BTP cockpit and open the service key details of your XSUAA service instance and copy the entire content, as shown in Figure 6.36.
7. Paste this content to the `default-env.json` file for your project in the specified path, as shown in Figure 6.37.

dev-key	
Form	JSON
<pre>{ "clientId": "sb-firstcapdev-uaa!t54983", "clientsecret": "rYYIB0lPsG8rXuTkgwtdE8eQHGo=", "url": "https://69fcc01ctrial.authentication.us10.hana.ondemand.com", "identityzone": "69fcc01ctrial", "identityzoneid": "f88f2a73-0e86-4e52-ae42-78bd84d9f2de", "tenantid": "f88f2a73-0e86-4e52-ae42-78bd84d9f2de", "tenantmode": "shared", "sburl": "https://internal-xsuaa.authentication.us10.hana.ondemand.com", "apiurl": "https://api.authentication.us10.hana.ondemand.com", "verificationkey": "-----BEGIN PUBLIC KEY -----MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAAqzN3pqcRasZ40zWFQHBMRy0ISloAq pw3m9IBIkaA8C1BdrV9jZazQGxENMHTnG0at50qzTPvYh5mf0QwZh5veHxIteanM00pDr0cdg0FUA +JqqSCqWM7vBwnHToL+GwaAIceieFQsDgJL/+JgTntQ37b0MADTCGW0zm +n75zVl9K2VccfgcM2RRWnN88Z1N33TPDJqF8T3lRf1eUhXPsPoLhiNWSH8YDkx +gW9nk8XhdhASJfqPu4/ovB1MiPi0Hzboc80/mnMUq7UwLRwwd2sLf3fgdr7lgeb6PqnojmBJ1C8eew /SrsxZ3ui/sI98ku0tAN0Fjx4QKBI04DJwIDAQAB-----END PUBLIC KEY-----", "xsappname": "firstcapdev-uaa!t54983", }</pre>	

Figure 6.36: JSON Content for the XSUAA Service Key

<pre>"VCAP_SERVICES": { "xsuaa": [{ "name": "firstcapdev-uaa", "label": "xsuaa", "tags": ["xsuaa"], "credentials": { "tenantmode": "shared", "sburl": "https://internal-xsuaa.authentication.us10.hana.ondemand.com", "subaccountid": "f88f2a73-0e86-4e52-ae42-78bd84d9f2de", "credential-type": "instance-secret", "clientId": "sb-firstcapdev-uaa!t54983", "xsappname": "firstcapdev-uaa!t54983", "clientsecret": "rYYIB0lPsG8rXuTkgwtdE8eQHGo=", "url": "https://69fcc01ctrial.authentication.us10.hana.ondemand.com", "uaadomain": "authentication.us10.hana.ondemand.com", "verificationkey": "-----BEGIN PUBLIC KEY-----MIIBIjANBgkqhkiG9w0BAQEFAA apiurl": "https://api.authentication.us10.hana.ondemand.com", "identityzone": "69fcc01ctrial", "identityzoneid": "f88f2a73-0e86-4e52-ae42-78bd84d9f2de", "tenantid": "f88f2a73-0e86-4e52-ae42-78bd84d9f2de", "zoneid": "f88f2a73-0e86-4e52-ae42-78bd84d9f2de" }] }, "hana": [{ "name": "FirstCapDev-db", "tags": [</pre>	
--	--

Figure 6.37: Final Content of the default-env.json File

8. You should also update the *package.json* file, under the *cds:requires* section, with the content shown in Listing 6.14.

```
"uaa": {
  "kind": "xsuaa"
}
```

Listing 6.14: Content for *package.json* Under *cds.requires*

9. Add *@requires: 'authenticated-user'* to your *ord-service.cds* file, as shown in Figure 6.38.

```
srv > ord-service.cds > CatalogService > ...
1  using { company.orders as ord } from '../db/schema';
2  @requires: 'authenticated-user'
3  service CatalogService {
4
5      entity Orders as SELECT from ord.Header { *
6
7      };
8      entity Orderitems as select from ord.Items{*};
9      entity Customers as select from ord.Customers;
10     entity Products as select from ord.Products;
11
12 }
```

Figure 6.38: Adding the Annotation to Authenticate the Service

10. Now, add the additional node modules to your *package.json* by running the command `npm install --save passport @sap/xssec @sap/xsenv @sap/audit-logging` from the project's root folder.
11. You can now go to the SAP BTP cockpit and create a role collection. Then, add the roles created through *xs-security.json* and assign the role collection to the required users.
12. Adjust *xs-app.json* to add the statement `"authenticationMethod": "route"`.
13. Adjust the *mta.yaml* file to bind the XSUAA module to your service module.
14. You can now test your SAP Cloud Application Programming Model application's authentication by using `cds watch`, or you can deploy the application to SAP BTP and test from there.

So, you now understand how authentication works with an SAP Cloud Application Programming Model project and how to set up authentication using SAP Business Application Studio.

We'll now briefly explain how you can add authorization concepts along with authentication for your SAP Cloud Application Programming Model application. An authorization concept can be implemented using various annotations in your service definition. Let's explore some key authorization-specific annotations next:

- `@requires` adds authenticated users or adds specific custom roles for any entities.
- `@restrict` restricts or allows certain actions (e.g., read/write access) to specific custom roles. You can add various other conditions as well.
- `@readonly` restricts certain entities as read-only for all users.

Once your role and authorization design is complete, run the command `cds compile service.cds --to xsuaa > xs-security.json`, which will update the `xs-security.json` file with additional scopes and attributes.

With this brief look at authentication and authorization, let's move on to the next topic: annotations.

6.9.3 Annotations

As discussed in Section 6.6, SAP Fiori UIs are generated automatically through annotations. You may already be familiar with the annotation concept in SAP Fiori, but let's now look at annotations in more detail.

An annotation is additional metadata that enriches a service; annotations are understood by consumers that interpret the annotations in a meaningful way. Various annotation consumers exist. For instance, the ABAP platform interprets ABAP-related annotations, the SAPUI5 framework consumes SAP Fiori UI-related annotations, and the SAP S/4HANA analytics engine consumes embedded analytics annotations, among others. SAP Cloud Application Programming Model supports annotations, but these annotations are interpreted by different runtime engines. Annotations start with an `@` symbol and can have multiple levels, (e.g., `@UI.LineItem`). In this section, we'll focus on the most common UI- and OData-related annotations. SAP Cloud Application Programming Model uses annotation vocabularies following the OASIS standard as well as some SAP-specific annotation vocabularies. Some examples of annotations in SAP-specific vocabularies include the following:

- `@Common`: This annotation is common for all types of vocabularies.
- `@Communication`: This annotation carries communication related information (e.g., contact, address, email, phone, etc.).
- `@PersonalData`: These annotations represent personal and/or sensitive data.
- `@Analytics`: These annotations represent various analytical resources (e.g., dimensions, measures, aggregations, etc.).
- `@UI`: These annotations are related to SAP Fiori UIs. Some key UI annotations include the following:
 - `HeaderInfo`: This annotation helps you define the various UI properties of the header section of an object page. This annotation can have further subannotations (e.g., `Title`, `Description`, `ImageUrl`, etc.).

- **LineItem:** Line-item annotations are responsible for defining various UI properties for the fields of a list report (e.g., `Labels` for column headings, `Criticality` to represent values in different colors, and `IconURL` if the field is an icon, etc.).
- **Identification:** Identification annotations similar to `LineItems` but responsible for defining various properties of an object page’s fields.
- **SelectionFields:** To define the selection/filter fields for list report applications.
- **SelectionVariant:** To bundle all your filter selections together, similar to ABAP’s selection-screen variant.
- **PresentationVariant:** To bundle the result of the queried data along with their UI positions, sorting criteria, and more. This annotation is similar to an SAP List Viewer (ALV) variant.
- **SelectionPresentationVariant:** If you want to bundle `SelectionVariant` and `PresentationVariant` together using a single variant name, then you need to define the `SelectionPresentationVariant`.
- **MultiLineText:** This annotation will identify the fields that can have unlimited number of characters (string) represented by multiple lines of texts on the screen.
- **Facets:** To organize the header area of an object page into various sections and sub-sections.

Refer to <http://s-prs.co/v550408> for more information on the available annotations—the possibilities are endless!

6.9.4 SAP Cloud Application Programming Model LLM Plugin

So far, we have covered the process of creating SAP Cloud Application Programming Model applications using SAP Business Application Studio. However, SAP Cloud Application Programming Model applications, in the context of today’s autonomous enterprise, can become significantly more sophisticated and advanced—especially when addressing complex AI/ML or enterprise integration scenarios. The SAP Cloud Application Programming Model LLM plugin is a major step toward enabling these applications to connect to the generative AI hub and SAP HANA Cloud vector engine to perform tasks such as anonymization, embeddings, similarity search, chat completion, and end-to-end retrieval-augmented generation (RAG) flows. The plugin lets you connect to LLMs via SAP AI Core/generative AI hub using SAP Cloud Application Programming Model configurations (for example, model settings in `.cdsrc.json`) and SAP BTP destinations. However, it remains a separate SAP Cloud Application Programming Model-side component rather than an SDK module of SAP AI Core. As depicted in Figure 6.39, the SAP Cloud Application Programming Model LLM plugin offers key features, including the following:

- **Data anonymization:** The plugin enables you to leverage SAP HANA Cloud’s advanced data anonymization techniques to seamlessly protect sensitive data using simple annotations such as `@anonymize`, as well as algorithms like k-anonymity and l-diversity, ensuring data integrity and compliance.

- **Enhanced AI-powered functionalities:** The plugin lets you integrate SAP Cloud Application Programming Model applications with SAP AI Core, facilitating the generation of vector embeddings and performing similarity searches, which are especially crucial for applications that require precise and insightful data analysis.
- **Simplified access to LLMs:** The plugin also offers streamlined access to LLMs for tasks such as chat completion, which deliver contextually appropriate AI-generated responses, enhancing user interaction.
- **Effective RAG:** The plugin optimizes the RAG process by integrating the functionalities of both SAP AI Core and the SAP HANA Cloud vector engine into a single, streamlined method. This integration enables dynamic data fetching and utilization in AI operations, significantly boosting efficiency.

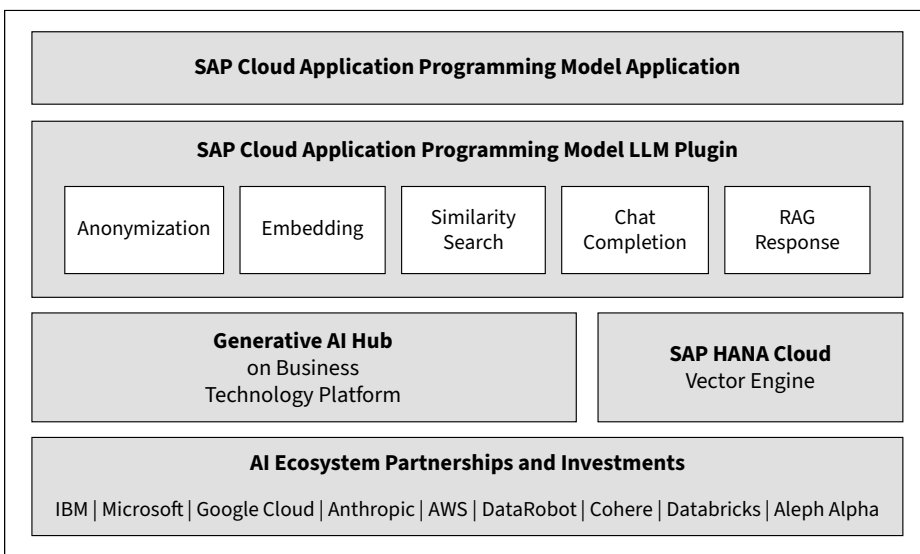


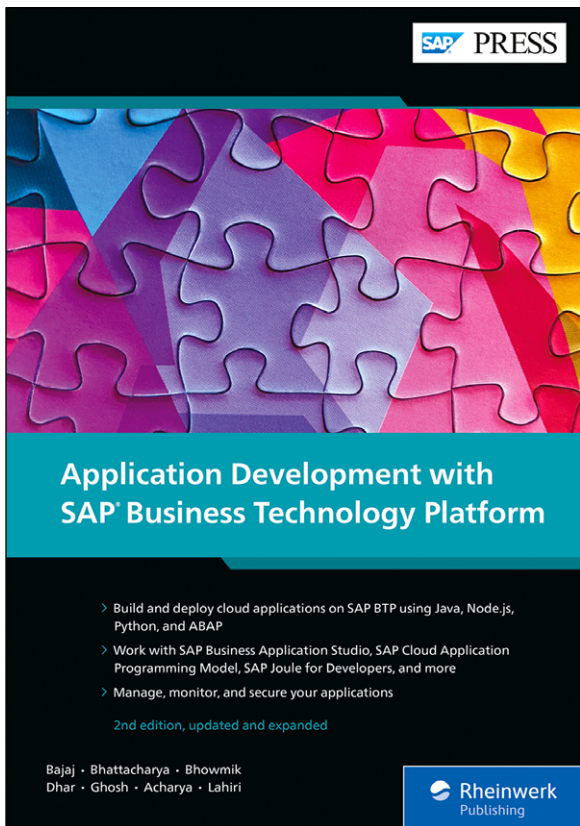
Figure 6.39: SAP Cloud Application Programming Model LLM Plugin

For more information related to the SAP Cloud Application Programming Model LLM plugin and APIs, refer to <https://github.com/SAP-samples/cap-llm-plugin-samples/blob/main/docs/api-documentation.md>.

6.10 Summary

Building a simple SAP Cloud Application Programming Model application is easy, but building an enterprise-grade one is not as simple because you'll need to handle all aspects of building a service, including service consumption from external systems, local persistence, complex UI development, authentication and authorization, and so on. SAP provides extensive ready-to-use content for faster application development, and more and more businesses are implementing this content to quickly see the benefits of using

domain modeling and wizard-driven application development features. We hope to see more customers adopt SAP Cloud Application Programming Model as SAP continuously improves its features and functionalities. Concluding this chapter on SAP Cloud Application Programming Model, let's now turn to building Node.js applications in SAP BTP.



Bajaj, Bhattacharya, Bhowmik, Dhar,
Ghosh, Acharya, Lahiri

Application Development with SAP Business Technology Platform

- Build and deploy cloud applications on SAP BTP using Java, Node.js, Python, and ABAP
- Work with SAP Business Application Studio, SAP Cloud Application Programming Model, SAP Joule for Developers, SAP Build, and more
- Manage, monitor, and secure your applications



www.sap-press.com/6322

We hope you have enjoyed this reading sample. You may recommend or pass it on to others, but only in its entirety, including all pages. This reading sample and all its parts are protected by copyright law. All usage and exploitation rights are reserved by the author and the publisher.

The Authors

Govind Bajaj is an SAP business intelligence solution architect at IBM. Arup Bhattacharya is an SAP S/4HANA transformation leader at IBM. Laboni Bhowmik is an SAP integration and automation architect at IBM. Avijit Dhar is a senior application architect at IBM India. Dr. Anup K Ghosh is a distinguished engineer at IBM Consulting. Gairik Acharya is an associate partner at IBM. Asidhara Lahiri is a managing director at Accenture.

ISBN 978-1-4932-2855-3 • 567 pages • 09/2026

E-book: \$84.99 • Print book: \$89.95 • Bundle: \$99.99



Rheinwerk
Publishing