FULL STACK DEVELOPMENT WITH SAP WEB IDE (FROM SAP HANA TO SAP FIORI UX)

CPL283

Exercises / Solutions

Elina Visoki, Rafi Pinto, Yuval Anafi,
Ran Hassid, Yotam Shapira, Shimon Tal,
Murali Shanmugham, Dipak Agarwal, Boris Tsirulnik / SAP

SAP TECHED

TABLE OF CONTENTS

1 BEFORE YOU START

Welcome to the SAP TechEd 2017 hands-on session CPL283.

This document will guide you through the development of a full-stack application for SAP Cloud Foundry using SAP Web IDE.

SAP Web IDE is an extensible, web-based integrated development tool for end-to-end application development.

By the end of this session, you will have created a full-stack application including the database, service, and UI layers. You will create a bookstore application containing a list of books with data obtained from an SAP HANA database.

All steps will be performed within SAP Web IDE and the resulting application will be deployed to the Cloud Foundry environment.

The exercise is divided into chapters, each describing a different part of our application creation.

Important: File and folder names throughout the session are **case-sensitive**. Please make sure you copy the names the way they appear in this document.

Chapter 2: Open SAP Web IDE

Estimated Duration: 10 minutes

Chapter 3: Create a Book Store Application

Estimated Duration: 90 minutes

Chapter 4: Connect the Application to GIT

Estimated Duration: 10 minutes

Chapter 5: Connect to the Cloud Foundry Environment

Estimated Duration: 10 minutes

Chapter 6: Add Create and Delete Capabilities

Estimated Duration: 60 minutes

Chapter 7: Create the Bookstore Application Using the Upcoming SAP Cloud Platform Programming Model

Estimated Duration: 50 minutes

Chapter 8: Introduction to Analytical SAP HANA Database Development

Estimated Duration: 10 minutes

2 OPEN SAP WEB IDE

Overview

Estimated time: 10 minutes

Objective

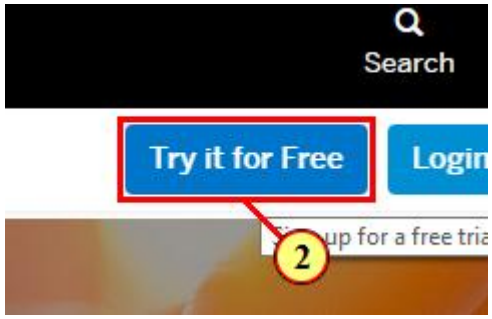
In the following exercise, you will learn how to create and log into an SAP Cloud Platform Trial account and then how to access SAP Web IDE.

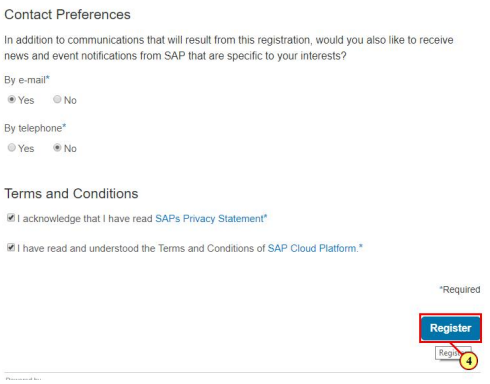
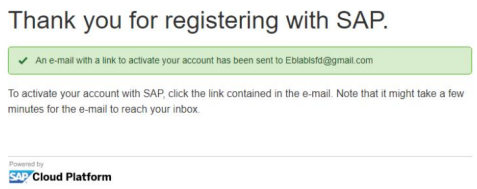
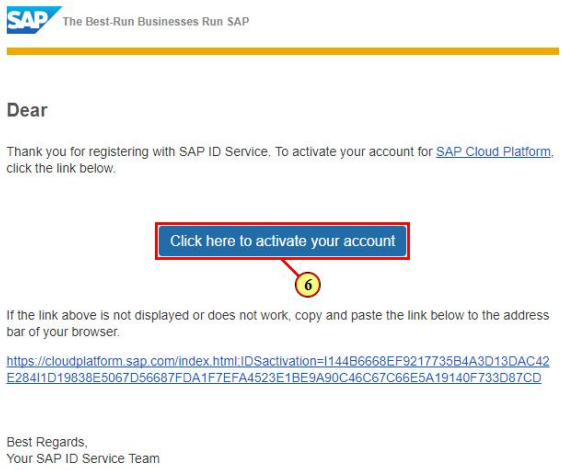
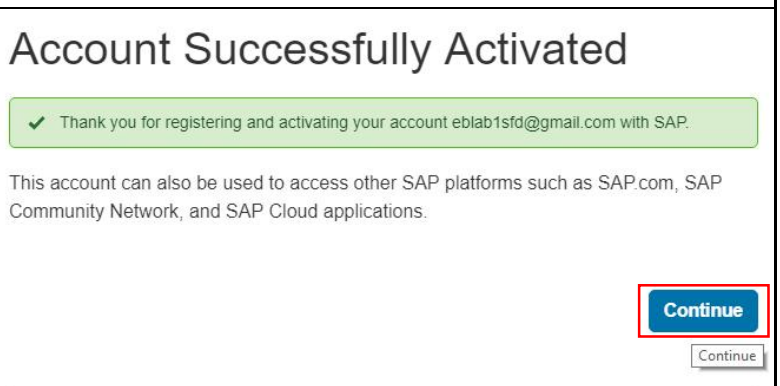
For this session, you will need to use SAP Web IDE on an SAP Cloud Platform trial account. If you already have one you can start the exercise from step 2.2

Exercise Description

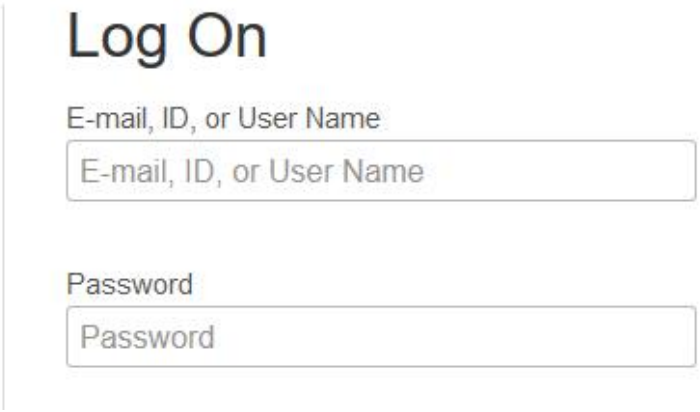
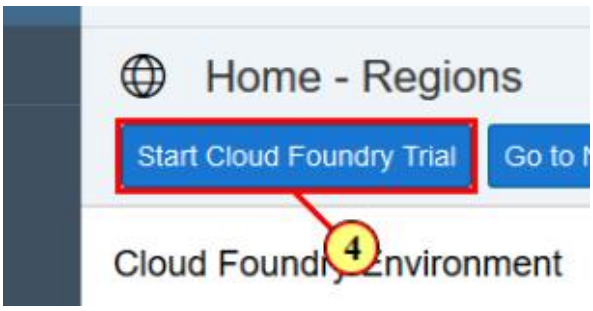
1. Create an SAP Cloud Platform Trial account.
2. Login to the SAP Cloud Platform Trial account and start the Cloud Foundry Trial Environment.
3. Open SAP Web IDE.
4. Enable SAP HANA Tools in SAP Web IDE.

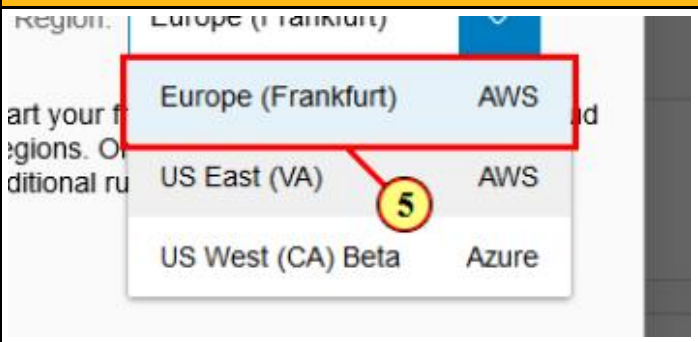
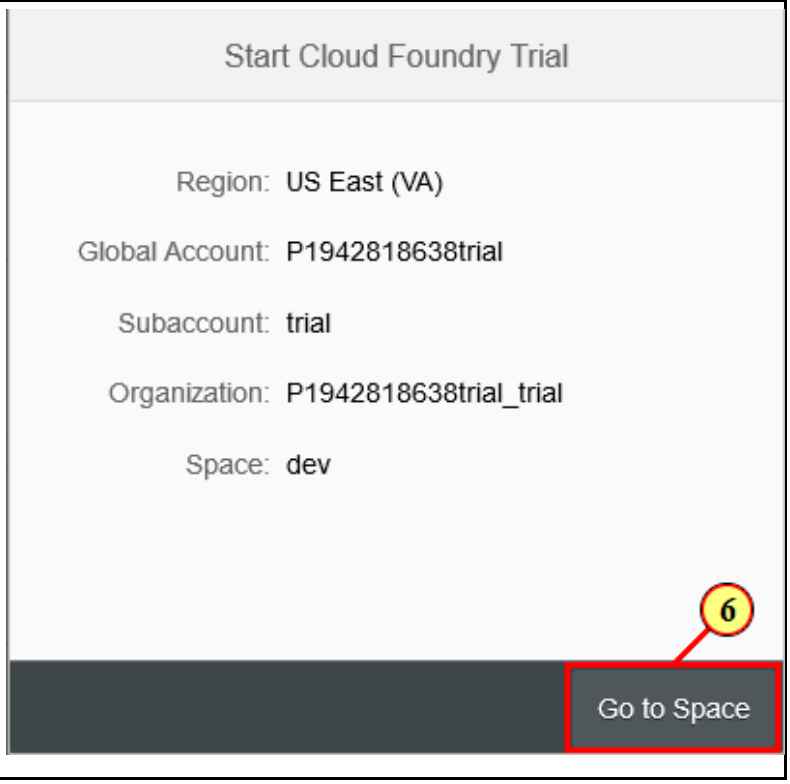
2.1 Create an SAP Cloud Platform Trial Account

Explanation	Screenshot
1. Enter **cloudplatform.sap.com** in the browser.	
2. Click Try it for Free.	
3. Enter the required fields in the registration form.	

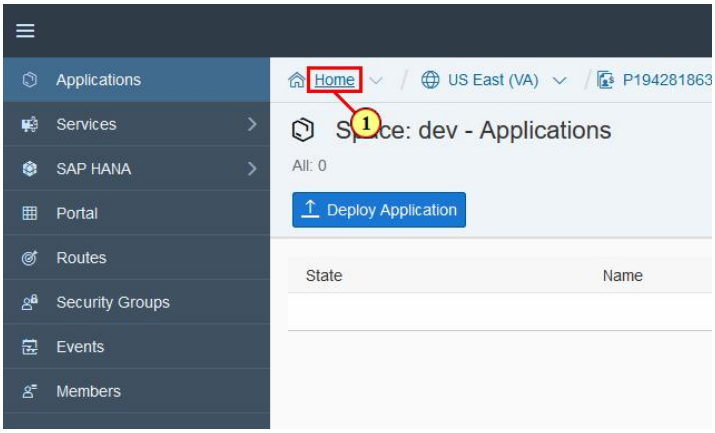
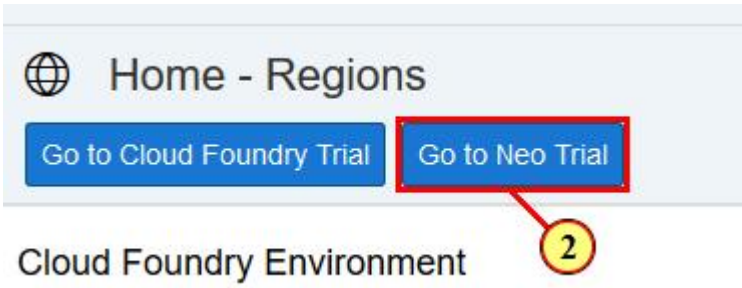
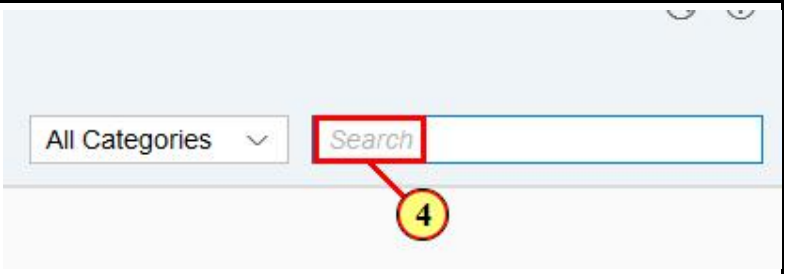
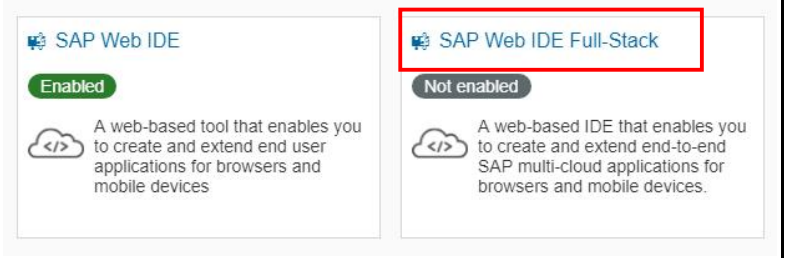
Explanation	Screenshot
4. Click **Register**.	
5. Close the browser.	
6. Open the confirmation email, and click **Click here to activate your account**.	
7. Click **Continue**.	

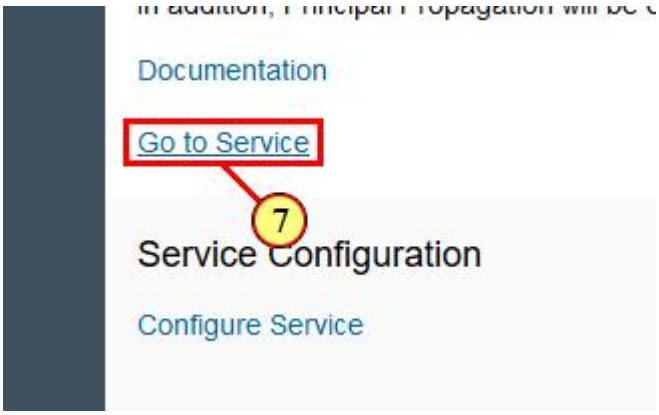
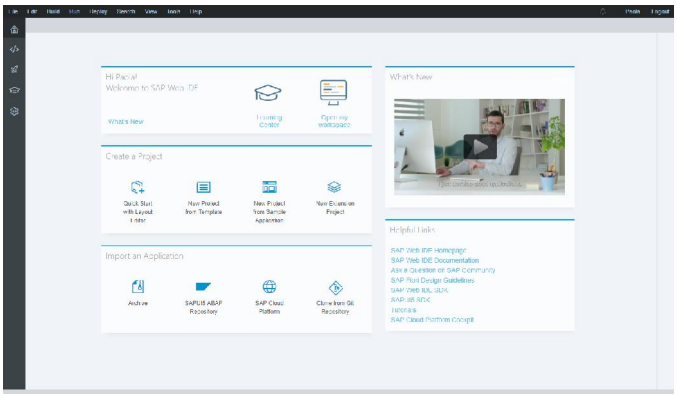
2.2 Log In to the SAP Cloud Platform Trial Account and Start the Cloud Foundry Trial Environment

Explanation	Screenshot
1. Enter **cloudplatform.sap.com** in the browser and then click Login.	
2. Enter your credentials.	
3. Click Log On.	
4. In the SAP Cloud Platform cockpit, click Home and then Start Cloud Foundry Trial.	

Explanation	Screenshot
5. From the **Region** dropdown list, select **Europe (Frankfurt)**, and click **OK**.	
6. Click Go to Space.	

2.3 Open SAP Web IDE

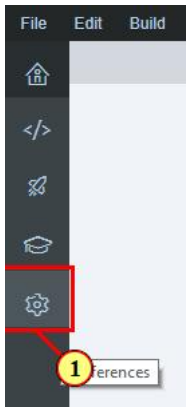
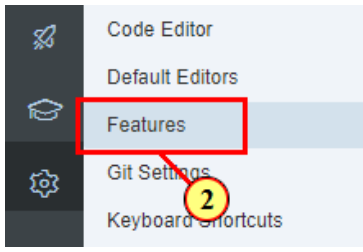
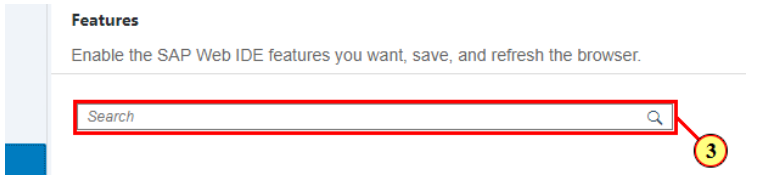
Explanation	Screenshot
1. In the SAP Cloud Platform cockpit, click Home.	
2. Click Go to Neo Trial.	
3. Click Services.	
4. Search for **web ide**.	
5. Select the **SAP Web IDE Full-Stack** tile.	

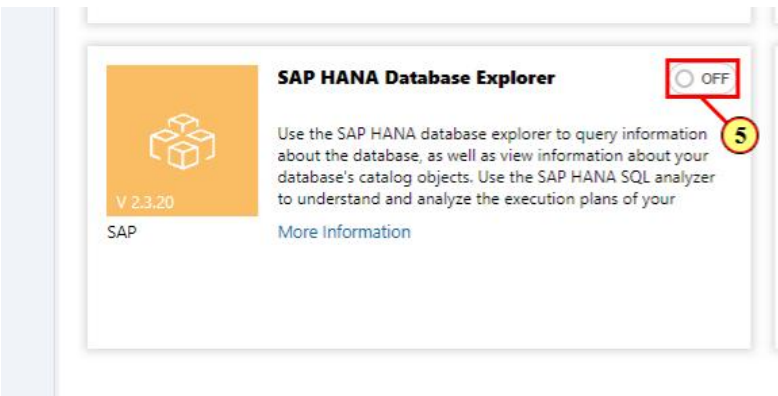
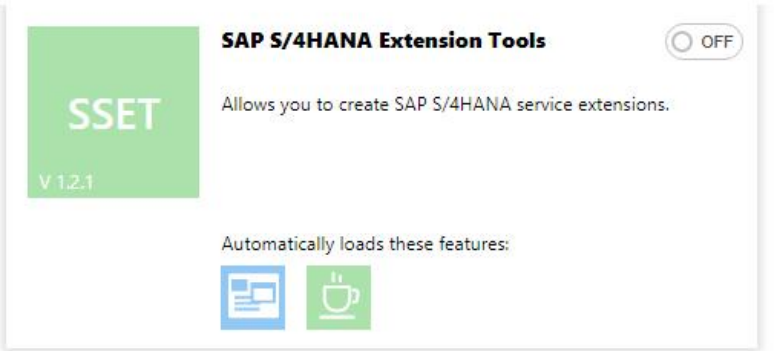
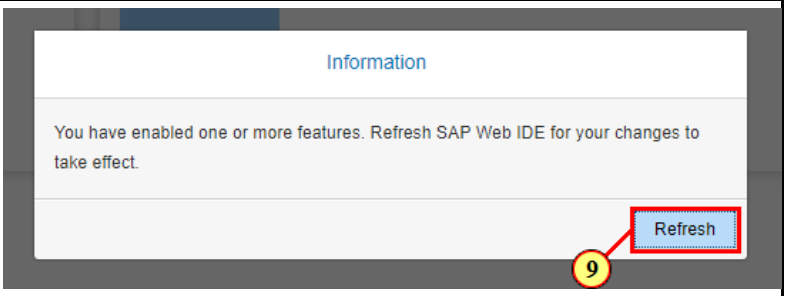
Explanation	Screenshot
6. Click Enable.	
7. Click Go to Service.	
8. SAP Web IDE opens.	

2.4 Enable SAP HANA and Java Tools

SAP Web IDE contains a wide set of features that are enabled by default. There are additional features that you can enable upon request.

For this exercise, we will enable the SAP HANA and Java tools.

Explanation	Screenshot
1. Click  to open the preferences perspective.	
2. Click **Features**.	
3. Enter **HANA** in the search field.	
4. Look for the **Tools for SAP HANA Database Development** feature and enable it by clicking the **On/Off** toggle button.	

Explanation	Screenshot
5. Look for the **SAP HANA Database Explorer** feature and enable it by clicking the **On/Off** toggle button.	
6. Clear the search field and enter **Java**.	
7. Look for the **SAP S/4HANA Extension Tools** feature and enable it by clicking the **On/Off** toggle button. 8. Click Save.	
9. Click Refresh to reload SAP Web IDE with the new features.	

Summary

You have completed the exercise!

You are now able to:

- Access the SAP Cloud Platform Trial account.
- Open SAP Web IDE.
- Enable features within SAP Web IDE.

3 CREATE A BOOK STORE APPLICATION

Overview

Estimated time: 90 minutes

Objective

In the following exercise you will learn how to create a full-stack application from within SAP Web IDE.

The application will include 3 modules:

Database (based on SAP HANA)	The database module uses the SAP HANA Core Data Services (CDS). With CDS, you can define a persistence model that includes objects such as tables, views, and structured types; the database objects specify what data to make accessible for consumption by applications and how. For more information, click here.
Service (Java)	The service module exposes an OData V4 service based on a Java framework component that allows the automatic exposure of the CDS service definition as a read-only data service.
UI (SAPUI5)	The UI module contains the user interface application created using SAPUI5. SAPUI5 is a collection of libraries that developers can use to build desktop and mobile applications that run in a browser. For more information, click here.

At the end of the exercise you will have an application displaying the list of books available in the bookstore.

Exercise Description

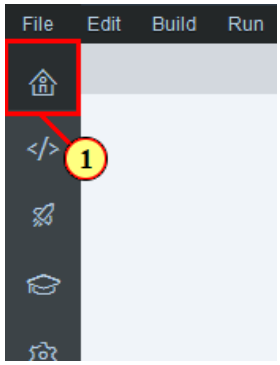
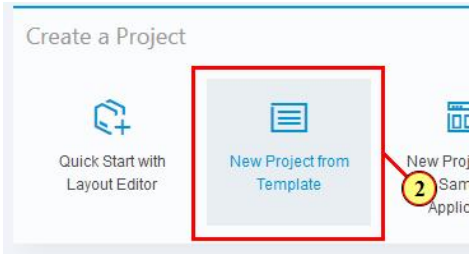
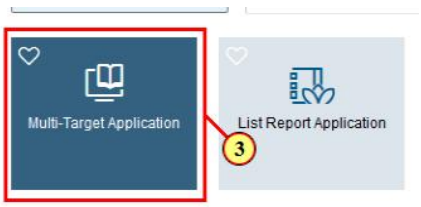
- Create a new project in SAP Web IDE
- Define the application data model
- Add data to the database
- Create the OData service
- Create an interface for the application

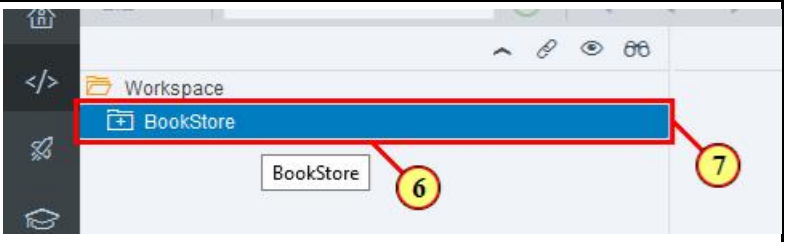
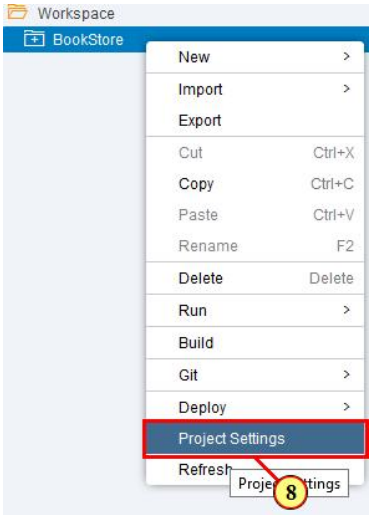
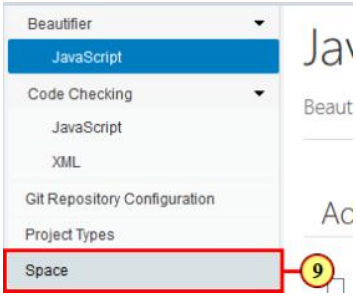
3.1 Create a New Project in SAP Web IDE

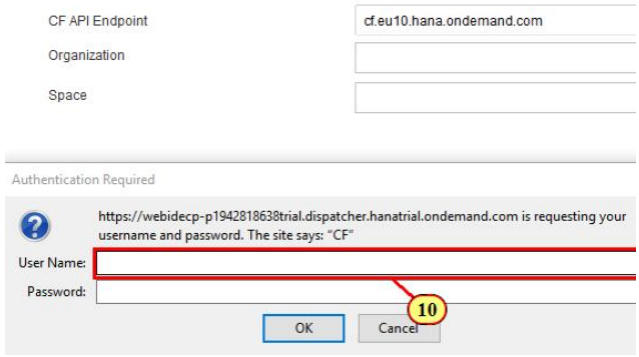
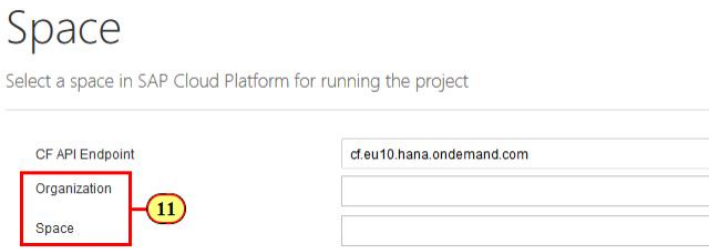
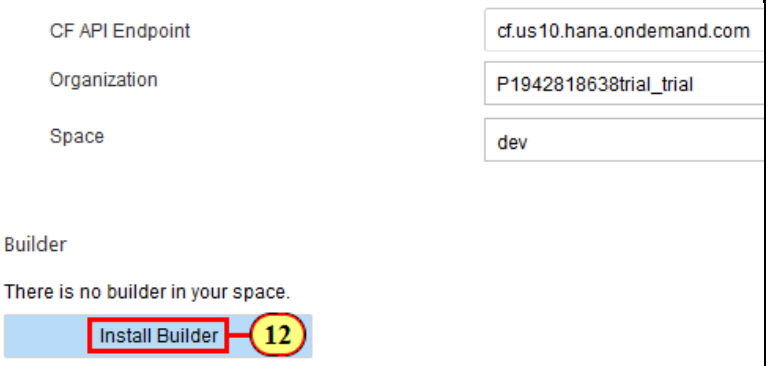
This step describes how to create a Multi-Target Application (MTA) project in SAP Web IDE.

An MTA is an application comprised of multiple software modules which are created with different technologies and can be deployed to different target platforms, yet they share the same lifecycle.

Each MTA project contains an **mta.yaml** file that is used to define the elements of the application and the dependencies between them.

Explanation	Screenshot
1. Open the **Welcome Page** perspective.	
2. Click **New Project from Template**.	
3. Select the **Multi-Target Application** tile, and click **Next**.	
4. Enter **BookStore** as the project name, and click **Next**.	

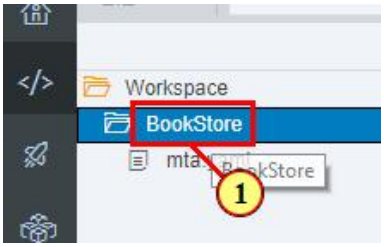
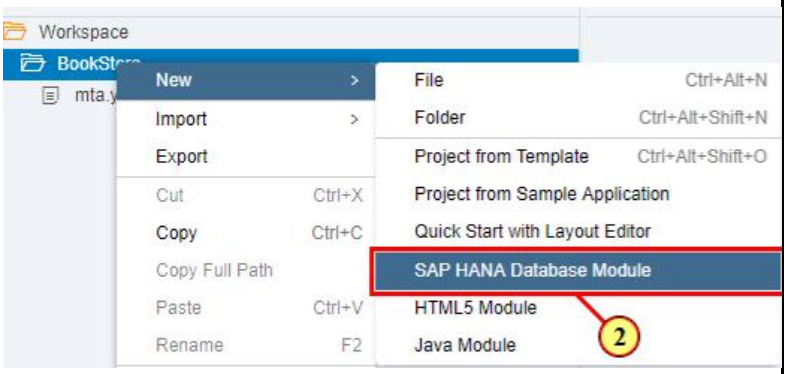
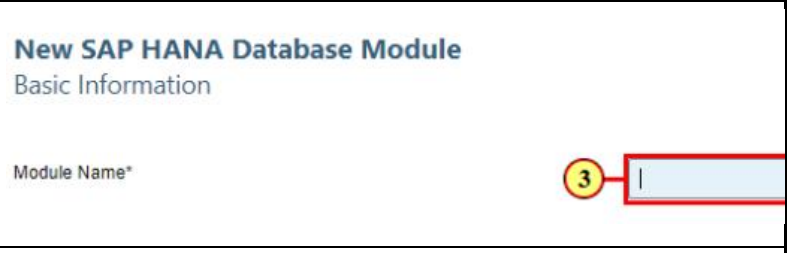
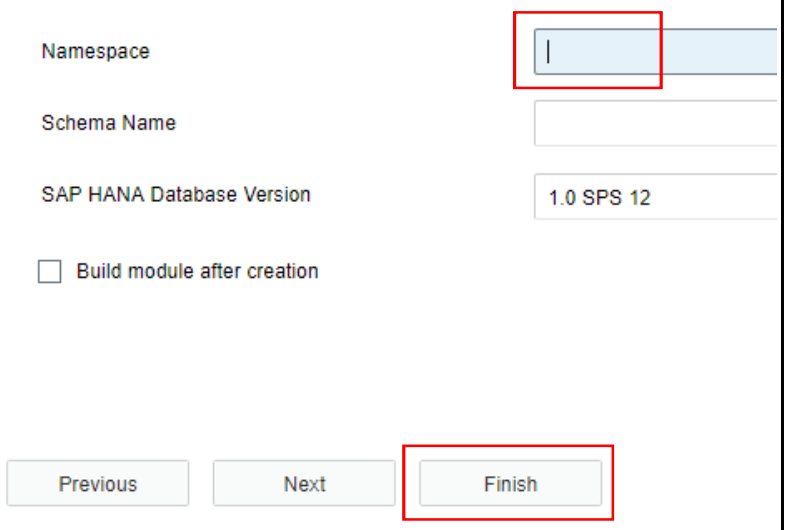
Explanation	Screenshot
5. Click Finish. The **Bookstore** project is created in your Workspace.	
6. You will now define a Cloud Foundry org and space. You will be able to use this space for development needs such as running the Java service. The org and space are also required to install a builder. 7. Right-click on the **Bookstore** project.	
8. Click **Project Settings**.	
9. Click **Space**.	

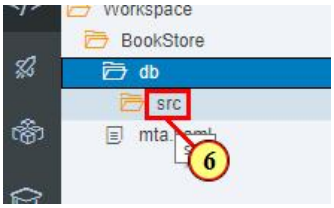
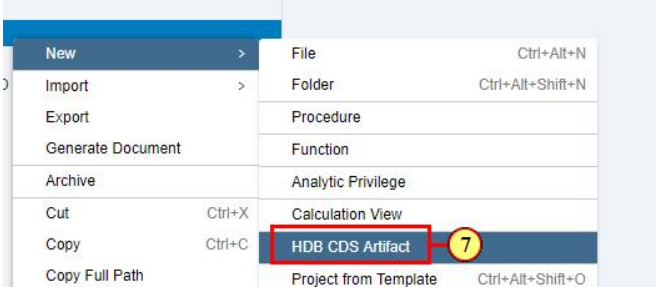
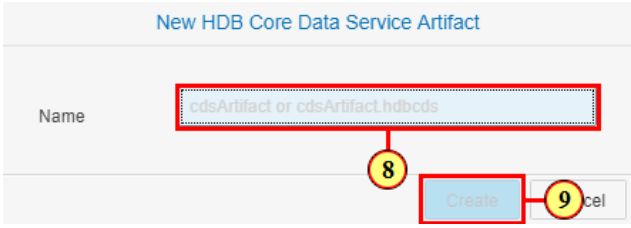
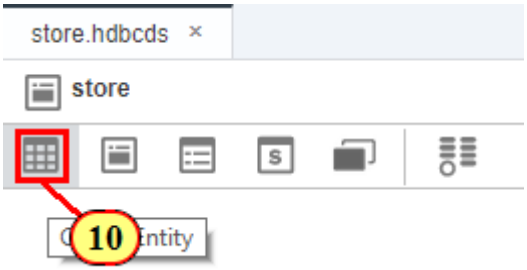
Explanation	Screenshot
Make sure the **CF API Endpoint** is: **cf.eu10.hana.ondemand.com** 10. Enter your Cloud Platform account credentials (email and password).	
11. Make sure the correct **Organization** and **Space** appear as defined in the Trial account you created in the previous section.	
12. Click Install Builder to install the builder in the Cloud Foundry space defined above. This might take a couple of minutes. When the builder is installed, click **Save**.	

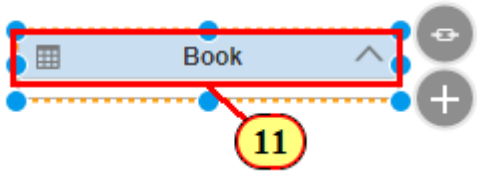
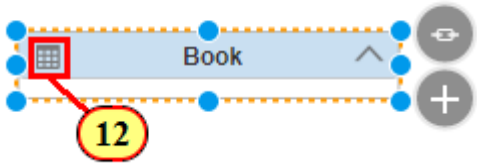
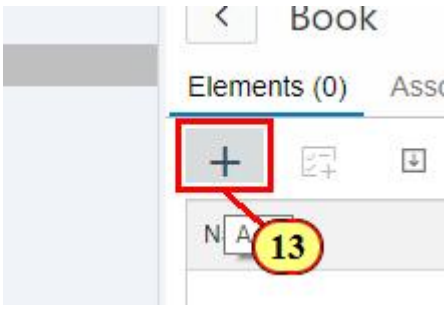
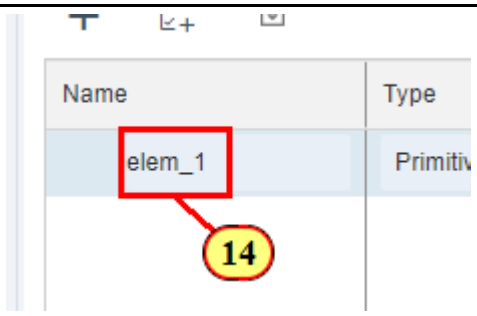
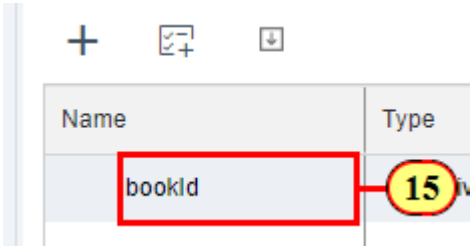
3.2 Define the Application Data Model

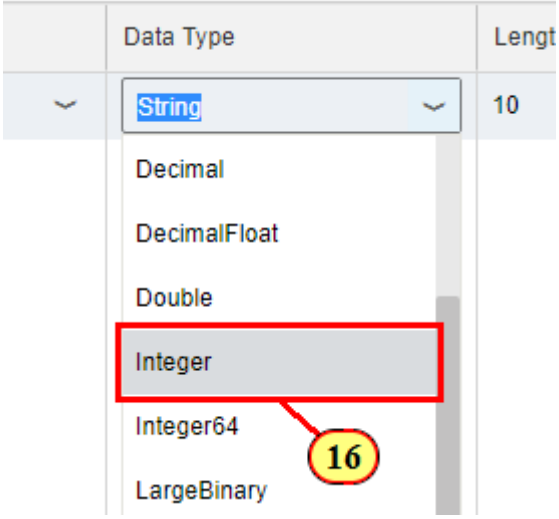
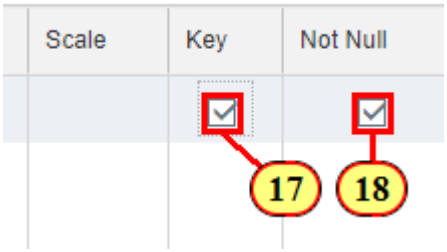
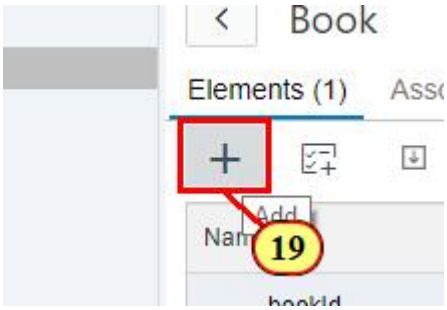
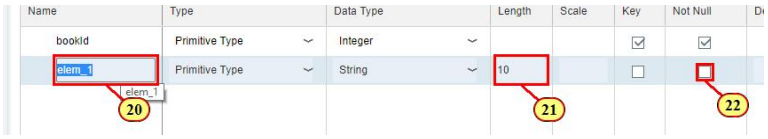
In this step, you will create the HDB module within which you will define the database schemas and tables.

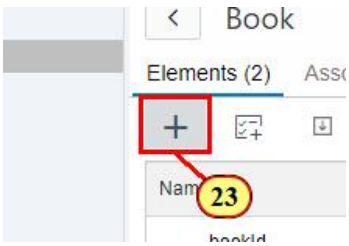
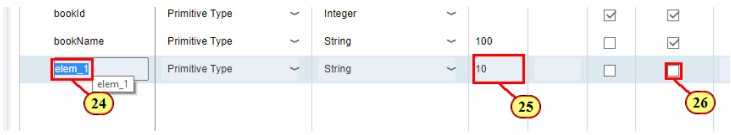
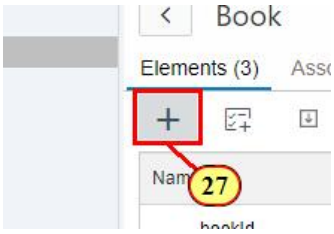
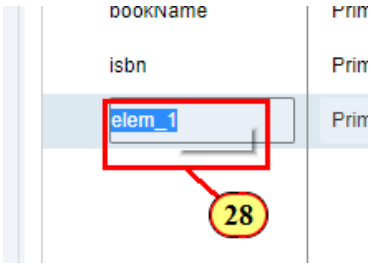
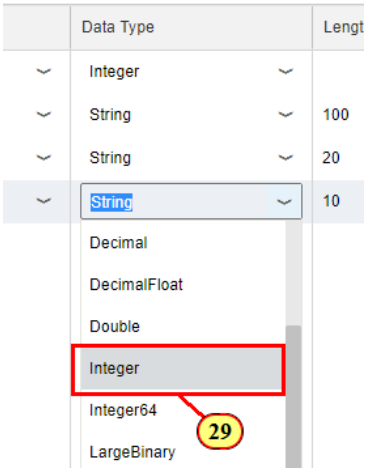
You can use this module in SAP Web IDE to define other SAP HANA artifacts such as calculation views.

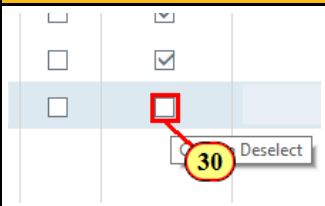
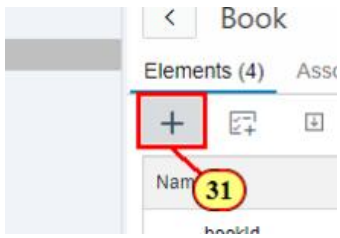
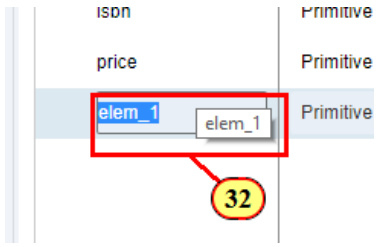
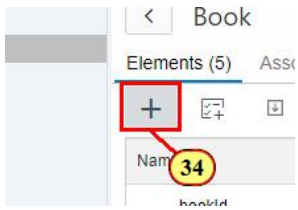
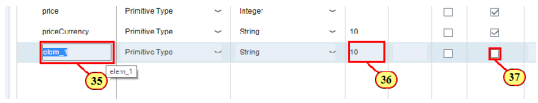
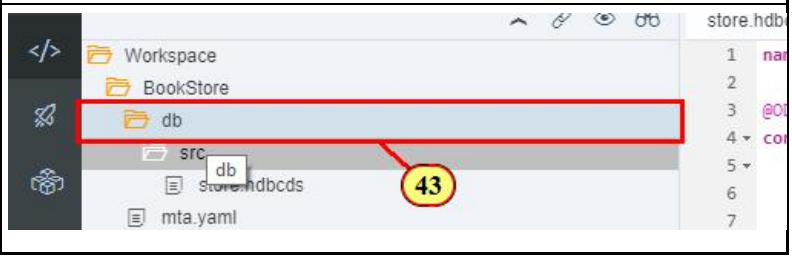
Explanation	Screenshot
1. Right-click the **Bookstore** project.	
2. Select **New > SAP HANA Database Module**.	
3. Enter **db** as the module name, and click **Next**.	
4. Clear the **Namespace** field and click Finish. The **hdb** module is created within your MTA project. You can open the **mta.yaml** file in your project to see the module that was created and the dependencies between the module and the HDI container.	

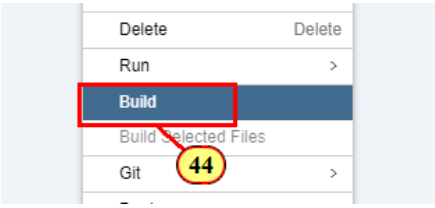
Explanation	Screenshot
5. You will now add an HDB CDS artifact to your module to define the schemas and the tables of the database. Open the **db** folder.	
6. Right-click src.	
7. Select **New > HDB CDS Artifact**.	
8. In the **Name** field, enter **store**. 9. Click Create. The **store.hdbcds** file is created and opened in the graphical modeler.	
10. Click in the editor and then click on the canvas to create a new entity.	

Explanation	Screenshot
11. Enter **Book** as the name of the entity.	
12. Double-click to open the Book entity definition.	
13. Click + to add properties to the entity.	
14. Click elem_1 .	
15. Enter **bookId** in the name column.	

Explanation	Screenshot
16. Change the **Data Type** to **Integer**	
17. Select the checkbox to mark the property as **Key**. 18. Select the checkbox to mark the property as **Not Null**.	
19. Click + to add another property.	
20. Enter **bookName** in the **Name** column. 21. Enter **100** in the **Length** field. 22. Select the checkbox to mark the property as **Not Null**.	

Explanation	Screenshot
23. Click + to add another property.	
24. Enter **isbn** in the **Name** column. 25. Enter **20** in the **Length** column. 26. Select the checkbox to mark the property as **Not Null**.	
27. Click + to add another property.	
28. Enter **price** in the **Name** column.	
29. Change the **Data Type** to **Integer**	

Explanation	Screenshot
30. Select the checkbox to mark the property as **Not Null**.	
31. Click + to add another property.	
32. Enter **priceCurrency** in the **Name** column. 33. Select the checkbox to mark the property as **Not Null**.	
34. Click + to add another property.	
35. Enter **authorName** in the **Name** column. 36. Enter **100** in the **Length** column. 37. Select the checkbox to mark the property as **Not Null**. 38. Click Save.	
39. Go to **BookStore > db**.	

Explanation	Screenshot
40. Right-click **db,** and click **Build** to create the schema and tables in the SAP HANA database.	

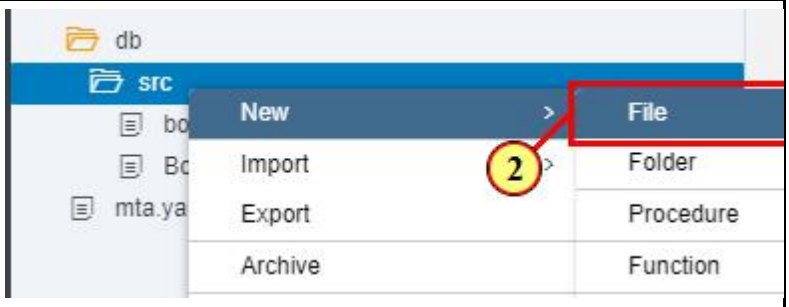
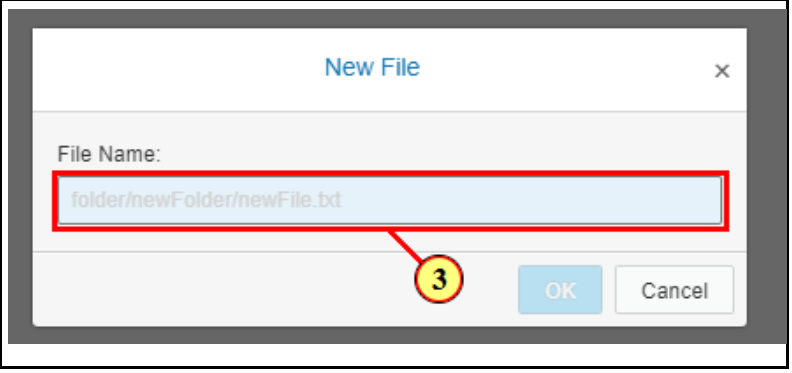
3.3 Add Data to the Database

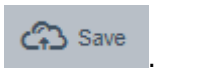
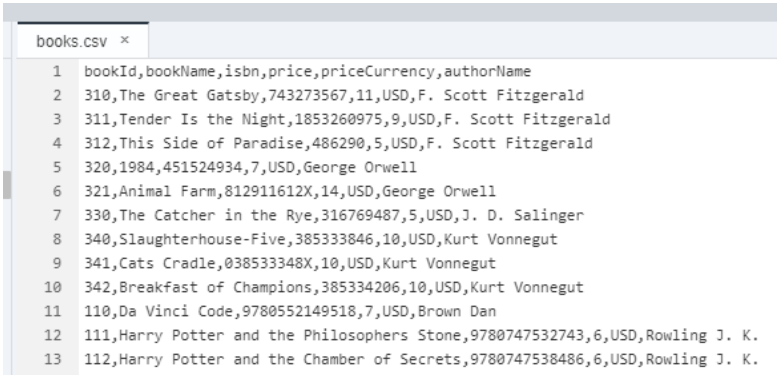
In this step, you will create a **CSV** file that will contain the data for your app.

You will then create an **hdbtabledata** file with the following content:

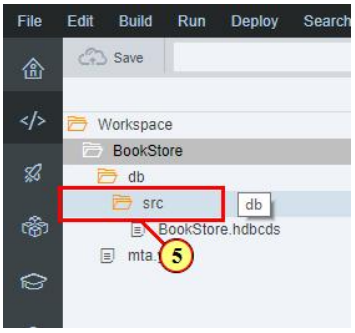
`target_table`	The name of the data table in which the data should be inserted.
`source_data`	Describes the structure of the **CSV** file you created which is used as the data source for the import.
`import_settings`	Specifies the target table columns for the data that is inserted into the target table
`column_mappings`	Connects the target tables column (specified in import_settings) with the data specified in source data.

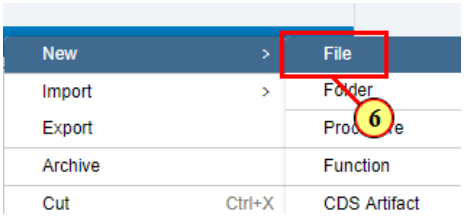
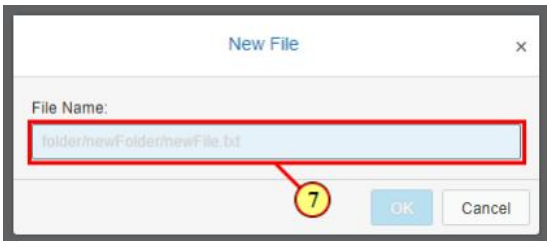
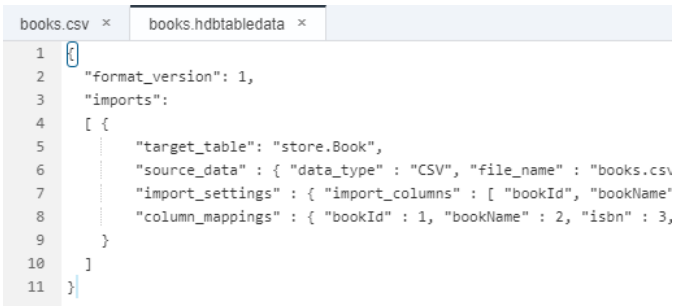
The file is used to insert data from files such as **CSV** into SAP HANA database tables.

Explanation	Screenshot
1. Go to **BookStore > db > src**.	
2. Right-click the **src** folder, and click **New > File**.	
3. Enter **books.csv** and, click **OK**.	

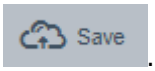
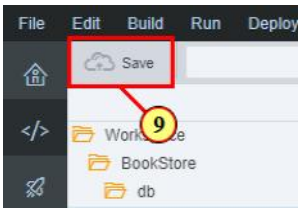
Explanation	Screenshot
4. Copy the following text to the created file. Click  .	

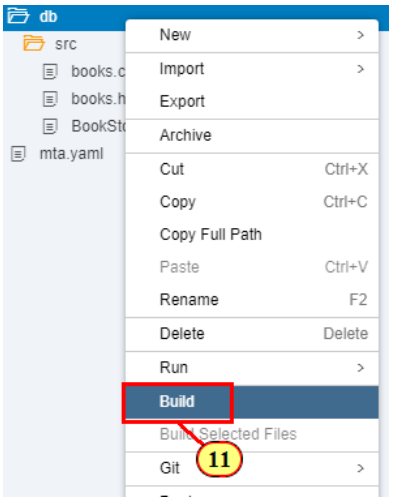
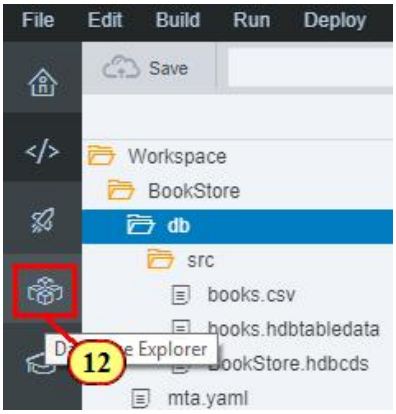
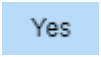
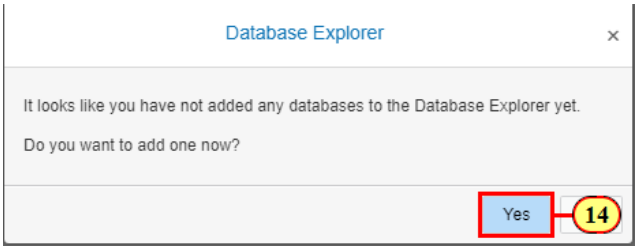

```
bookId,bookName,isbn,price,priceCurrency,authorName
310,The Great Gatsby,743273567,11,USD,F. Scott Fitzgerald
311,Tender Is the Night,1853260975,9,USD,F. Scott Fitzgerald
312,This Side of Paradise,486290,5,USD,F. Scott Fitzgerald
320,1984,451524934,7,USD,George Orwell
321,Animal Farm,812911612X,14,USD,George Orwell
330,The Catcher in the Rye,316769487,5,USD,J. D. Salinger
340,Slaughterhouse-Five,385333846,10,USD,Kurt Vonnegut
341,Cats Cradle,038533348X,10,USD,Kurt Vonnegut
342,Breakfast of Champions,385334206,10,USD,Kurt Vonnegut
110,Da Vinci Code,9780552149518,7,USD,Brown Dan
120,Angels and Demons,9780552150736,7,USD,Brown Dan
130,Harry Potter and the Half-blood PrinceChildrens
Edition,9780747581086,16,USD,Rowling J. K.
142,Twilight,9781904233657,7,USD,Meyer Stephenie
143,Harry Potter and the Goblet of Fire,9780747550,8,USD,Rowling J. K.
144,Deception Point,9780552151764,7,USD,Brown Dan
145,New Moon,9781904233886,7,USD,Meyer Stephenie
146,Lovely Bones,9780330457729,7,USD,Sebold Alice
147,Digital Fortress,9780552151696,7,USD,Brown Dan
148,Curious Incident of the Dog in the Night-time,9780450252,7,USD,Haddon
Mark
149,Eclipse,9781904233916,7,USD,Meyer Stephenie
150,Girl with the Dragon Tattoo,9781847245458,7,USD,Larsson Stieg
151,Kite Runner,9780747566533,7,USD,Hosseini Khaled
152,Time Travelers Wife,9780464464,7,USD,Niffenegger Audrey
153,World According to Clarkson,9780141017891,7,USD,Clarkson Jeremy
154,Atonement,9780429791,7,USD,McEwan Ian
155,Lost Symbol,9780593054277,18,USD,Brown Dan
156,Short History of Nearly Everything,9780557041,9,USD,Bryson Bill
157,Breaking Dawn,9781905654284,14,USD,Meyer Stephenie
158,Harry Potter and the Goblet of Fire,9780747546245,17,USD,Rowling J. K.
160,Girl Who Played With Fire,9781849163422,7,USD,Larsson Stieg
161,Child Called It,9780752837505,6,USD,Pelzer Dave
```

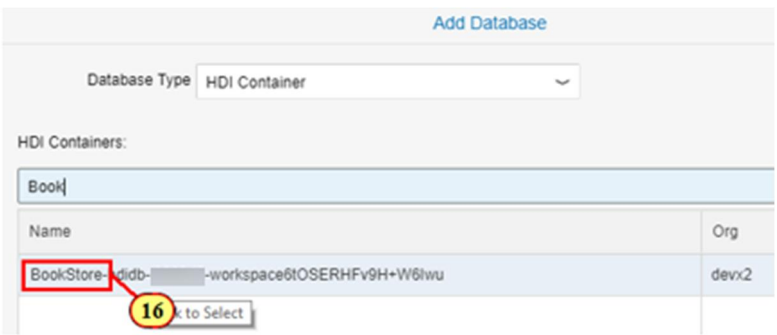
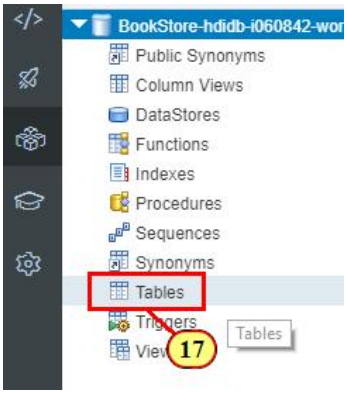
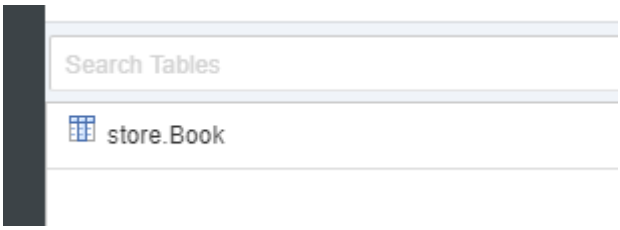
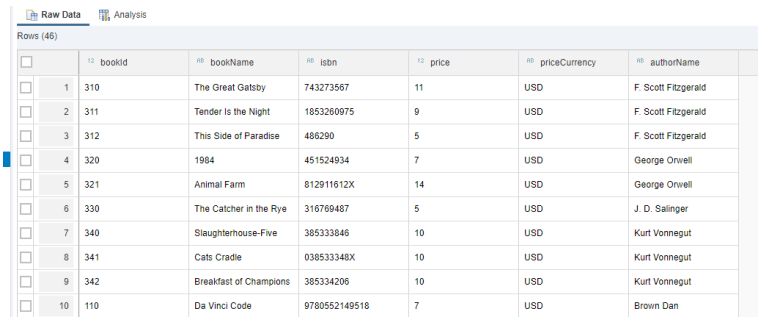
5. Go to **BookStore > db > src**.	

Explanation	Screenshot
6. Right-click the **src** folder, and click **New > File**.	
7. Enter **books.hdbtabledata**, and click **OK**.	
8. Add the following content to the file that was created:	

```
{
  "format_version": 1,
  "imports":
  [ {
            "target_table": "store.Book",
            "source_data" : { "data_type" : "CSV", "file_name" :
"books.csv", "has_header" : true },
            "import_settings" : { "import_columns" : [ "bookId",
"bookName","isbn","price","priceCurrency", "authorName" ] },
            "column_mappings" : { "bookId" : 1, "bookName" : 2, "isbn" :
3, "price": 4, "priceCurrency" : 5, "authorName": 6   }
      }
  ]
}
```

9. Click  .	
10. Go to **BookStore > db**.	

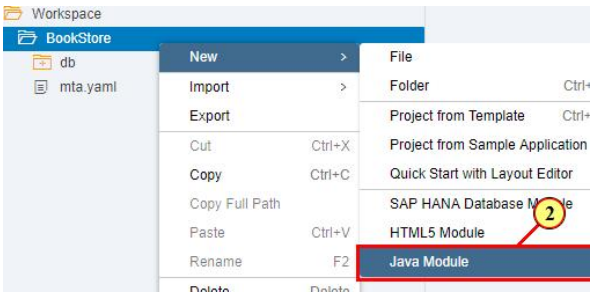
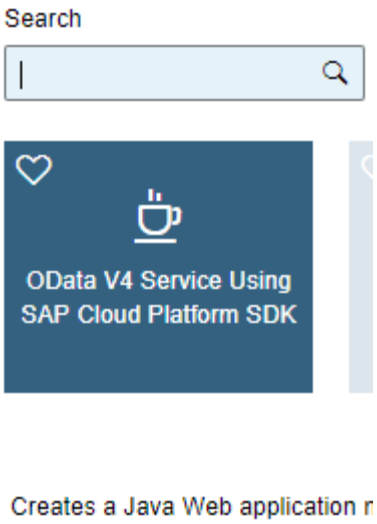
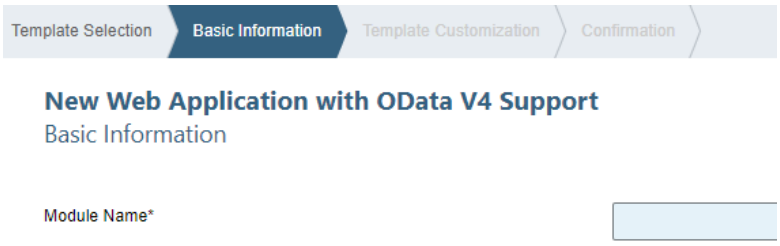
Explanation	Screenshot
11. Right-click the **db** folder, and click **Build** to add the data to the book table in the SAP HANA Database.	
12. Click  to open the Database Explorer. In the Database Explorer you can view information about your database's catalog objects and execute queries.	
13. Click **Connect** to connect to the Database Explorer.	
14. Click  to add the database you created.	
15. Under **HDI Containers**, enter **Book**.	

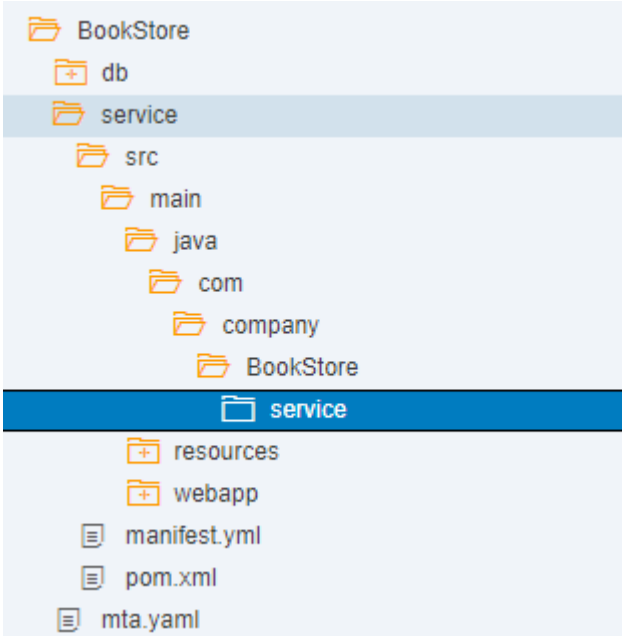
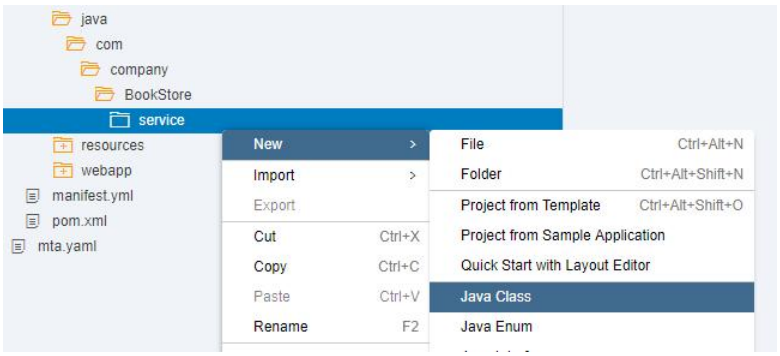
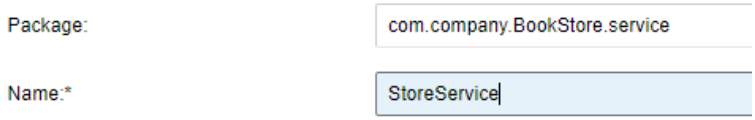
Explanation	Screenshot
16. Select the **BookStore** container, and click **OK**. Your database is added to the workspace and you can explore it.	
17. Click **Tables** to explore the existing tables in your database.	
18. Click the Book table .	
19. In the tab that opens you can see all the information connected to the table. Click Open Data to view the table data.	
	

3.4 Create the OData Service

In this step, you will create the Java module within which you will create and expose the OData service using the SAP Cloud Platform SDK for service development.

This SDK is a set of libraries that help you, as a developer, to build a new OData service based on the Java runtime and deployable on Cloud Foundry environment of SAP Cloud Platform. For more information, see the SAP Cloud Platform SDK documentation.

Explanation	Screenshot
1. Right-click the **BookStore** project.	
2. Select **New > Java Module**.	
3. Select the **OData V4 Service Using SAP Cloud Platform SDK** template, and click **Next**. This template uses the Cloud Extension SDK to generate the dependencies required to provide an OData V4 service.	
4. Enter **service** as the name of the JAVA module. Click **Next** and then **Finish**. The **java** module is added to your MTA project.	

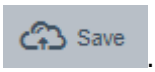
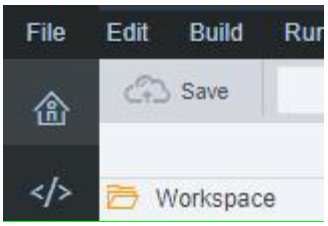
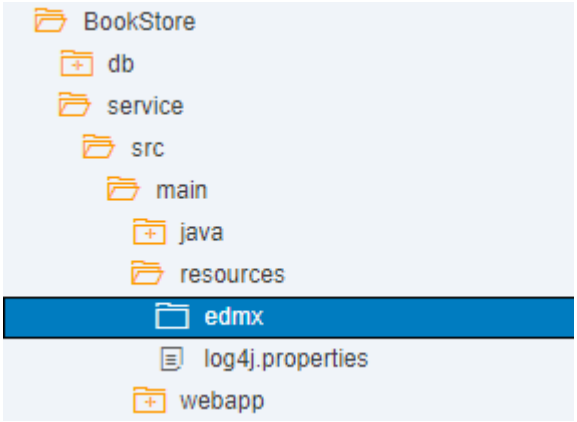
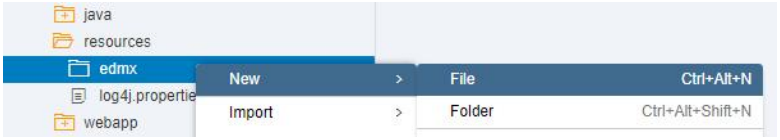
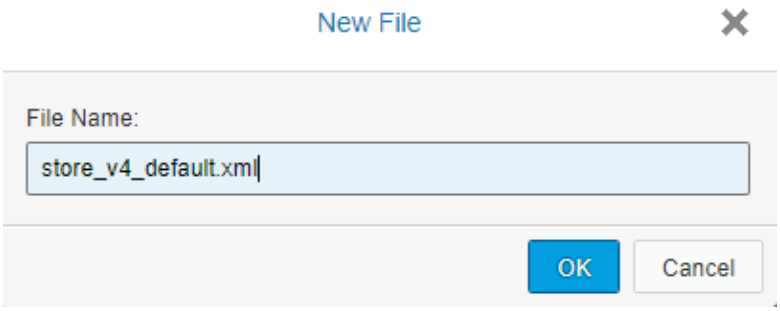
Explanation	Screenshot
You can open the **mta.yaml** file in your project to see the module that was created and the dependencies between the module and the HDI container.	
5. Open the **service** module and expand the tree until you reach the **service** folder.	
6. Right-click **service** and select **New >Java Class**.	
7. Enter **StoreService** in the **Name** field and click **Next**. 8. Click **Finish**. A new Java class is created and added to your Java project.	

Explanation	Screenshot
9. Double-click the **StoreService.java** file to open it in the editor. Implement the **Query** and **Read** functions to enable the **Book** service to be exposed. 10. Replace the current content in the **StoreService.java** file with the code below.	

```
package com.company.BookStore.service;

import java.util.List;

import java.sql.Connection;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.sql.DataSource;

import com.sap.cloud.sdk.hana.connectivity.cds.CDSQuery;
import com.sap.cloud.sdk.hana.connectivity.cds.CDSSelectQueryBuilder;
import com.sap.cloud.sdk.hana.connectivity.cds.CDSSelectQueryResult;

import com.sap.cloud.sdk.hana.connectivity.handler.CDSDataSourceHandler;
import
com.sap.cloud.sdk.hana.connectivity.handler.DataSourceHandlerFactory;

import com.sap.cloud.sdk.service.prov.api.EntityData;
import com.sap.cloud.sdk.service.prov.api.operations.Query;
import com.sap.cloud.sdk.service.prov.api.operations.Read;
import com.sap.cloud.sdk.service.prov.api.request.QueryRequest;
import com.sap.cloud.sdk.service.prov.api.request.ReadRequest;
import com.sap.cloud.sdk.service.prov.api.response.QueryResponse;
import com.sap.cloud.sdk.service.prov.api.response.ReadResponse;

public class StoreService {

      private static Logger logger =
LoggerFactory.getLogger(StoreService.class);


      @Query(entity = "Book", serviceName = "store")
      public QueryResponse getAllProposedBooks(QueryRequest queryRequest)
{
          try {
                  QueryResponse queryResponse =
QueryResponse.setSuccess().setEntityData(getEntitySet(queryRequest)).respo
nse();
                  return queryResponse;
            } catch (Exception e) {
                  return null;
      }
```

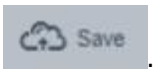
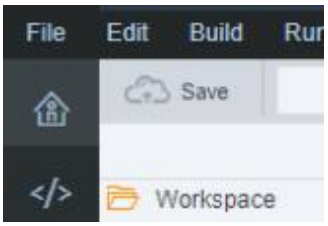
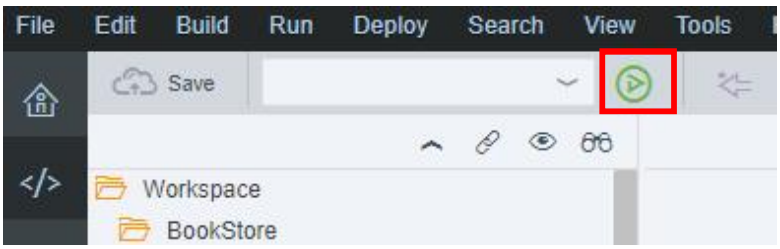
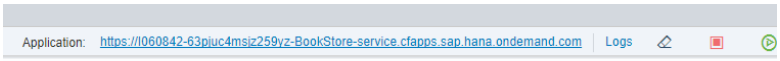
Explanation	Screenshot

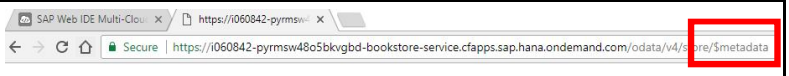
```
	}

	@Read(entity = "Book", serviceName = "store")
	public ReadResponse getProposedBooks(ReadRequest readRequest){
		try {
			ReadResponse readResponse =
ReadResponse.setSuccess().setData(readEntity(readRequest)).response();
			return readResponse;
		} catch (Exception e) {
			return null;
		}
	}

	private List<EntityData> getEntitySet(QueryRequest queryRequest) {
		String fullQualifiedName =
queryRequest.getEntityMetadata().getNamespace()+ "."
+queryRequest.getEntityMetadata().getName();
		CDSDataSourceHandler dsHandler =
DataSourceHandlerFactory.getInstance().getCDSHandler(getConnection(),
queryRequest.getEntityMetadata().getNamespace());
		try {
			CDSQuery cdsQuery = new
CDSSelectQueryBuilder(fullQualifiedName).build();
			CDSSelectQueryResult cdsSelectQueryResult =
dsHandler.executeQuery(cdsQuery);
			return cdsSelectQueryResult.getResult();
		} catch (Exception e) {
			logger.error("==> Eexception while fetching query data
from CDS: " + e.getMessage());
			e.printStackTrace();
		}
		return null;
	}

	private EntityData readEntity(ReadRequest readRequest) throws
Exception {
		CDSDataSourceHandler dsHandler =
DataSourceHandlerFactory.getInstance().getCDSHandler(getConnection(),
readRequest.getEntityMetadata().getNamespace());
		EntityData ed =
dsHandler.executeRead(readRequest.getEntityMetadata().getName(),
readRequest.getKeys(), readRequest.getEntityMetadata().getElementNames());
		return ed;
	}

	private static Connection getConnection(){
		Connection conn = null;
		Context ctx;
		try {
			ctx = new InitialContext();
			conn = ((DataSource)
ctx.lookup("java:comp/env/jdbc/java-hdi-container")).getConnection();
			System.out.println("conn = " + conn);
		} catch (Exception e) {
			e.printStackTrace();
		}
		return conn;
	}

}
```

Explanation	Screenshot
11. Click  .	
12. Go to **BookStore > service > src > main > resources** and right-click the **edmx** folder.	
13. Select **New >File**.	
14. Enter **store_v4_default.xml** as the file name. This file will contain the metadata of the OData service.	
15. Double click the **store_v4_default.xml** file and add the following code:	

Explanation	Screenshot

`<?xml version="1.0" encoding="utf-8"?>`
 	
`<edmx:Edmx Version="4.0" xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx">`
`  <edmx:DataServices>`
`  <Schema Namespace="store" Alias="store" xmlns="http://docs.oasis-open.org/odata/ns/edm">`
`  <EntityContainer Name="EntityContainer">`
`  <EntitySet Name="Book" EntityType="store.Book"/>`
`  </EntityContainer>`
`  <EntityType Name="Book">`
`  <Key>`
`  <PropertyRef Name="bookId"/>`
`  </Key>`
`  <Property Name="bookId" Type="Edm.Int32" Nullable="false"/>`
`  <Property Name="bookName" Type="Edm.String" MaxLength="100" Nullable="false"/>`
`  <Property Name="isbn" Type="Edm.String" MaxLength="20" Nullable="false"/>`
`  <Property Name="price" Type="Edm.Int32" Nullable="false"/>`
`  <Property Name="priceCurrency" Type="Edm.String" MaxLength="10" Nullable="false"/>`
`  <Property Name="authorName" Type="Edm.String" MaxLength="100"/>`
`  </EntityType>`
`  </Schema>`
`  </edmx:DataServices>`
`</edmx:Edmx>`	

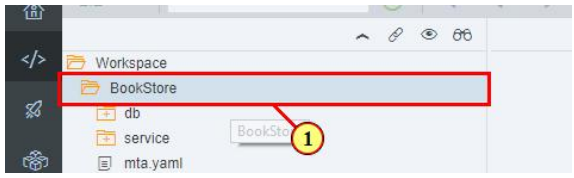
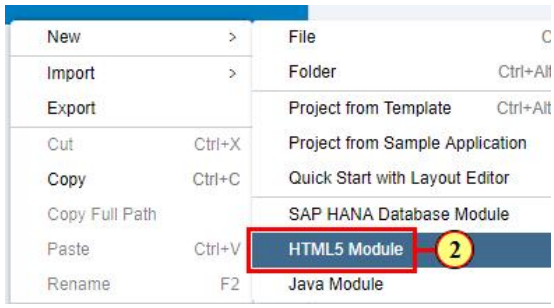
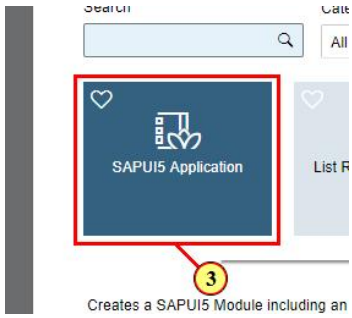
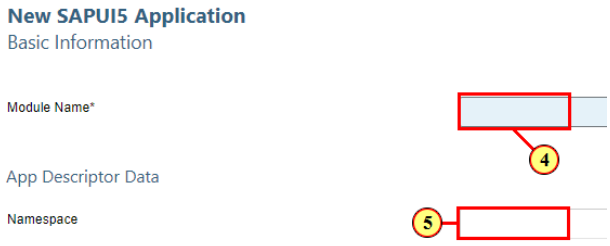
16. Click  .	
17. Click  to run the OData service. The Run console will open showing the status of your application.	
18. Once the application is ready, click the application URL.	

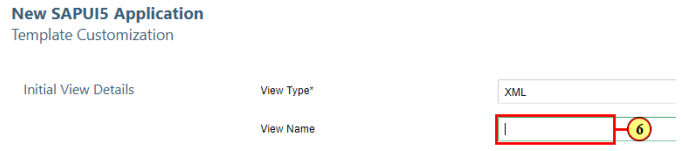
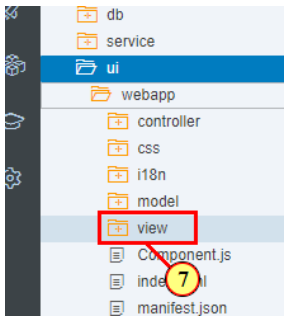
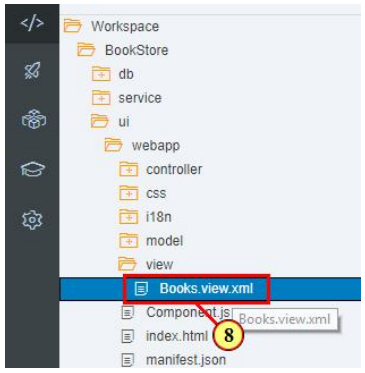
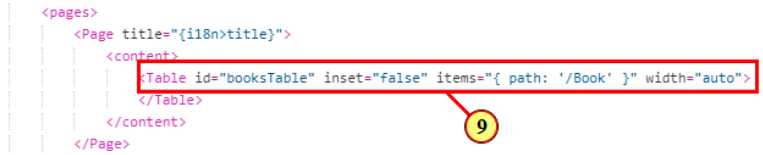
Explanation	Screenshot
19. In the tab that opens, add **/odata/v4/store/$metadata** to the URL to get the metadata of the service.	
20. In the URL, replace **$metadata** with **Book** to get the Book entity.	

3.5 Create an SAPUI5 Application

In this step, you will create the HTML5 module within which you will define the application's interface.

You will be using SAPUI5 to create the interface.

Explanation	Screenshot
1. Right-click the **BookStore** project.	
2. Click **New > HTML5 Module**.	
3. Select the **SAPUI5 Application** template, and click **Next**. This template creates a simple SAPUI5 application based on the mvc paradigm.	
4. Enter **ui** as the module name. 5. Enter **book** as the namespace. Click **Next**.	

Explanation	Screenshot
6. Enter **Books** as the View name, and click **Finish**. The **HTML5** module is created within your MTA project. You can open the **mta.yaml** file in your project to see the module that was created.	
7. You now need to create a table that will display the book information from the database. Go to **BookStore >ui > webapp > view**.	
8. Double-click Books.view.xml to open the file.	
9. Add the **Table** control to the page with the following content:	

```
<Table id="booksTable" inset="false" items="{ path: '/Book' }"
width="auto">
</Table>
```

Explanation	Screenshot

10. Add the **headerToolbar** to the table with the following content:	Books.view.xml
	`<mvc:View controllerName="book.ui.ui.controller.Books" xmlns:html="http://www.w3.org/1999/xhtml" xmlns:`
	`displayBlock="true" xmlns="sap.m">`
	`<App>`
	`<pages>`
	`<Page title="{i18n>title}">`
	`<content>`
	`<Table id="booksTable" inset="false" items="{ path: '/Book' }" width="auto">`
	`<headerToolbar>`
	`<Toolbar width="100%" id="__toolbar3">`
	`<content>`
	`<Bar id="__bar1">`
	`<contentLeft>`
	`<Title text="Books" width="100%" id="__title1"/>`
	`</contentLeft>`
	`</Bar>`
	`</content>`
	`</Toolbar>`
	`</headerToolbar>`
	`</Table>`
	`</content>`
	`</Page>`
	`</pages>`
	`</App>`
	`</mvc:View>`

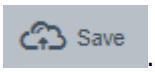
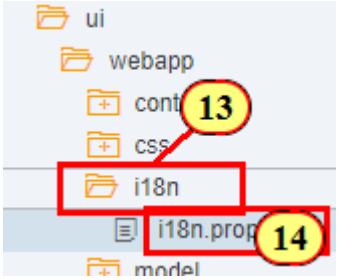
```
<headerToolbar>
<Toolbar width="100%" id="__toolbar3">
<content>
<Bar id="__bar1">
<contentLeft>
<Title text="{i18n>tableTitle}" width="100%" id="__title1"/>
</contentLeft>
</Bar>
</content>
</Toolbar>
</headerToolbar>
```

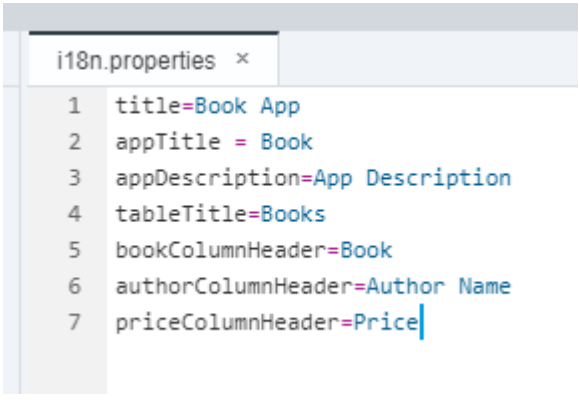
11. Add columns to the table with the following content:	`<Table id="booksTable" inset="false" items="{ path: '/Book' }" width="auto">`
	`<headerToolbar>`
	`<Toolbar width="100%" id="__toolbar3">`
	`<content>`
	`<Bar id="__bar1">`
	`<contentLeft>`
	`<Title text="Books" width="100%" id="__title1"/>`
	`</contentLeft>`
	`</Bar>`
	`</content>`
	`</Toolbar>`
	`</headerToolbar>`
	`<columns>`
	`<Column>`
	`<Text text="Book"/>`
	`</Column>`
	`<Column minScreenWidth="Tablet" demandPopin="true">`
	`<Text text="Author Name"/>`
	`</Column>`
	`<Column>`
	`<Text text="Price"/>`
	`</Column>`
	`</columns>`
	`</Table>`

```
<columns>
<Column>
<Text text="{i18n>bookColumnHeader}"/>
</Column>
<Column minScreenWidth="Tablet" demandPopin="true">
<Text text="{i18n>authorColumnHeader}"/>
</Column>
<Column>
<Text text="{i18n>priceColumnHeader}"/>
</Column>
</columns>
```

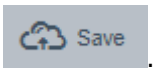
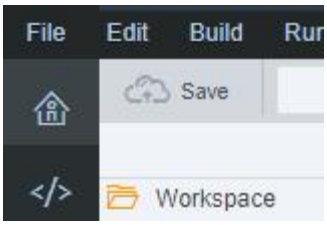
Explanation	Screenshot
12. Add items to the table with the following content:	

```
<items>
<ColumnListItem>
<cells>
<ObjectIdentifier title="{bookName}" text="{isbn}"/>
<Text text="{authorName}"/>
<ObjectNumber
number="{ parts:[{path:'price'},{path:'priceCurrency'}], type:
'sap.ui.model.type.Currency', formatOptions: {showMeasure: false} }"
unit="{priceCurrency}"/>
</cells>
</ColumnListItem>
</items>
```

13. Click  . Go to **BookStore > ui > webapp > i18n**. 14. Double-click i18n.properties.	

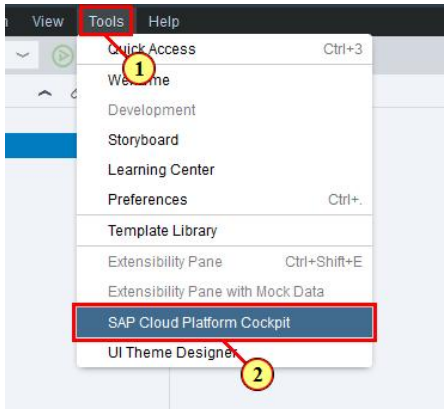
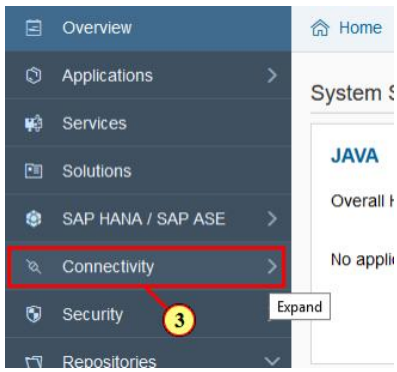
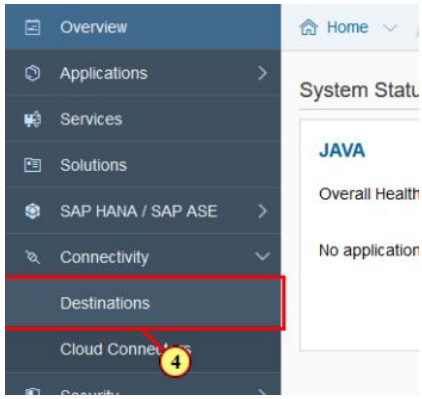
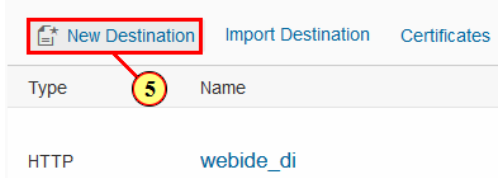
Explanation	Screenshot
15. Update the default translation strings as follows:	

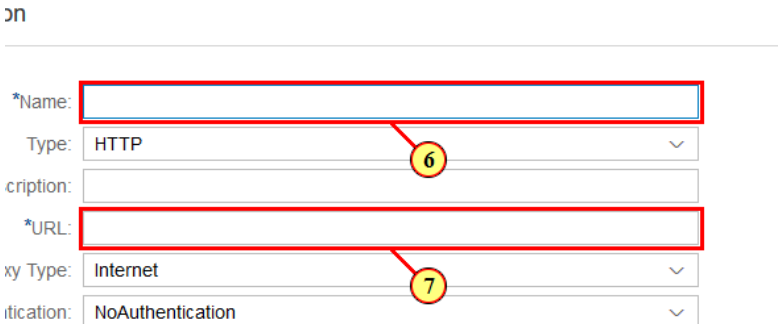
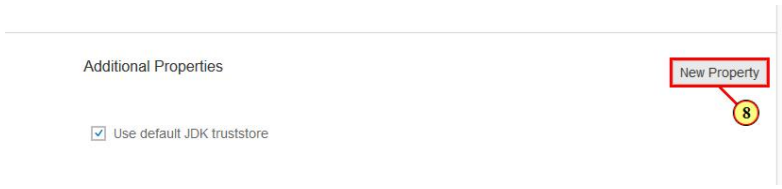
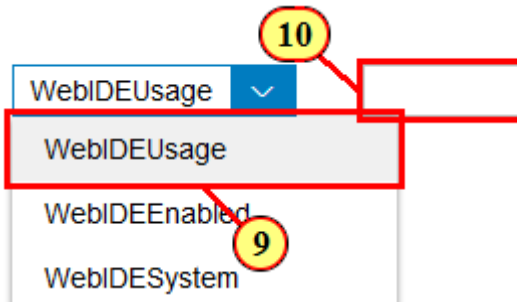
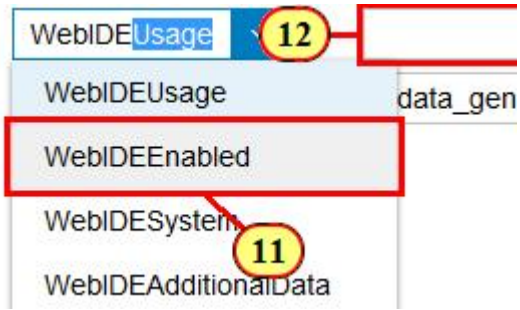
```
title=Book App
appTitle = Book
appDescription=App Description
tableTitle=Books
bookColumnHeader=Book
authorColumnHeader=Author Name
priceColumnHeader=Price
```

16. Click  .	

3.6 Create a Neo Destination

In order to allow the UI application to consume data from an external source (in this case the Java module), you need to create a destination in the Neo environment.

Explanation	Screenshot
1. Open the **Tools** menu. 2. Click SAP Cloud Platform Cockpit .	
3. Click **Connectivity**.	
4. Click **Destinations**.	
5. Click **New Destination**.	

Explanation	Screenshot
6. Enter **books_odata_v4** in the **Name** and **Description** fields. 7. In the **URL** field, enter the OData V4 service URL displayed in the run console that was created before.	
8. Click New Property.	
9. Start typing **Web**, and click WebIDEUsage. 10. Enter **odata_gen** in the description field.	
11. Click New Property. Start typing **Web** and click WebIDEEnabled. 12. Enter **true** in the description field. 13. Click Save.	
14. Your destination is created. 15. Refresh your browser so SAP Web IDE reloads.	

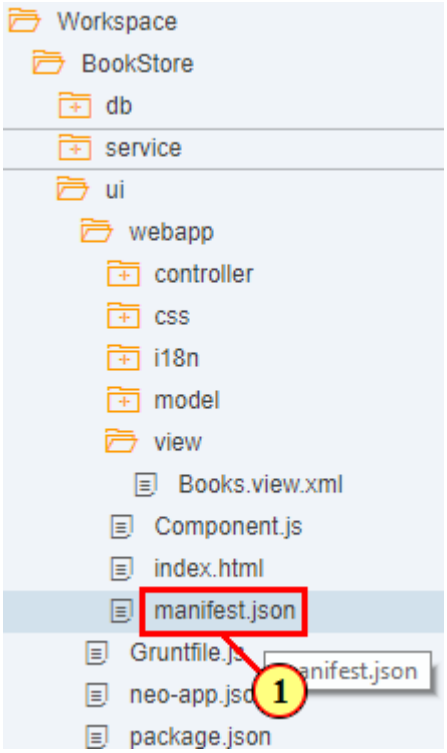
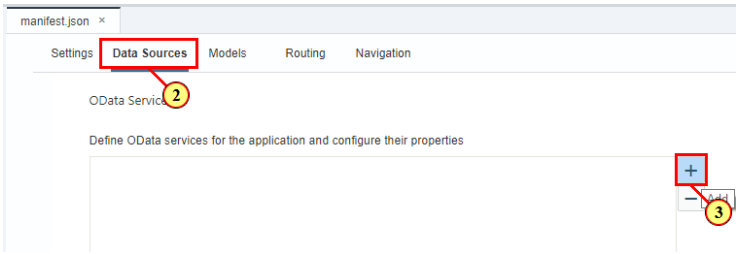
3.7 Configure the OData V4 Model and Preview the Application

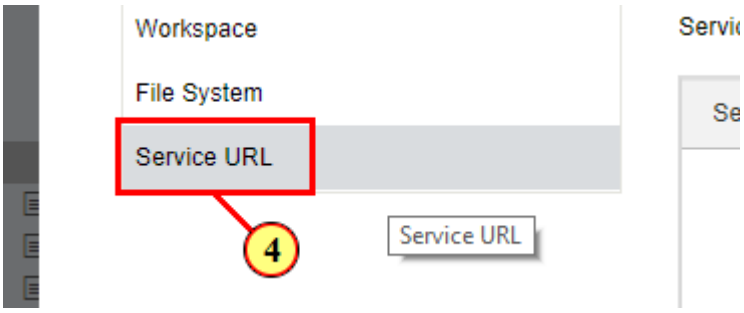
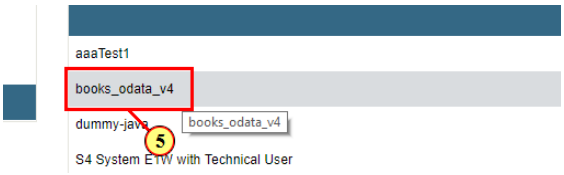
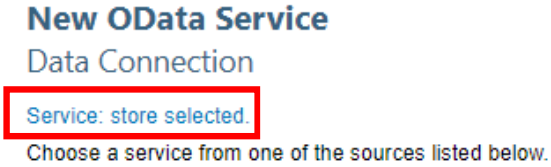
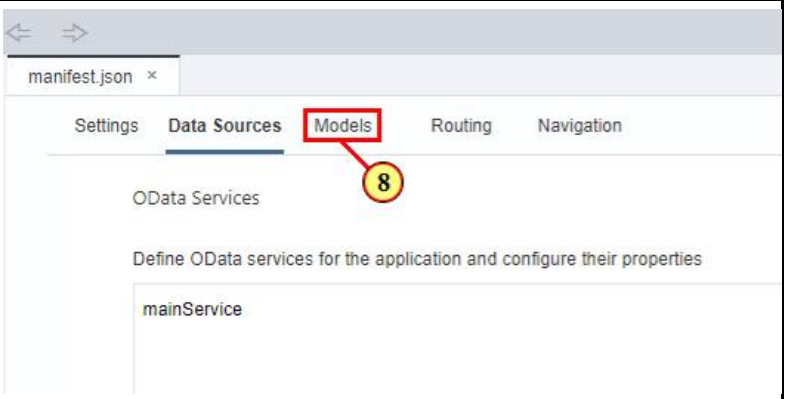
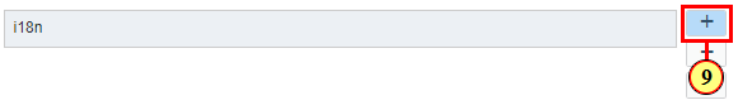
In this step, you will configure the data source and the way in which it connects to the new destination you created.

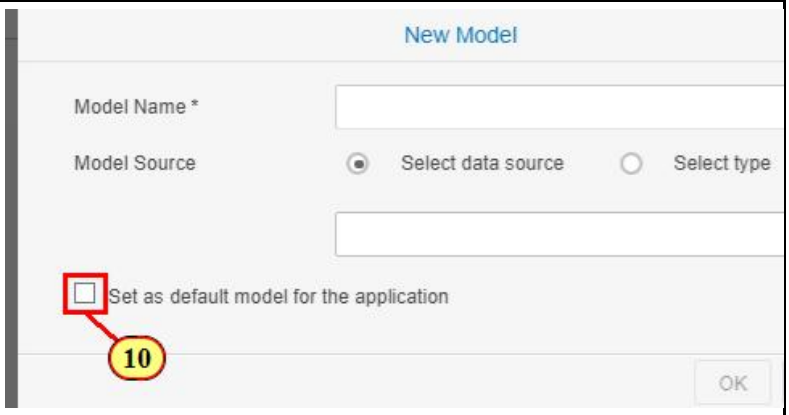
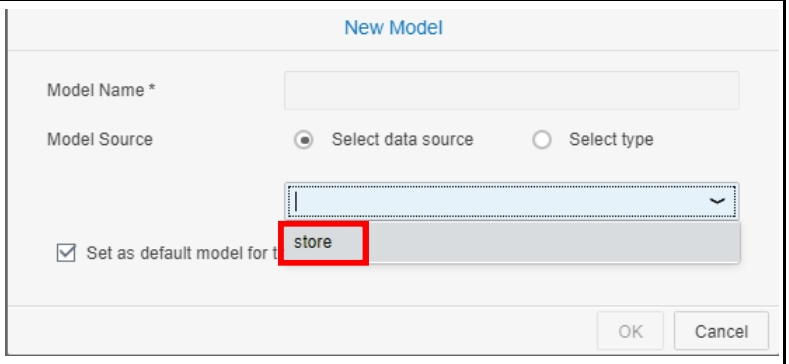
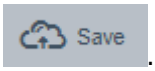
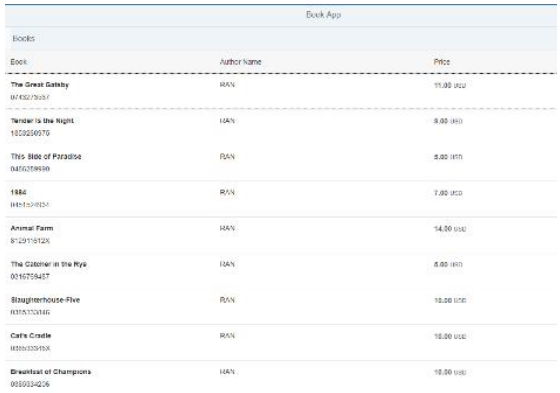
You will then create an OData model that consumes this data source and which is used by the application.

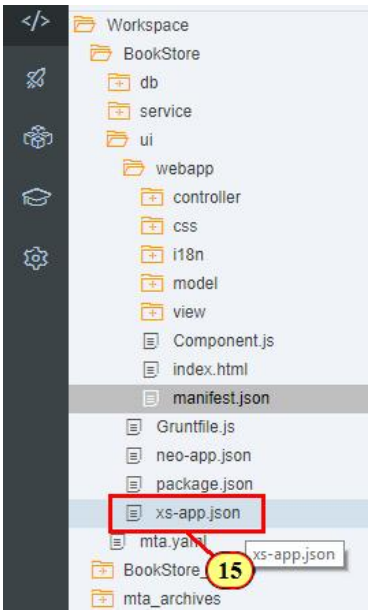
Once this is done, you will be able to preview the application and see the data.

Lastly, you will make some changes to the files to ensure the app runs smoothly in SAP Cloud Foundry.

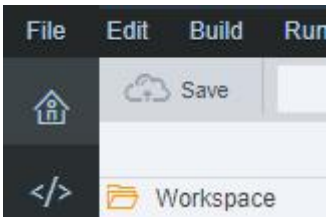
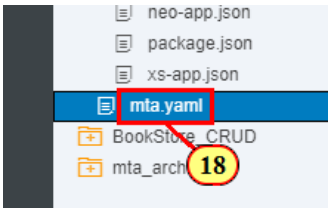
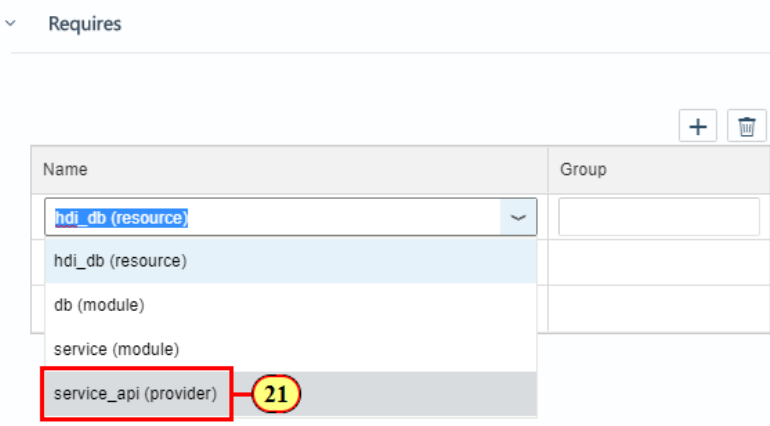
Explanation	Screenshot
1. Define a data source and connect it to the destination. Go to **BookStore > ui > webapp**, and click manifest.json.	
2. Open the **Data Sources** tab. 3. Click +.	

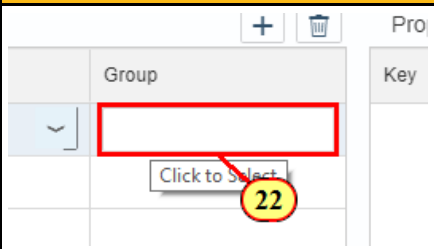
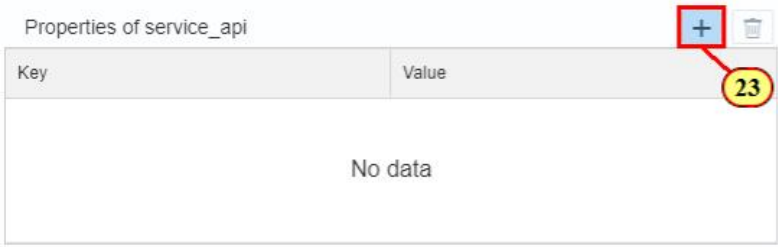
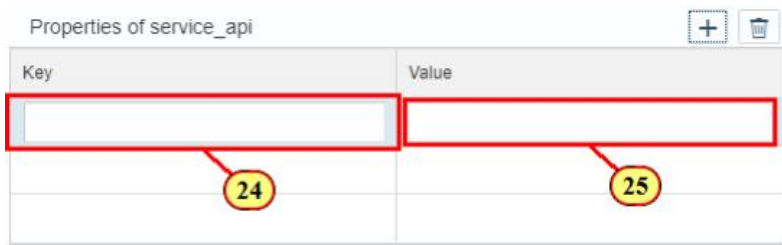
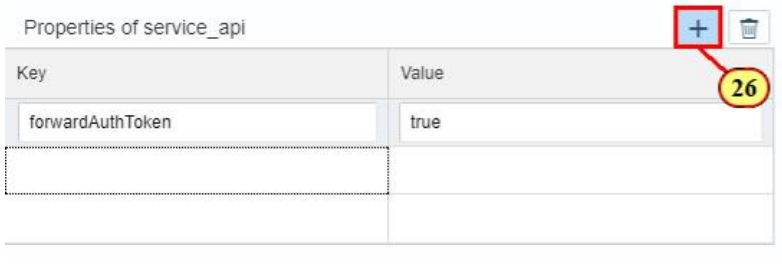
Explanation	Screenshot
4. Click Service URL.	
5. From the dropdown list, select **books_odata_v4**.	
6. In the **Service URL** field, enter `/odata/v4/store/` Click Test.	
A message is displayed under the Data Connection title. 7. Click **Next,** and then click **Finish**.	
8. Define the OData model. In the **manifest.json** file, open the **Models** tab of the graphical editor.	
9. Click +.	

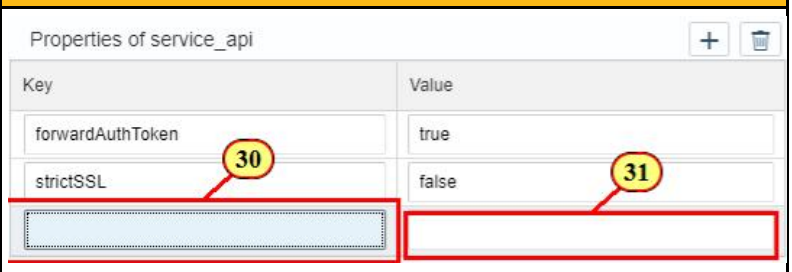
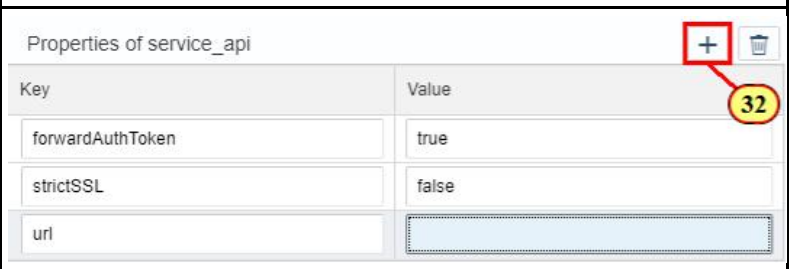
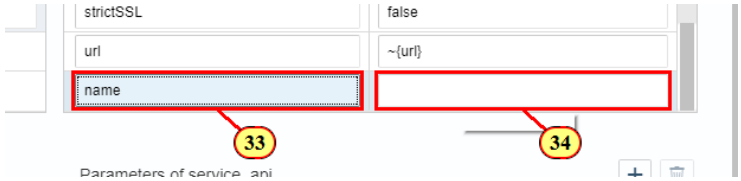
Explanation	Screenshot
10. Select the checkbox.	
11. From the data source dropdown list, select **store**, and click **OK**.	
12. Click  . 13. Click  .	
14. Your application is displayed.	

Explanation	Screenshot
Now you will update the **xs-app.json** and **MTA.yaml** so the app can run smoothly in SAP Cloud Foundry. 15. Update the **xs-app** file to handle service routes. Go to **BookStore > ui** , and click xs-app.json .	
16. Add the following route to the routes section:	

```
, {
            "source": "/books_odata_v4/(.*)$",
            "authenticationType": "none",
            "destination": "service_api",
            "csrfProtection": false,
            "target": "$1"
}
```

Explanation	Screenshot
17. Click Save.	
18. Under **Bookstore**, double-click mta.yaml.	
19. Click ui html5 to update the ui module so that it can consume the OData service from the Java module.	
20. Click + in the **Requires** section to add new entry.	
21. From the dropdown list, select service_api (provider).	

Explanation	Screenshot
22. In the **Group** field, enter **destinations**.	
23. In the **Properties of service_api** section, click + .	
24. Enter **forwardAuthToken** in the **Key** field. 25. Enter **true** in the **Value** field.	
26. Click + to create another property.	
27. Enter **strictSSL** in the **Key** field. 28. Enter **false** in the **Value** field.	
29. Click + to create another property.	

Explanation	Screenshot
30. Enter **url** in the **Key** field. 31. Enter **~{url}** in the **Value** field.	
32. Click + to create another property.	
33. Enter **name** in the **Key** field. 34. Enter **service_api** in the **Value** field. 35. Click Save.	

Summary

You have completed the exercise!

You are now able to:

- Create a Cloud Foundry project.
- Create the SAP HANA schemas for the project.
- Add data to the SAP HANA schemas.
- Create an OData V4 service.
- Create an SAPUI5 application together with a Neo destination.

4 CONNECT THE APPLICATION TO GIT

Overview

Estimated time: 10 minutes

Objective

In the following exercise, you will learn how to create your own git repository in SAP Cloud Platform.

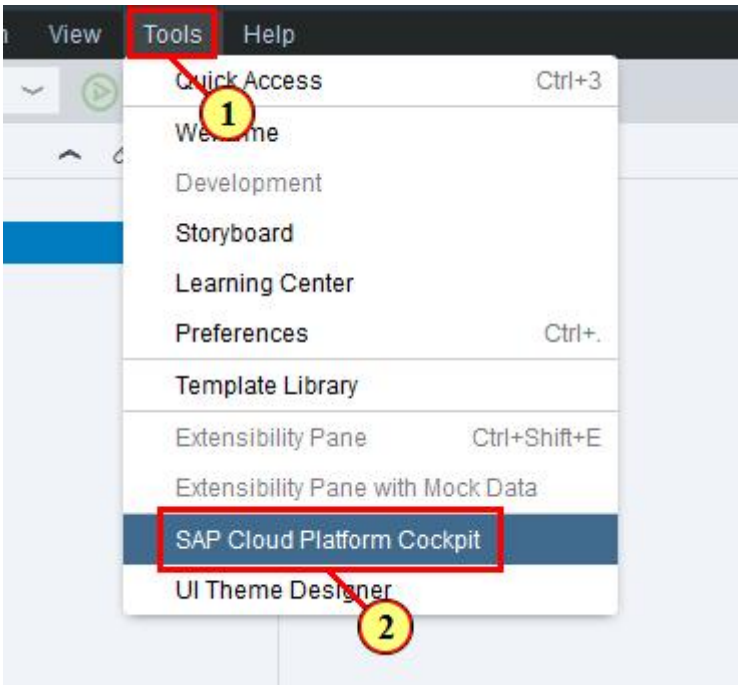
You will then connect the BookStore application we created to the git source control system in SAP Web IDE.

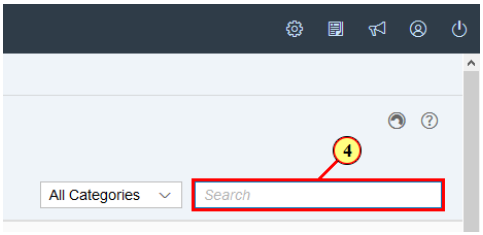
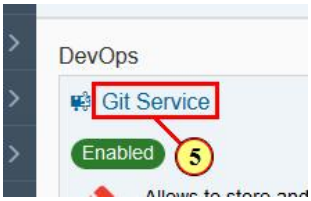
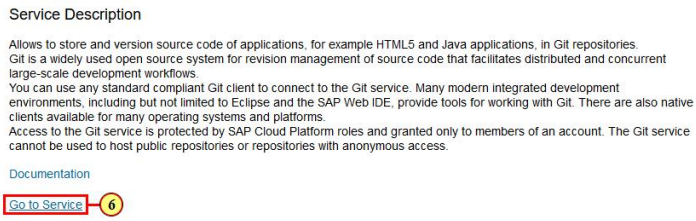
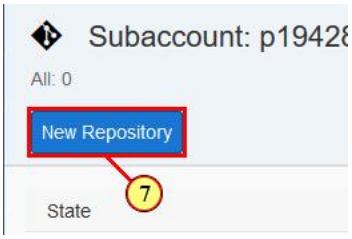
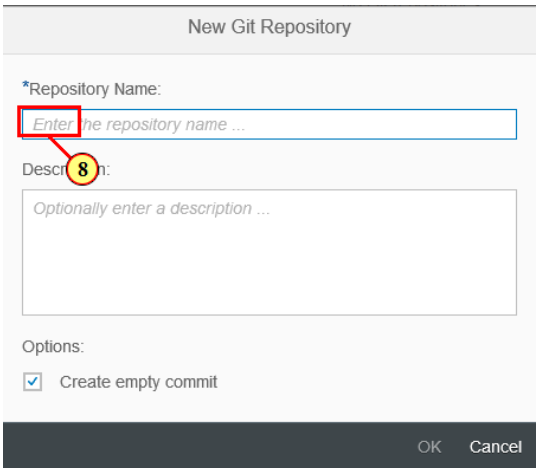
Exercise Description

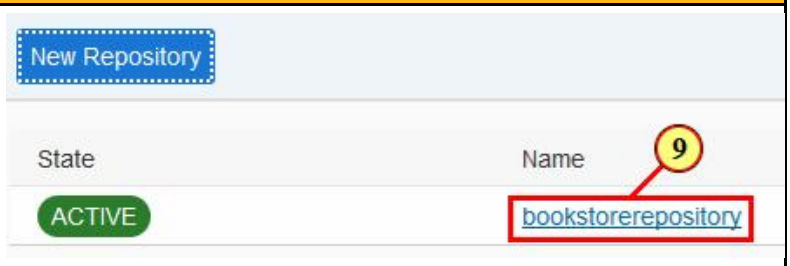
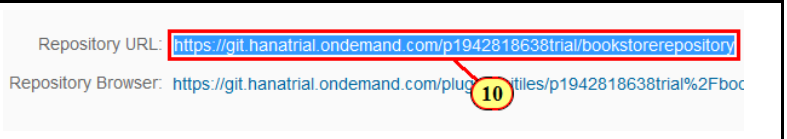
- Create a git repository in SAP Cloud Platform.
- Initialize a local repository in SAP Web IDE and connect to the remote repository

4.1 Create a Git Repository in SAP Cloud Platform

In this step, you will learn how to use the git service located in the SAP Cloud Platform cockpit.

Explanation	Screenshot
1. Open the **Tools** menu. 2. Click SAP Cloud Platform Cockpit.	

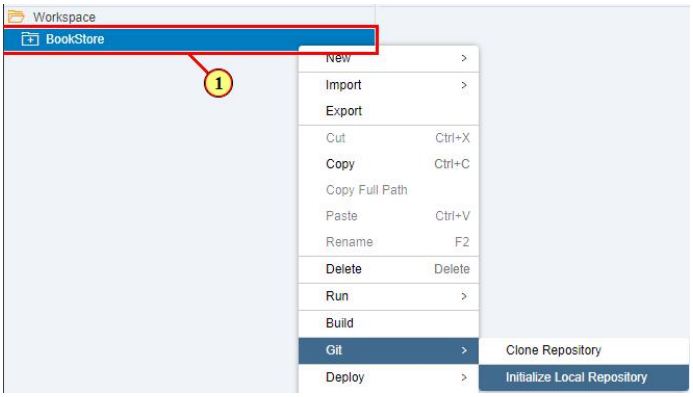
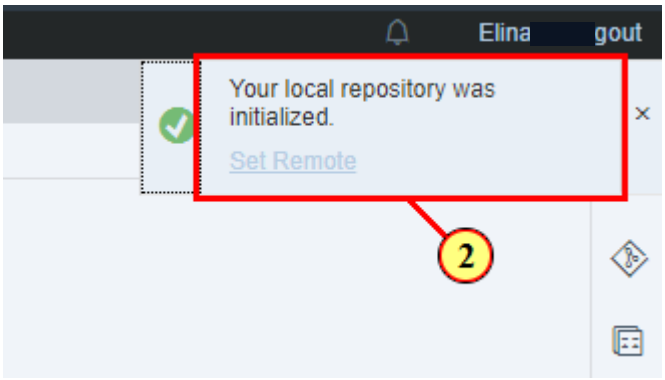
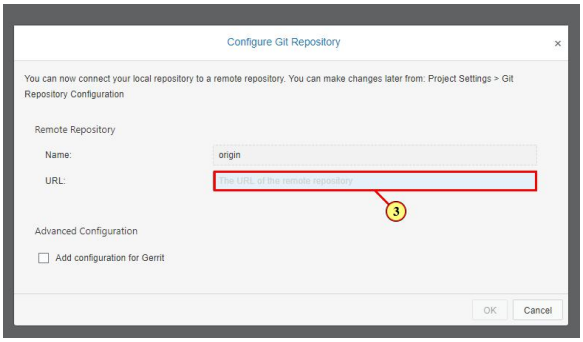
Explanation	Screenshot
3. Click  .	
4. Search for **git.**	
5. Select the Git Service tile.	
6. Click Go to Service .	
7. Click New Repository.	
8. Enter **bookstorerepository** as the repository name and click **OK**. A new repository is created to which you can connect any application.	

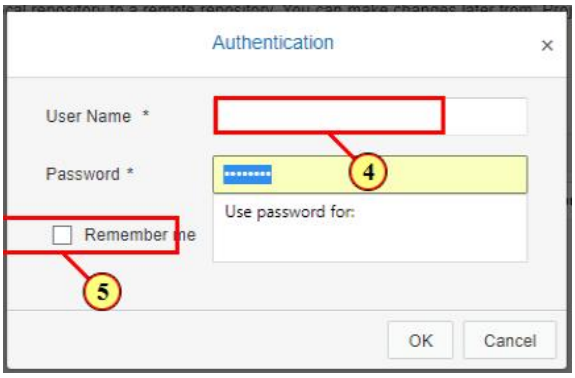
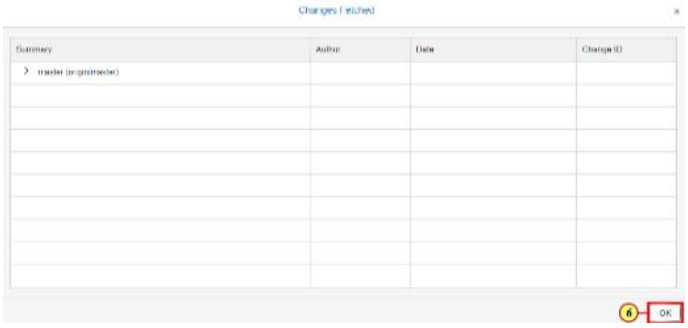
Explanation	Screenshot
9. Click bookstorerepository to see the repository details.	
10. Copy the Repository URL. You will need this URL in the next step.	

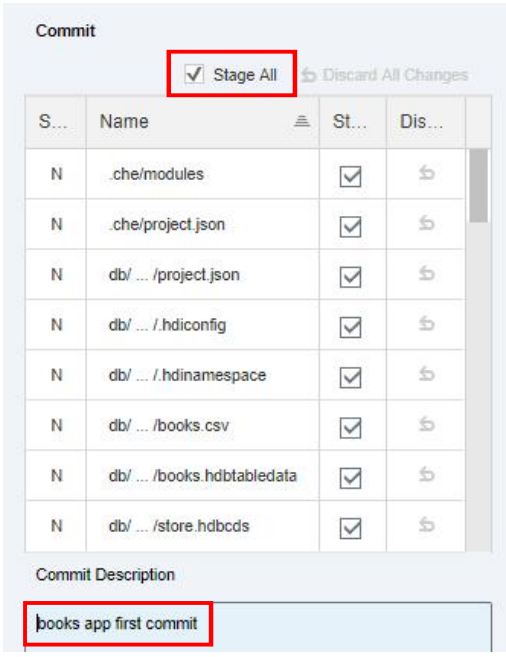
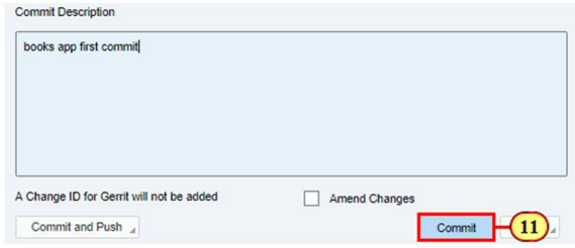
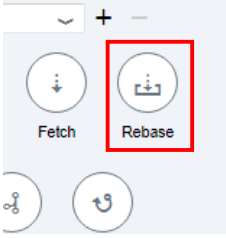
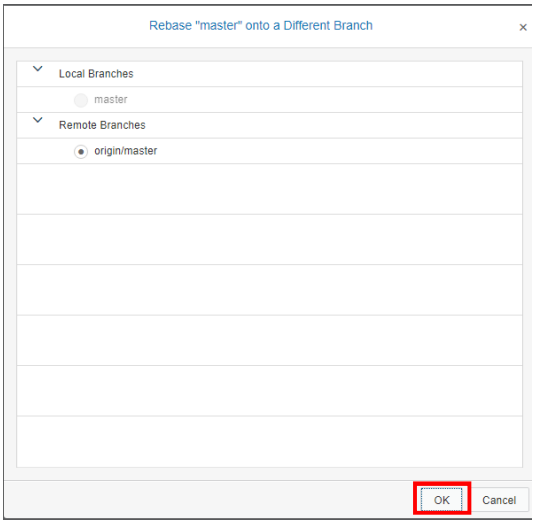
4.2 Init Local Repository in SAP Web IDE and Connect to the Remote Repository

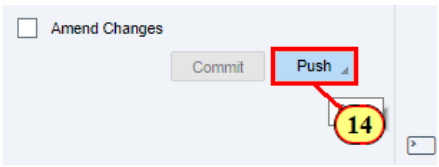
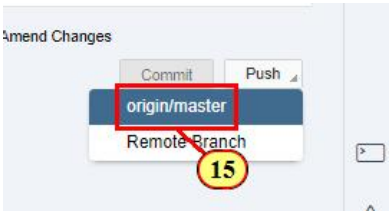
The local repository contains all the files and their commit history, thus giving you all the features of the distributed version control.

In order to share your code with others, you will need to connect the local repository to a remote repository.

Explanation	Screenshot
1. Right-click the **BookStore** project and select **Git > Initialize Local Repository**. Your local repository is created. You can now work with the git features for your project but you cannot share this with others.	
2. Click **Set Remote** to connect to the git repository we created in the previous step.	
3. Enter the Git Repository URL you saved on the previous step, and click **OK**.	

Explanation	Screenshot
4. Enter your SAP Cloud Platform credentials. 5. Select the **Remember me** checkbox and click **OK**.	
6. In the **Changes Fetched** dialog, click **OK**.	
7. Click  in the right-side pane to open the Git pane.	
8. Click **Fetch** to bring all changes and the existing branches from the remote repository, and then click **OK**.	

Explanation	Screenshot
9. Make sure the **Stage All** checkbox is selected to include all your files in the first commit. 10. Enter a description for the commit. For example, **books app first commit**.	
11. Click Commit.	
12. Click **Rebase** to include the changes.	
13. Make sure the **origin/master** branch is selected. The changes introduced to this remote branch will be integrated onto the local master branch that you are currently working on. Click **OK** to confirm the rebase.	

Explanation	Screenshot
14. Click **Push** to push the first commit to the remote repository.	
15. Click origin/master to specify the branch within the remote repository the changes will be pushed into.	
16. Back in the SAP Cloud Platform cockpit, click the **Repository Browser** URL to see the created git tree.	

Summary

You have completed the exercise!

You are now able to:

- Create Git repositories in SAP Cloud Platform.
- Connect your project to any git server.
- Use git as your source control system.

5 DEPLOY TO CLOUD FOUNDRY ENVIRONMENT

Overview

Estimated time: 10 minutes

Objective

In the following exercise you will learn how to deploy your application to the Cloud Foundry environment from SAP Web IDE.

When you build your MTA project, an MTA archive (MTAR) is created containing deployable application modules.

For more information on MTAR files, see

https://help.sap.com/viewer/58746c584026430a890170ac4d87d03b/Cloud/en-US

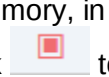
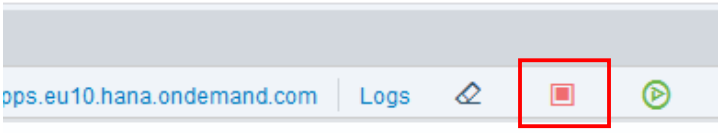
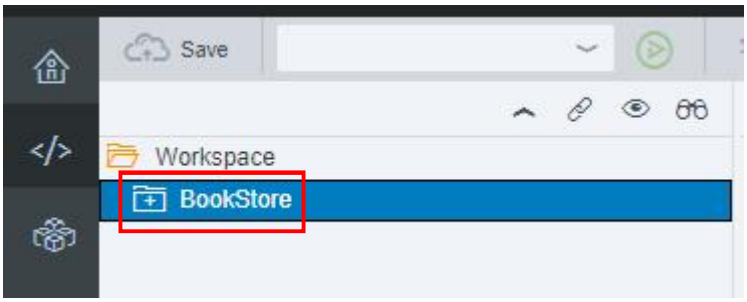
Exercise Description

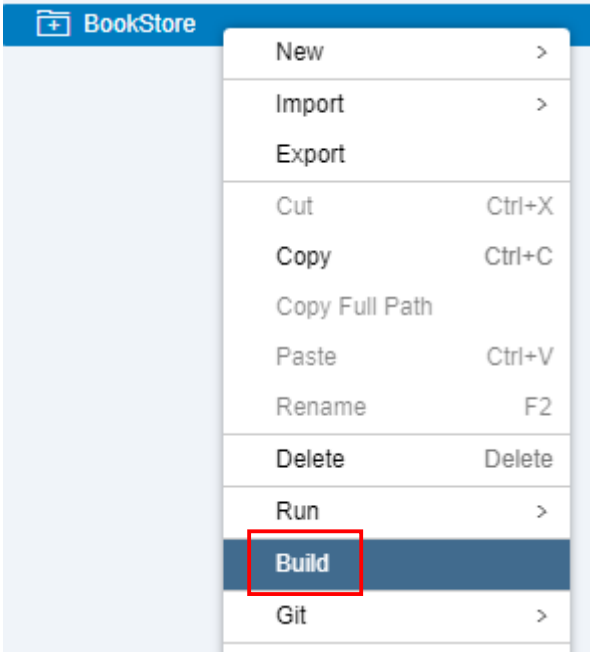
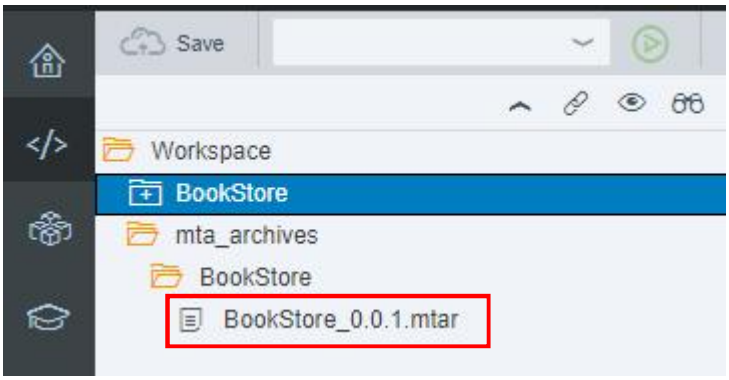
- Build a multi-target application.
- Deploy the application to Cloud Foundry.

5.1 Build the Application

In this step, you will build the MTA application you created. The outcome of the build is an MTAR file that contains the entire application.

This MTAR file is what will be deployed in the next step.

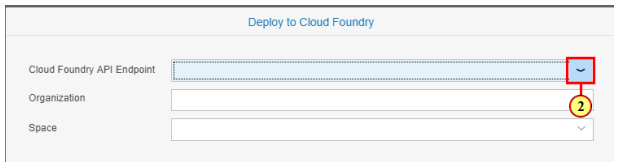
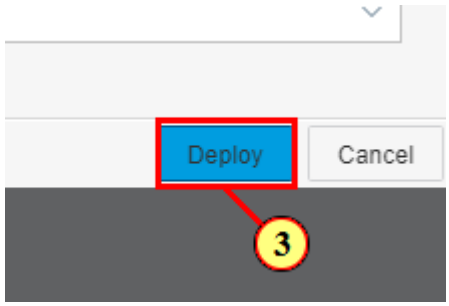
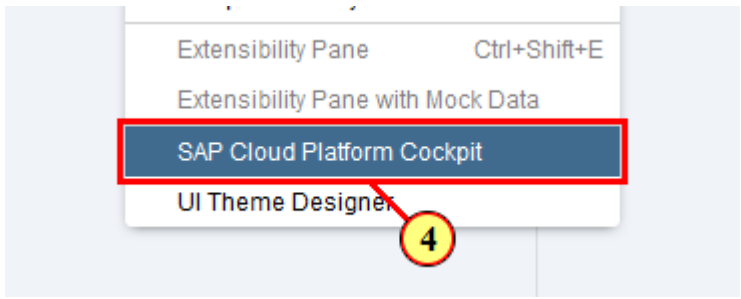
Explanation	Screenshot
Before you can deploy, you must free some memory in your Cloud Foundry space. This is due to the fact that you are working in the Trial landscape which has limited resources. 1. To free memory, in the Run console, click  to stop the Java application.	
2. Right-click the **BookStore** project.	

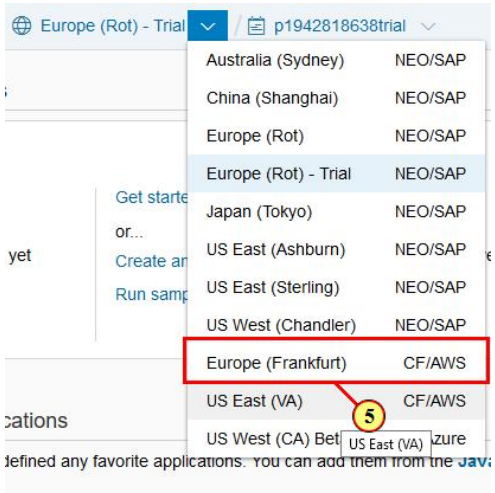
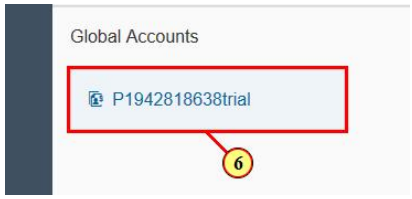
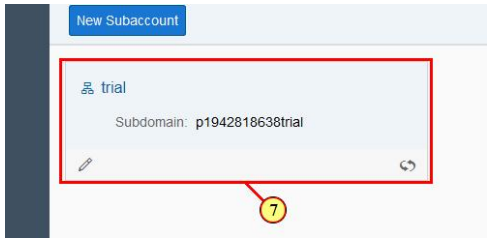
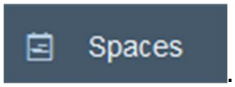
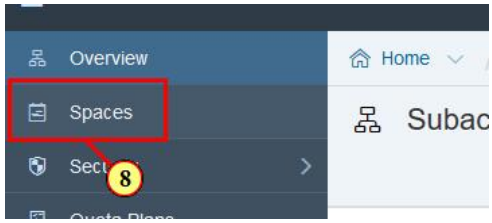
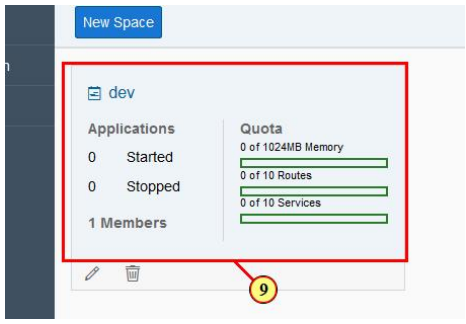
Explanation	Screenshot
3. Click **Build**.	
4. Once the build is complete, go to **BookStore > mta_archives > BookStore** and make sure you see the MTAR file that was created.	

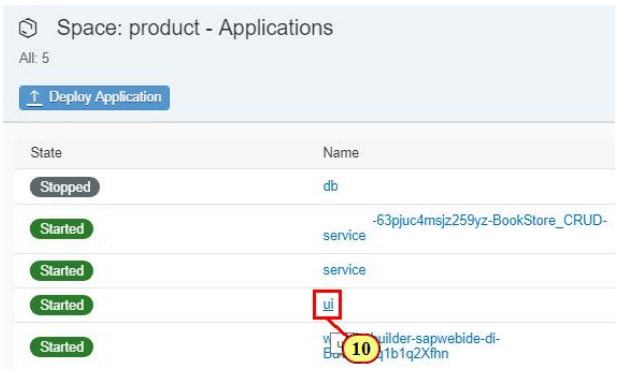
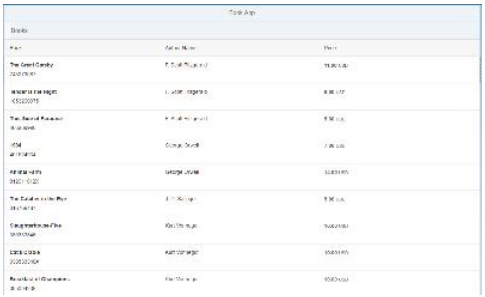
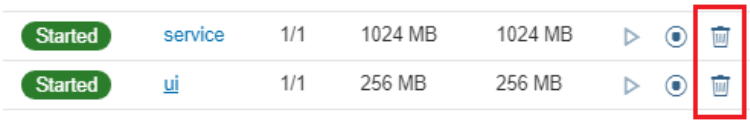
5.2 Deploy the Application

Once you have created and built your application, you can deploy it.

In this step, you will deploy the MTAR file you created to your Cloud Foundry space.

Explanation	Screenshot
1. Go to **BookStore > mta_archives > BookStore** and right-click **BookStore_0.0.1.mtar**. Select **Deploy > Deploy to Cloud Foundry**.	
2. From the **Cloud Foundry API Endpoint** dropdown list, select **cf.eu10.hana.ondemand.com** as the endpoint.	
3. Click **Deploy**. The application is deployed to Cloud Foundry.	
4. Once the app is deployed, click **Tools > SAP Cloud Platform Cockpit**.	

Explanation	Screenshot
5. From the **Europe (Rot) - Trial** menu, select **Europe (Frankfurt).**	
6. Click on your global account.	
7. Click on your subaccount.	
8. Click  .	
9. Click on the **dev** tile to see your space.	

Explanation	Screenshot
10. You can see all the 3 modules of the app in the list. Click ui .	
11. Click the link under **Application Routes** . The hostname of the URL is composed of your organization-space-app name.	
12. Your application is now deployed to the Cloud Foundry environment.	
13. In order to continue with the rest of the exercise, you need to stop the application you just deploied to free memory once again. Click on stop button next to the **UI** and **service** applications.	

Summary

You have completed the exercise!

You are now able to:

- Build a multi-target application.
- Deploy the application to Cloud Foundry.

6 ADD CREATE AND DELETE CAPABLITIES

Overview

Estimated time: 60 minutes

Objective

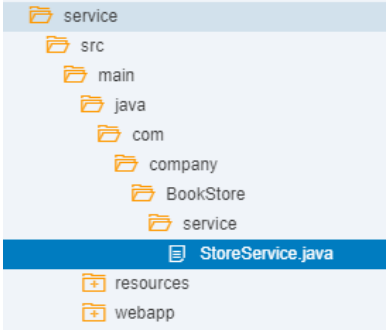
In this section, we will enrich the OData service capabilities and use them to enhance the UI.

Exercise Description

- Add the Create and Delete capabilities to the OData V4 service.
- Add a UI view to create a new book.

6.1 Update the Service

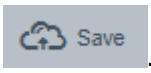
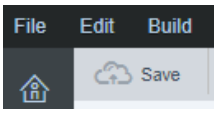
In this step, you will update the OData service in order to enable it with Create and Delete capabilities.

Explanation	Screenshot
1. Open the **service** module and expand the tree to the **StoreService.java** file. 2. Double-click the **StoreService.java** file to open it in the editor.	
3. Add the following packages to the file:	

```
import com.sap.cloud.sdk.service.prov.api.EntityData;
import com.sap.cloud.sdk.service.prov.api.operations.Query;
import com.sap.cloud.sdk.service.prov.api.operations.Read;
import com.sap.cloud.sdk.service.prov.api.request.QueryRequest;
import com.sap.cloud.sdk.service.prov.api.request.ReadRequest;
import com.sap.cloud.sdk.service.prov.api.response.QueryResponse;
import com.sap.cloud.sdk.service.prov.api.response.ReadResponse;

import com.sap.cloud.sdk.service.prov.api.operations.Create;
import com.sap.cloud.sdk.service.prov.api.operations.Delete;
import com.sap.cloud.sdk.service.prov.api.request.CreateRequest;
import com.sap.cloud.sdk.service.prov.api.request.DeleteRequest;
import com.sap.cloud.sdk.service.prov.api.response.CreateResponse;
import com.sap.cloud.sdk.service.prov.api.response.DeleteResponse;

import com.sap.cloud.sdk.hana.connectivity.cds.CDSException;
```

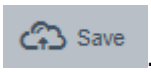
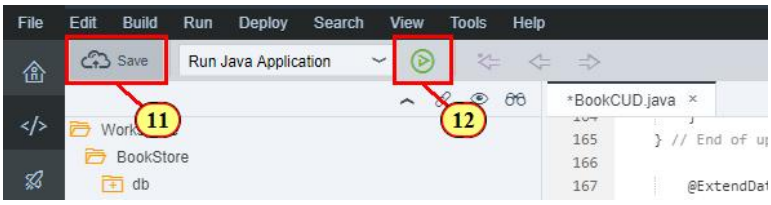
Explanation	Screenshot
4. Implement the **create** capability method by adding the code below.	

```
      @Create(entity = "Book", serviceName = "store")
    public CreateResponse createSalesOrderLineItems(CreateRequest
createRequest) {
        CreateResponse createResponse =
CreateResponse.setSuccess().setData(createEntity(
createRequest)).response();
        return createResponse;
    }


private EntityData createEntity(CreateRequest createRequest)
    {
        CDSDataSourceHandler dsHandler =
DataSourceHandlerFactory.getInstance().getCDSHandler(getConnection(),
createRequest.getEntityMetadata().getNamespace());
        EntityData ed = null;
        try{
            ed = dsHandler.executeInsert(createRequest.getData(), true);
        } catch (CDSException e){
            logger.error("Eexception while creating an entity in CDS: " +
e.getMessage());
        }
        return ed;
    }
```

5. Click  .	
6. Implement the **delete** cabaility by adding the code below:	

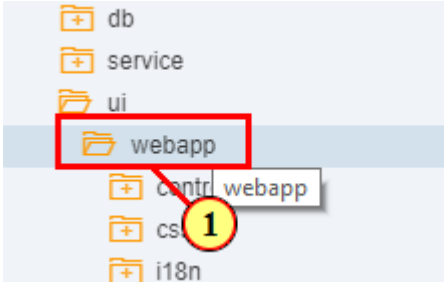
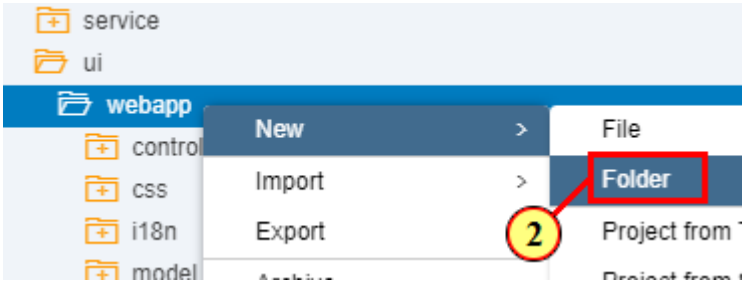
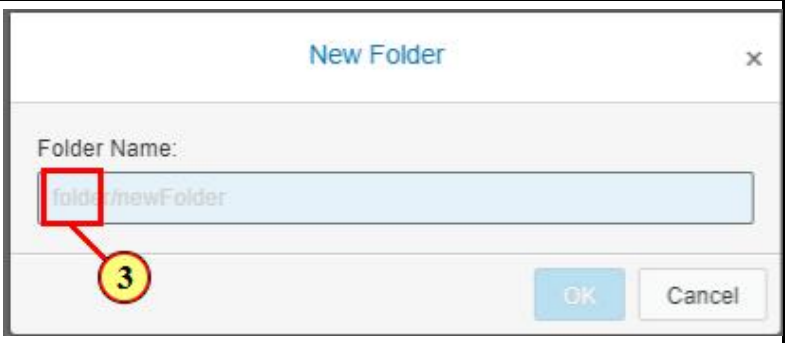
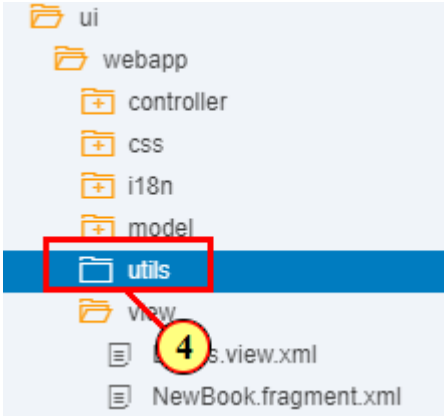
Explanation	Screenshot

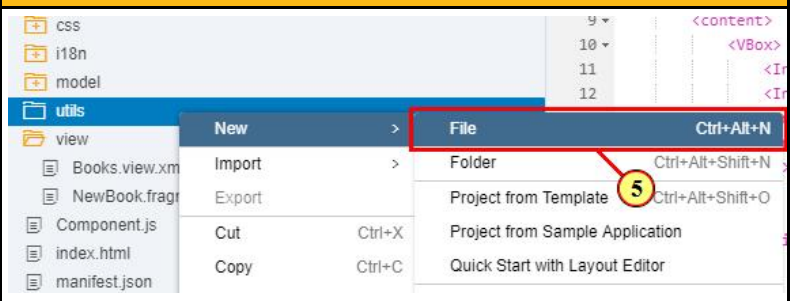
`@Delete(entity = "Book", serviceName ="store")`
`  public DeleteResponse deleteSalesOrder(DeleteRequest deleteRequest) {`
`  deleteEntity(deleteRequest);`
`  DeleteResponse deleteResponse = DeleteResponse.setSuccess().response();`
`  return deleteResponse;`
`  }`
 	
`private void deleteEntity(DeleteRequest deleteRequest){`
`  CDSDataSourceHandler dsHandler = DataSourceHandlerFactory.getInstance().getCDSHandler(getConnection(), deleteRequest.getEntityMetadata().getNamespace());`
`  try{`
`dsHandler.executeDelete(deleteRequest.getEntityMetadata().getName(), deleteRequest.getKeys());`
`  } catch (CDSException e){`
`  logger.error("Eexception while deleting an entity in CDS: " + e.getMessage());`
`  }`
`  }`	

7. Click  . 8. Click  (Run) to update the running OData service.	

6.2 Enhance the Application with Create and Delete Capabilities

In this step, you will edit the user interface by adding the options to create and delete book entires in the application.

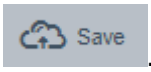
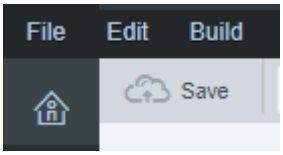
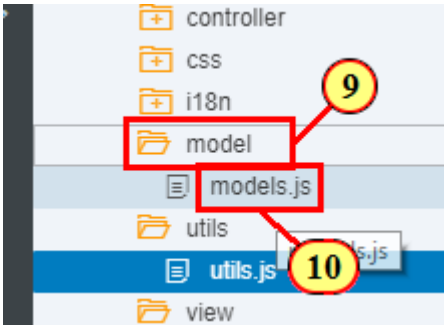
Explanation	Screenshot
1. You will now add a **utilities** file that contains functions used by the controller. Go to **Bookstore > ui** and right-click the **webapp** folder.	
2. Click **New > Folder**.	
3. Enter **utils** as the folder name and click **OK**.	
4. Go to **Bookstore > ui** > **webapp** , and right-click the **utils** folder.	

Explanation	Screenshot
5. Click **New > File**.	
6. Name the file **utils.js** and click **OK**.	
7. In the **utils.js** file, copy the following code snippets to add Utility methods to generate the book ID, get the default currency code, and display an info message.	

```
jQuery.sap.require('jquery.sap.resources');
sap.ui.define([], function() {
    "use strict";

    var Utils = {
        /**
         * function for generting random number between min and max
         * */
        generateRandomNumber: function(min, max) {
            return Math.random() * (max - min) + min;
        },

        getDefaultCurrencyCode: function() {
            return "EUR";
        },

        showInfoMessage: function(sMessage) {
            sap.m.MessageToast.show(sMessage, {
                duration: 3000
            });
        },

        getDefaultResourceBundle: function() {
            var oBundle = jQuery.sap.resources({
                url: "i18n/i18n.properties",
                locale: 'en_GB'
            });
            return oBundle;
        }
    };

    return Utils;
});
```

Explanation	Screenshot

<table><tr><td><pre>jQuery.sap.require("jquery.sap.resources");
sap.ui.define([], function() {
"use strict";
 	
var Utils = {
 	
generateRandomNumber: function(min, max) {
return Math.random() * (max - min) + min;
},
 	
getDefaultCurrencyCode: function() {
return "EUR";
},
 	
showInfoMessage: function(sMessage) {
sap.m.MessageToast.show(sMessage, {
duration: 3000
});
},
 	
getDefaultResourceBundle: function() {
var oBundle = jQuery.sap.resources({
url: "i18n/i18n.properties",
locale: "en_GB"
});
return oBundle;
}
};
 	
return Utils;
});</pre></td></tr></table>	

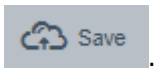
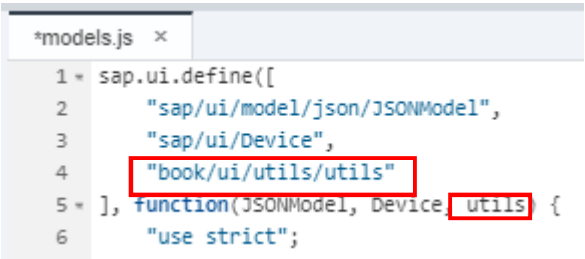
8. Click  .	
9. Create dialog models to support the "Add Book" dialog. Go to **Bookstore > ui** > **webapp** , and open the **model** folder. 10. Double-click  .	

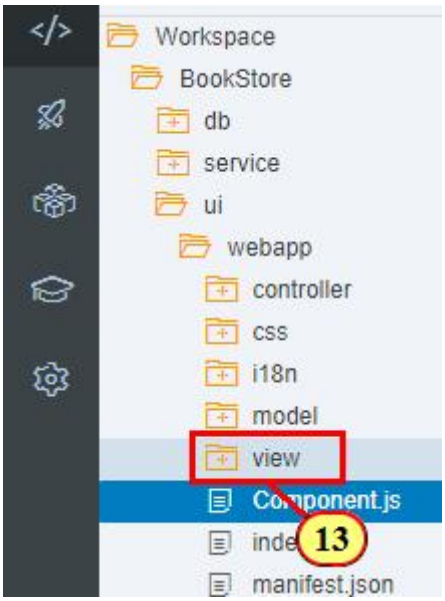
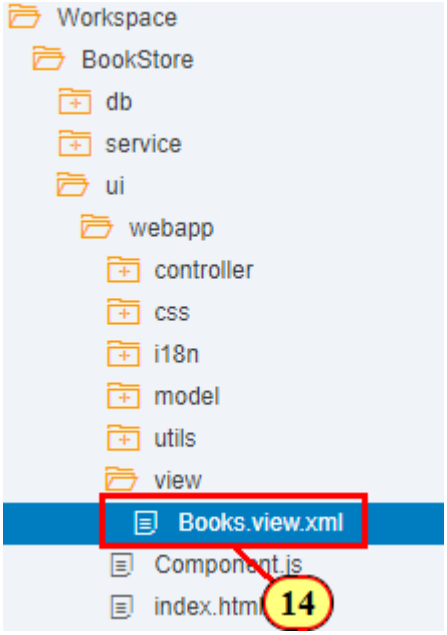
Explanation	Screenshot

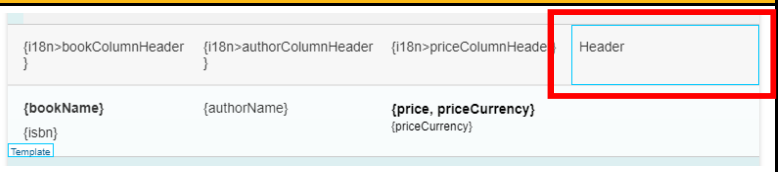
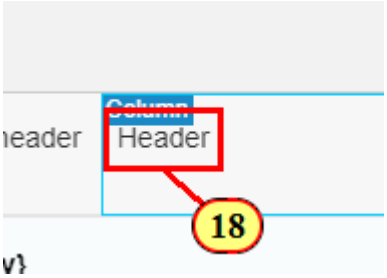
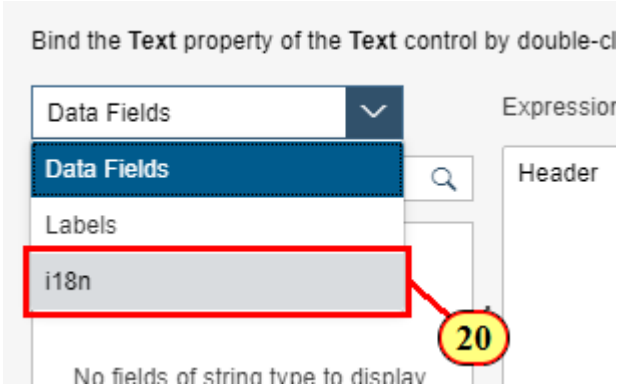
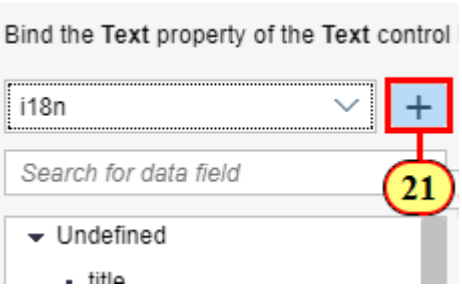
11. In the **models.js** file, go to the **return** object.
 	
In the **return** object, add the **createNewDialogEmptyModel** and **createNewBookPlaceholders Model** methods provided below.
 	
The former is responsible for creating a model with the book details and the latter is responsible for creating a model with the default values presented in the dialog.
	*models.js
	sap.ui.define([
	"sap/ui/model/json/JSONModel",
	"sap/ui/Device",
	"book/ui/utils/utils"
	], function(JSONModel, Device, utils) {
	"use strict";
	
	return {
	
	createDeviceModel: function() {
	var oModel = new JSONModel(Device);
	oModel.setDefaultBindingMode("OneWay");
	return oModel;
	
	},
	createNewBookDialogEmptyModel: function() {
	var oDialogModel = new JSONModel({
	data: {
	"bookId": parseInt(utils.generateRandomNumber(100, 9999)),
	"bookName": "",
	"isbn": "",
	"price": undefined,
	"priceCurrency": utils.getDefaultCurrencyCode(),
	"authorName": ""
	},
	"canCreate": false
	});
	
	return oDialogModel;
	},
	
	createNewNookPlaceholdersModel: function() {
	var oPlaceholdersModel = new JSONModel({
	"bookNamePL": "Book Name",
	"isbnPL": "ISBN",
	"pricePL": "Price",
	"authorNamePL": "Author Name"
	});
	
	// one way binding because placeholders cannot be changed from the UI
	oPlaceholdersModel.setDefaultBindingMode(sap.ui.model.BindingMode.OneWay);
	
	return oPlaceholdersModel;
	}
	
	};
	
	});

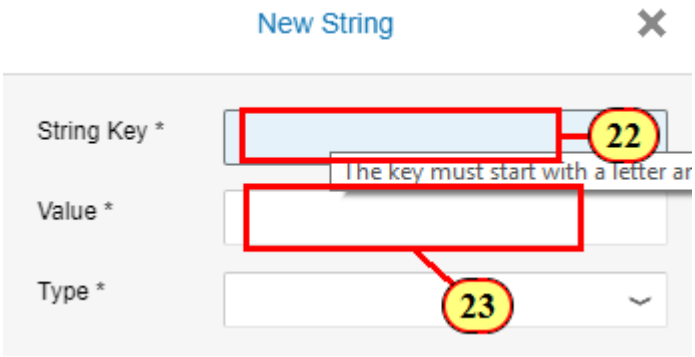
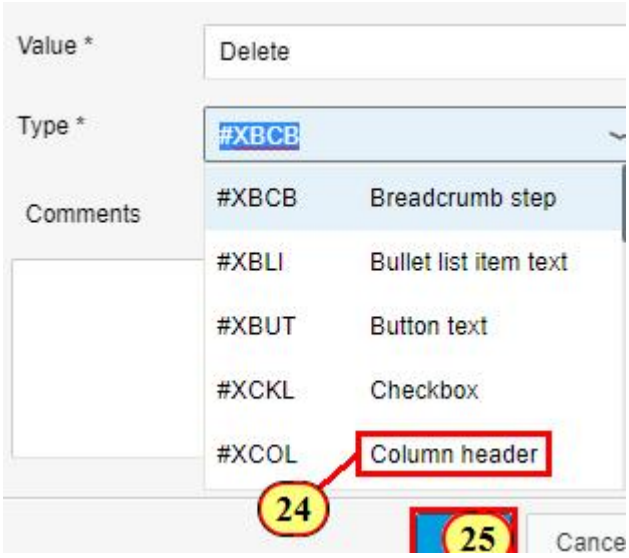
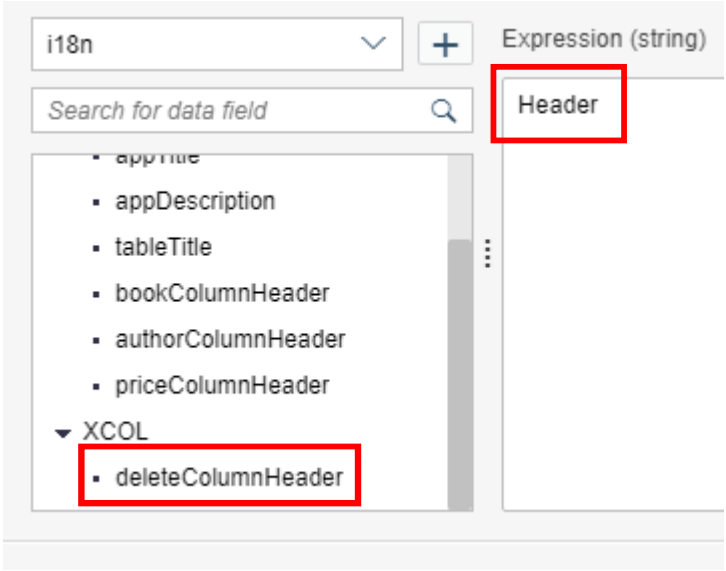
Explanation	Screenshot

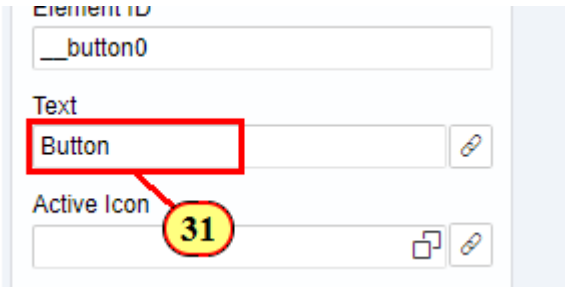
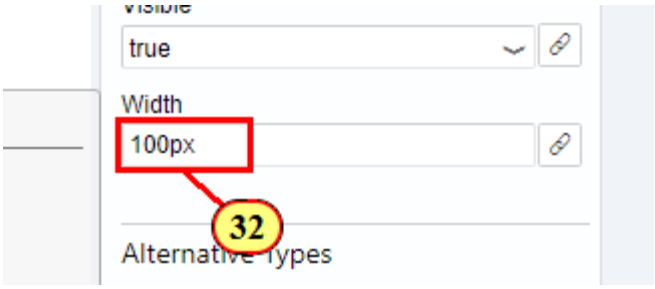
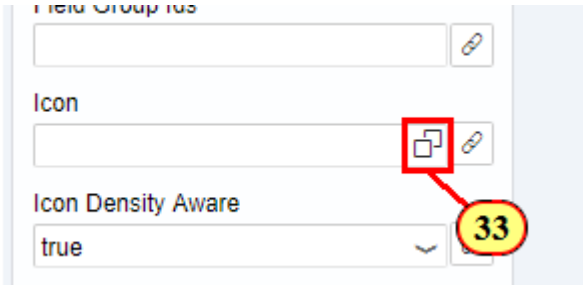
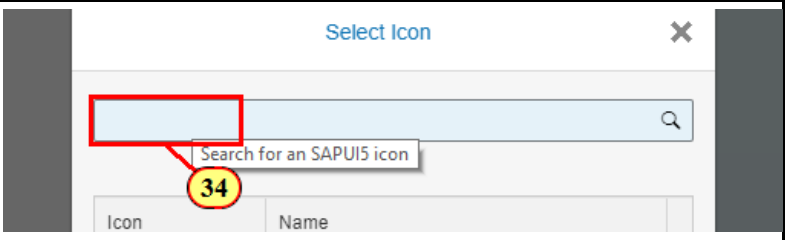
,
 	
`createNewBookDialogEmptyModel: function() {`
`var oDialogModel = new JSONModel({`
`data: {`
`"bookId": parseInt(utils.generateRandomNumber(100, 9999)),`
`"bookName": "",`
`"isbn": "",`
`"price": undefined,`
`"priceCurrency": utils.getDefaultCurrencyCode(),`
`"authorName": ""`
`},`
`"canCreate": false`
`});`
 	
`return oDialogModel;`
`},`
 	
`createNewNookPlaceholdersModel: function() {`
`var oPlaceholdersModel = new JSONModel({`
`"bookNamePL": "Book Name",`
`"isbnPL": "ISBN",`
`"pricePL": "Price",`
`"authorNamePL": "Author Name"`
`});`
 	
`// one way binding because placeholders cannot be changed from the UI`
 	
`oPlaceholdersModel.setDefaultBindingMode(sap.ui.model.BindingMode.OneWay);`
 	
`return oPlaceholdersModel;`
`}`	

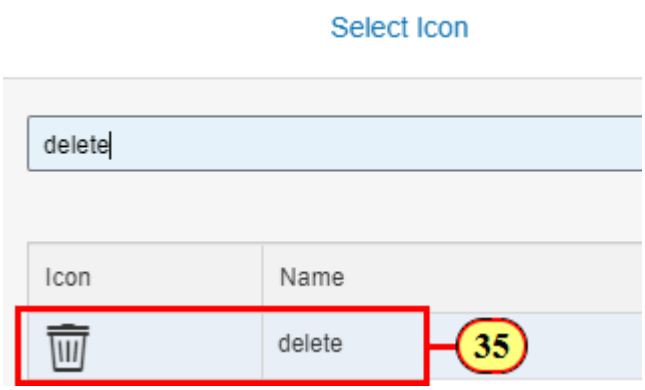
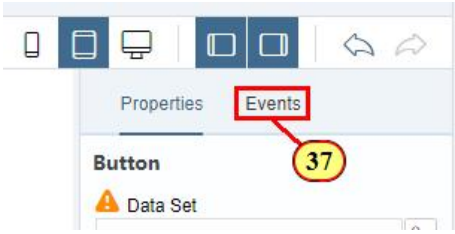
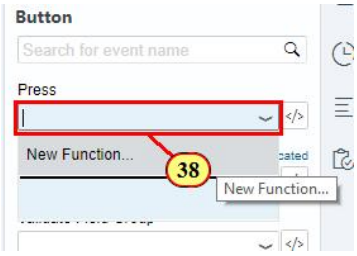
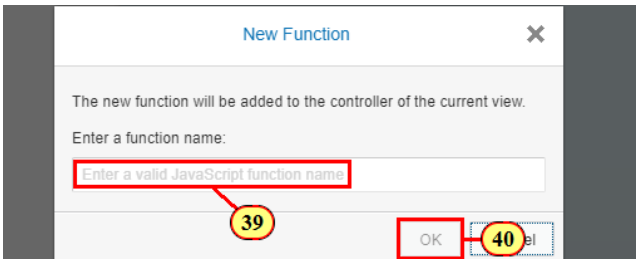
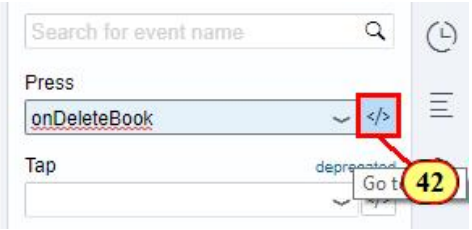
12. Add the **utils** file as the controller's required resource: `"book/ui/utils/utils"` Add **utils** as the controller's input parameter. Click  .	

Explanation	Screenshot
13. Go to **Bookstore > ui > webapp > view**.	
14. Right-click **Books.view.xml**.	
15. Click **Open With > Layout Editor**.	
16. In the **Controls** view, enter **column** in the search field.	

Explanation	Screenshot
17. Drag the Column control as the last element in the **Column** row of the table in the canvas.	
18. In the new **Column** control, click **Header**.	
19. In the **Properties** pane, click by the **Text** field to open the **Data Binding** dialog box.	
20. From the dropdown list, select **i18n**. The i18n model contains the ui strings for translation.	
21. Click to add a new translation entry.	

Explanation	Screenshot
22. In the **String Key** field, enter **deleteColumnHeader**. 23. In the **Value** field enter **Delete**.	
24. From the **Type** dropdown list, select Column header. 25. Click OK.	
26. In the **Expression** field, delete **Header**. 27. Double-click **deleteColumnHeader** to bind the expression. 28. Click OK.	

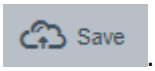
Explanation	Screenshot
29. Clear the **Controls** search field. 30. Drag the  control and drop it as the last element in the **Cells** row of the table in the canvas.	
31. In the **Properties** pane, delete **Button** from the **Text** field**.**	
32. Delete **100px** from the **Width** field.	
33. By the **Icon** field, click  to open the list of available icons.	
34. Enter **delete** in the search field.	

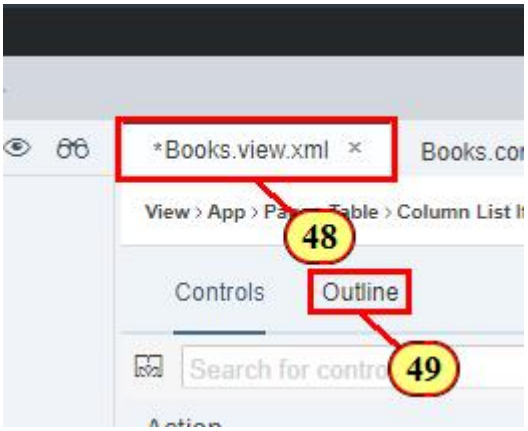
Explanation	Screenshot
35. Select the **delete** icon and click **OK**. 36. Click Save.	
37. Open the **Events** pane.	
38. You will now define the action to be triggered when the **Delete** button is pressed. From the **Press** dropdown list, select **New Function**.	
39. Enter **onDeleteBook** as the name of the new function to be added to the controller. 40. Click OK. 41. Click Save.	
42. By the **Press** field, click </>. The controller opens and you are directed to the location of the **onDeleteBook** method.	

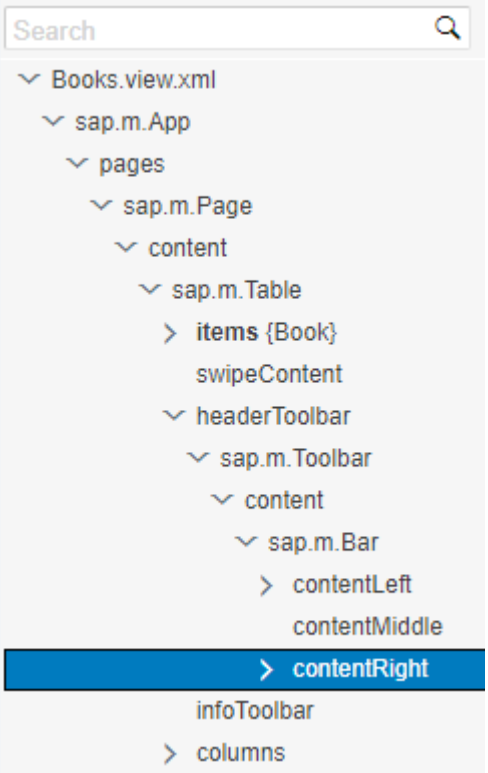
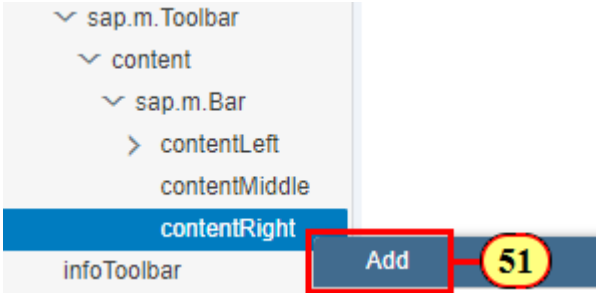
Explanation	Screenshot

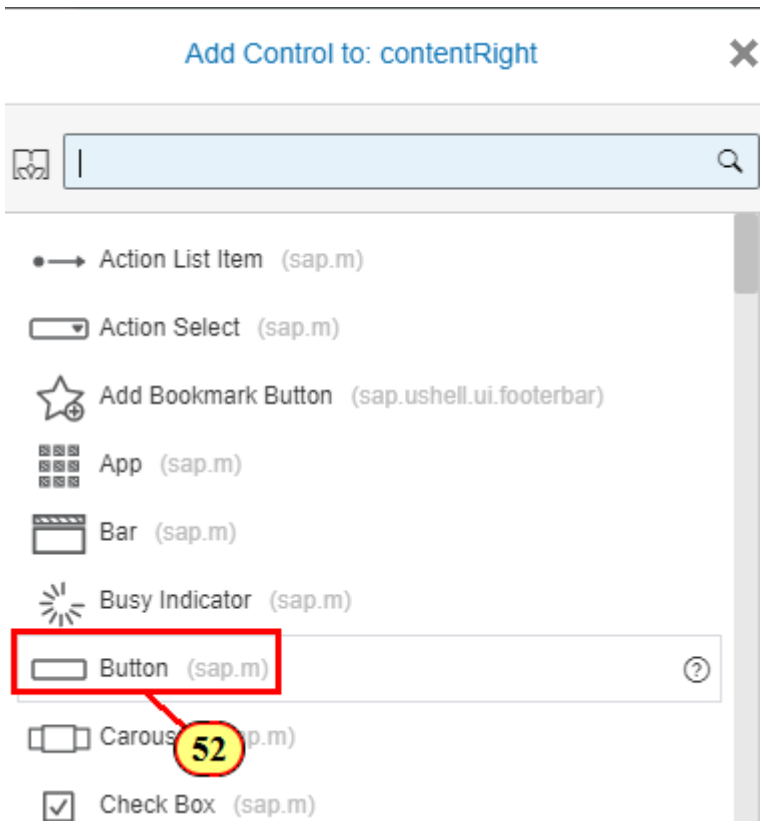
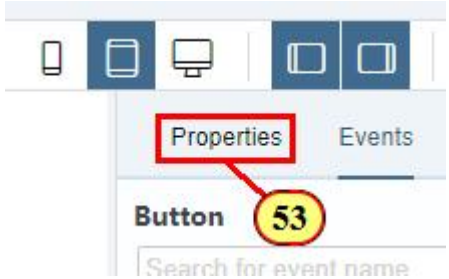
43. Add **oEvent** as the **onDeleteBook** method's input parameter.
 	
44. Add the following code in the created method:
	`onDeleteBook: function(oEvent) {`
	`  var oItemContext = oEvent.getSource().getBindingContext();`
	`  oItemContext.delete("$auto").then(function() {`
	`  utils.showInfoMessage(utils.getDefaultResourceBundle().ge`
	`  });`
	`},`

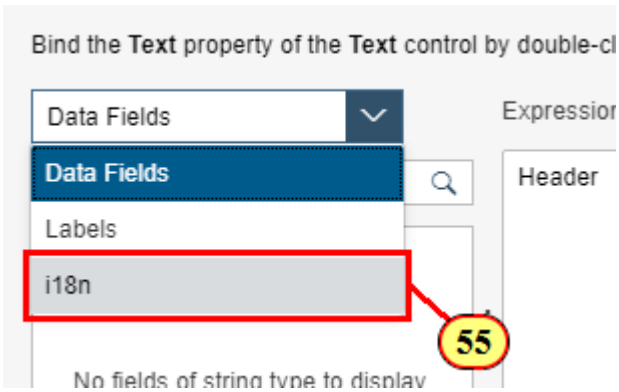
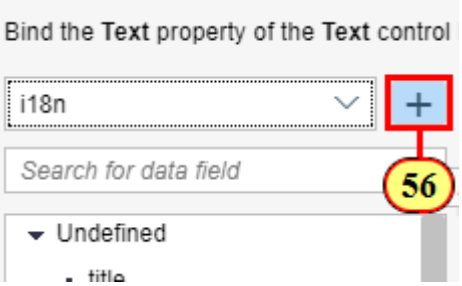
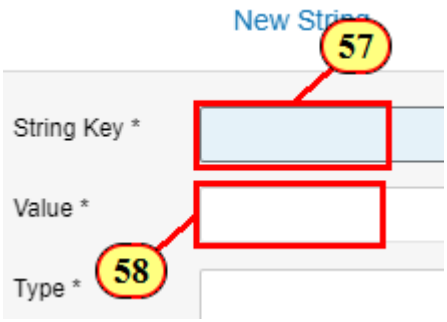
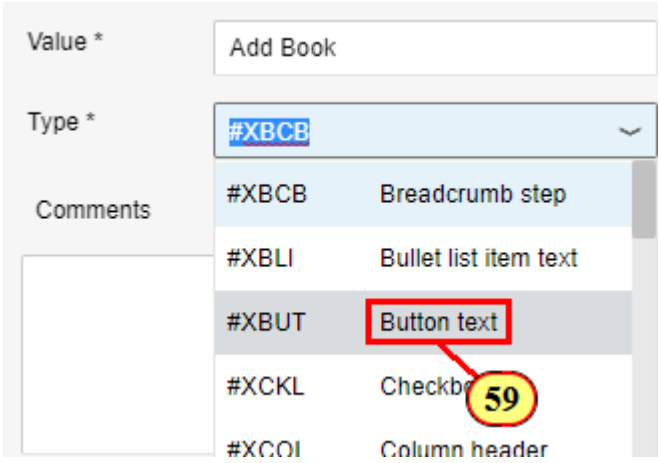
```
var oItemContext = oEvent.getSource().getBindingContext();
oItemContext.delete("$auto").then(function() {
      utils.showInfoMessage(utils.getDefaultResourceBundle().getText("book
DeletedMessage"));
});
```

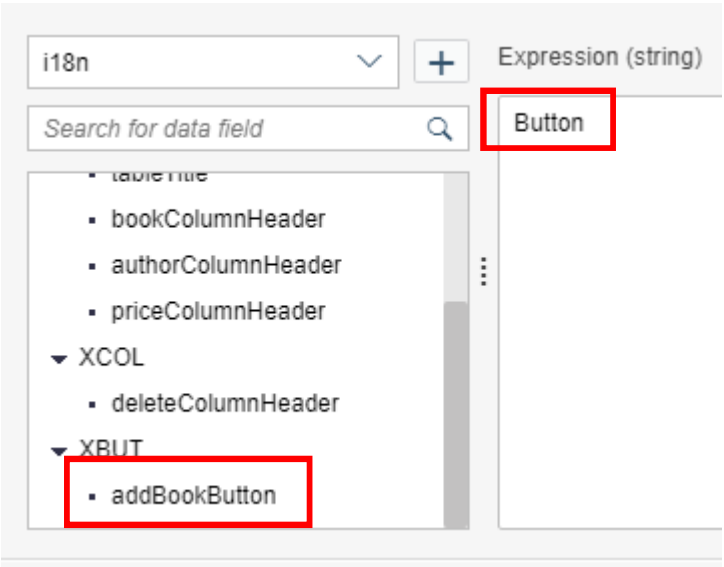
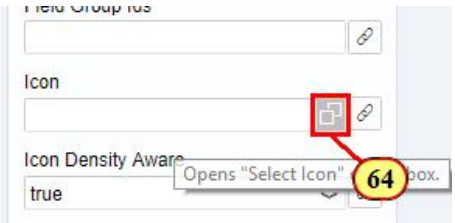
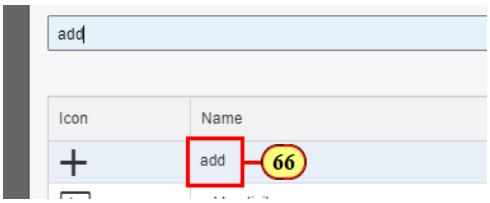
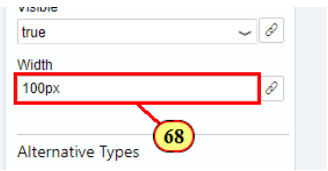
45. Add the **utils** file as the controller's required resource:
`"book/ui/utils/utils"`
 	
46. Add **utils** as the controller's input parameter.
 	
47. Click  .
	`sap.ui.define([`
	`  "sap/ui/core/mvc/Controller",`
	`  "book/ui/utils/utils"`
	`], function(Controller, utils) {`
	`  "use str`
	`  return Controller.extend(`
	`  /**`
	`  *@memberOf book.ui.controller.Books`
	`  */`
	`  onDeleteBook: function(oEvent) {`
	`  var oItemContext = oEvent.getSource().getBindingContext();`
	`  oItemContext.delete("$auto").then(function() {`
	`  utils.showInfoMessage(utils.getDefaultResourceBundle().getText("book_`
	`  });`
	`  }`

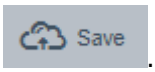
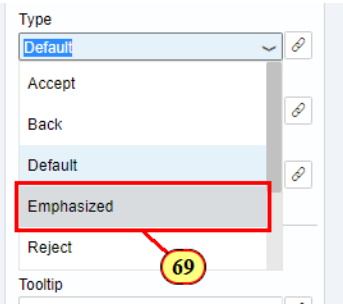
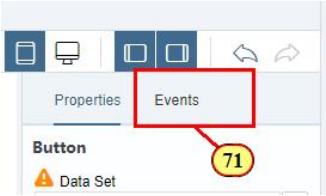
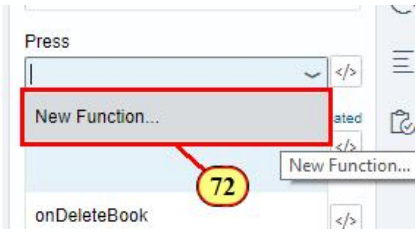
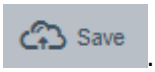
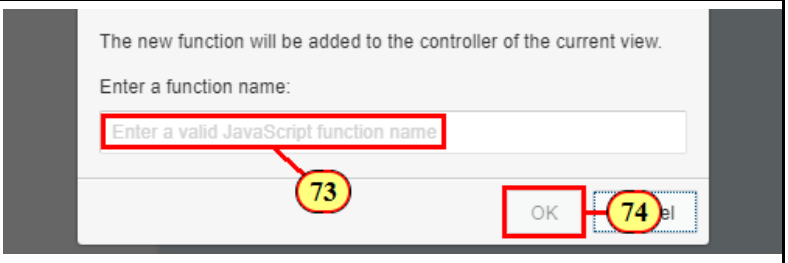
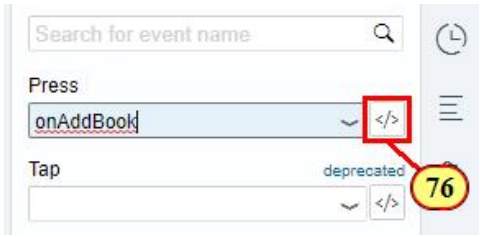
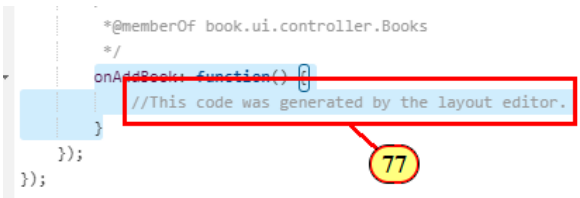
48. Click the **Books.view.xml** tab to return to the Layout Editor. 49. Open the **Outline** pane.	

Explanation	Screenshot
50. Expand the **Books.view.xml** navigation tree until you reach **sap.m.Table**. Expand the **headerToolbar** navigation tree until you reach **contentRight**.	
51. Right-click and select **Add**.	

Explanation	Screenshot
52. Click Button (sap.m) to add a button in the top right corner of the table.	
53. Open the **Properties** pane.	
54. In the **Properties** pane, click by the **Text** field to open the **Data Binding** dialog box.	

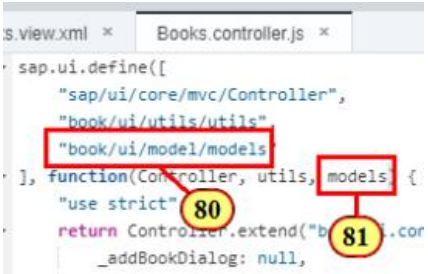
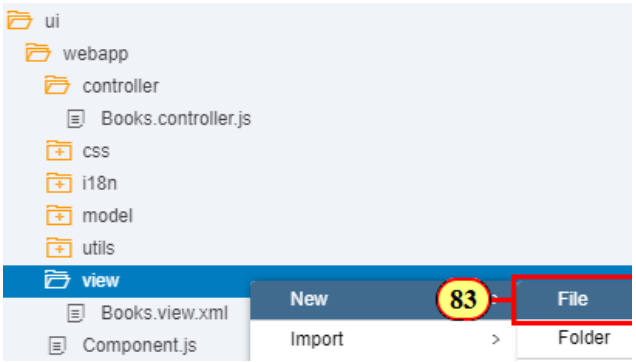
Explanation	Screenshot
55. From the dropdown list, select **i18n**. The i18n model contains the ui strings for translation.	
56. Click + to add a new translation entry.	
57. In the **String Key** field, enter **addBookButton**. 58. In the **Value** field, enter **Add Book**.	
59. From the **Type** dropdown list, select Button text. 60. Click OK.	

Explanation	Screenshot
61. Delete **Button** from the **Expression** dialog box. 62. Double-click **addBookButton**. 63. Click OK.	
64. By the **Icon** field, click to open the list of available icons.	
65. Enter **add** in the search field.	
66. Click add. 67. Click OK.	
68. Delete **100px** from the **Width** field.	

Explanation	Screenshot
69. From the **Type** dropdown list select **Emphasized**. 70. Click  .	
71. Open the **Events** pane.	
72. You will now define the action to be triggered when the **Add Book** button is pressed. From the **Press** dropdown list, select **New Function**.	
73. Enter **onAddBook** as the name of the new funtion to be added to the controller. 74. Click **OK**. 75. Click  .	
76. By the **Press** field, click  . The controller opens and you are directed to the location of the **onAddBook** method.	
77. Add the following code in the created method:	
`this._getNewBookDialog().open();`	

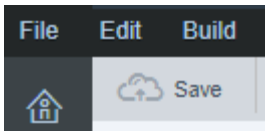
Explanation	Screenshot
78. Beneath the **onAddBook** method, add the **_getNewBookDialog** method using the following code:	

```
, _getNewBookDialog: function() {
this._addBookDialog = sap.ui.xmlfragment("book.ui.view.NewBook", this);
// set empty JSON model
this._addBookDialog.setModel(models.createNewBookDialogEmptyModel());
// set placeholders model
this._addBookDialog.setModel(models.createNewNookPlaceholdersModel(),
'PL');
this.getView().addDependent(this._addBookDialog);
return this._addBookDialog;
}
```

79. Add `_addBookDialog: null,` as a property of the controller.	
80. Add the **models** file as a resource of the controller after the **utils**. **"book/ui/model/models"** 81. Add **models** as an input property of the controller after **utils**. 82. Click Save.	
83. You will now create the dialog that will be ued to add books to the store. Go to **BookStore > ui > webapp > view**. Right-click and select **New > File**.	

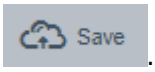
Explanation	Screenshot
84. Enter **NewBook.fragment.xml** as the file name. 85. Click OK.	
86. Add the following code to the file you just created:	

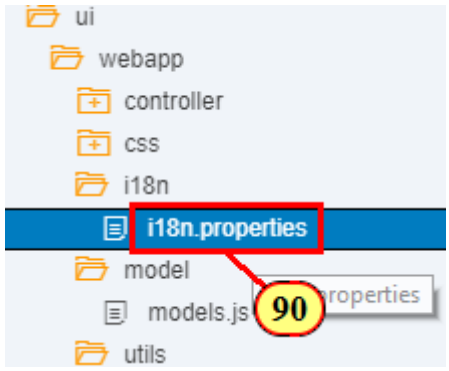
```
<core:FragmentDefinition xmlns="sap.m" xmlns:core="sap.ui.core">
      <Dialog title="{i18n>newBookDialogTitle}">
            <beginButton>
                  <Button text="{i18n>cancelButton}"
press="onDialogCancel"/>
            </beginButton>
            <endButton>
                  <Button text="Create" type="Emphasized"
press="onDialogCreate" enabled="{/canCreate}"/>
            </endButton>
            <content>
                  <VBox>
                        <Input type="Text" placeholder="{PL>/bookNamePL}"
value="{/data/bookName}" change="onDialogInputChange"/>
                        <Input type="Text" placeholder="{PL>/isbnPL}"
value="{/data/isbn}" change="onDialogInputChange"/>
                        <Input type="Text"
placeholder="{PL>/authorNamePL}" value="{/data/authorName}"
change="onDialogInputChange"/>
                        <Input type="Number" placeholder="{PL>/pricePL}"
value="{/data/price}" change="onDialogInputChange"/>
                  </VBox>
            </content>
      </Dialog>
</core:FragmentDefinition>
```

87. Click Save.	
88. Go to the controller file, add the following code to handle the **Add Book** dialog box:	

Explanation	Screenshot

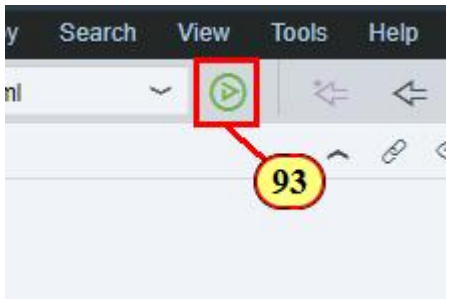
,
 	
onDialogCancel: function() {
this._addBookDialog.close();
},
 	
onDialogCreate: function() {
var oView = this.getView();
var self = this;
var oBinding =
this.getView().byId("booksTable").getBinding("items");
var oDialogModel = this._getDialogModel();
 	
// make sure price is an integer field
oDialogModel.setProperty("/data/price",
parseInt(oDialogModel.getProperty("/data/price")));
var oContext = oBinding.create(oDialogModel.oData.data);
 	
oContext.created().then(function() {
// refresh binding in order to allow the creation
of additional entities
self._addBookDialog.close();
 	
utils.showInfoMessage(utils.getDefaultResourceBundle().getText("newB
ookCreatedInfoMessage"));
});
 	
function resetBusy() {
self._addBookDialog.setBusy(false);
}
// lock UI
this._addBookDialog.setBusy(true);
 	
oView.getModel().submitBatch("myAppUpdateGroup").then(resetBusy,
resetBusy);
},
 	
onDialogInputChange: function() {
var oDialogModel = this._getDialogModel();
var canCreate =
oDialogModel.getProperty("/data/bookName").length > 0 &&
 	
oDialogModel.getProperty("/data/authorName").length > 0 &&
oDialogModel.getProperty("/data/isbn").length > 0
&&
oDialogModel.getProperty("/data/price").length >
0;
oDialogModel.setProperty("/canCreate", canCreate);
},
_getDialogModel: function() {
return this._addBookDialog.getModel();
}	

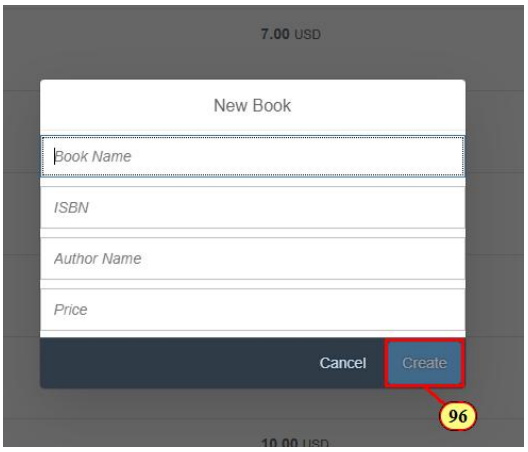
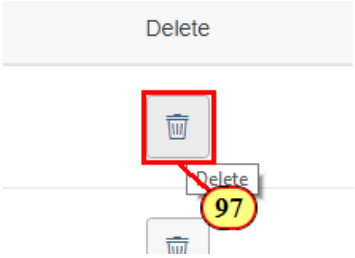
89. Click  .	

Explanation	Screenshot
90. Add translation capabilities to the strings we used in the **Add Book** dialog box. Go to **Bookstore > ui > webapp > i18n**. Double-click i18n.properties.	

91. Add the following strings at the bottom of the file.	`addBookButton=Add Book`
	
	`newBookDialogTitle=New Book`
	`cancelButton=Cancel`
	`newBookCreatedInfoMessage=Book has been created`
	`bookDeletedMessage=Book has been deleted`

`newBookDialogTitle=New Book`
`cancelButton=Cancel`
`newBookCreatedInfoMessage=Book has been created`
`bookDeletedMessage=Book has been deleted`	

92. Click Save.	
93. Click to preview the changes in the application.	
94. Click Add Book to create a new book.	

Explanation	Screenshot
95. Fill in the fields to create a new book entry in the store. 96. Click  .	
97. Click  to delete a book.	

Summary

You have completed the exercise!

You are now able to:

- Use Create and Delete capabilities in your application

7 CREATE THE BOOK STORE APPLICATION USING THE UPCOMMING SAP CLOUD PLATFORM PROGRAMMING MODEL

Overview

Estimated time: 50 minutes

Objective

In the following exercise, you will use the upcoming programming model.

You will see how easy it is to create the same application we did before, now with fewer steps and with the addition of CRUD capabilities.

Currently, the scope of the SAP Cloud Platform Programming Model is limited and experimental. This may still change.

The new BookStore application you create will be composed in the following way:

Core Data Services (CDS).

CDS allows you to define database tables, OData service entities, and even the annotations for the UI. In the previous application, we used HDB CDS (which is coupled with SAP HANA). The new CDS is built in such a way that will enable you (in the future) to work with other databases. Currently you will still use the SAP HANA database.

One major limitation is that currently no CDS views are supported. For this reason, it is not possible to define views composed of joint information from different database tables, or to calculate aggregates.

In the project, we will have 2 modules to represent this part:

CDS module - containing the CDS definitions

HDB module - containing the relevant outcome from the CDS module build and to which you can add other database capabilities.

JAVA

The second part is the OData V2 service based on a Java framework component that allows the automatic exposure of the CDS service definition without any additional application code.

In case of writable CDS views, the framework even offers an automatic create, update, and delete functionality.

Nevertheless, an application developer can extend the framework logic by providing its own Java code in the form of annotated extension classes.

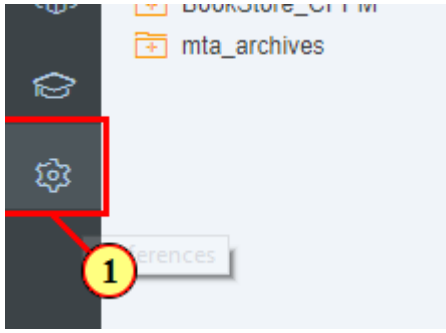
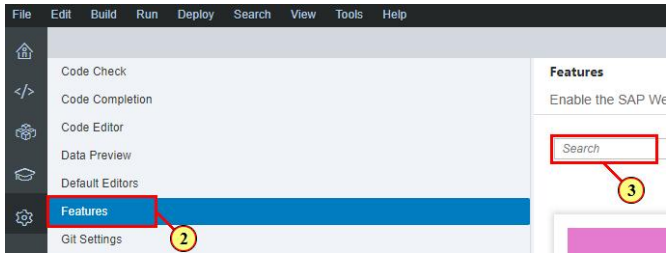
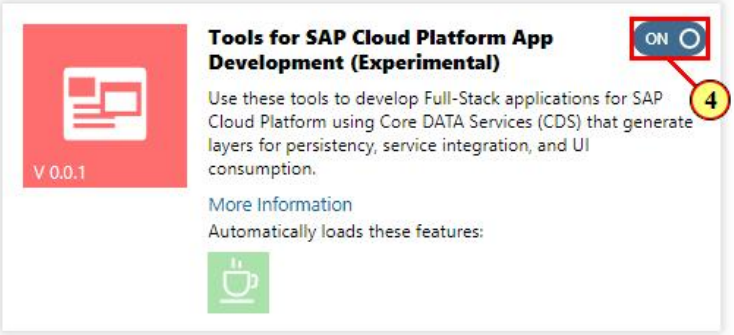
SAP Fiori Elements

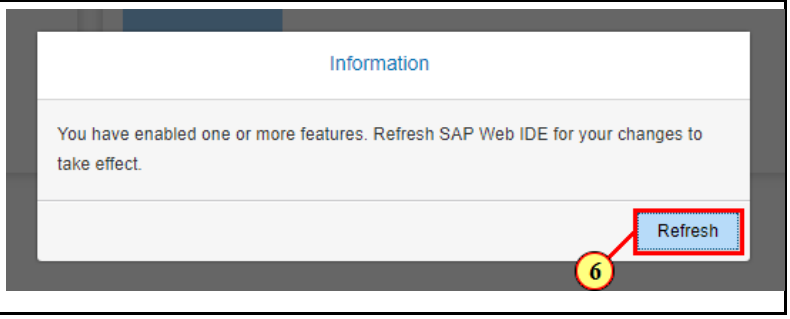
Finally, the UI forms the third part which is SAP Fiori Elemnts. With this technology based on SAPUI5, it is possible to choose from a predefined set of UI templates that allows you to create a UI just by defining annotations in CDS instead of writing JavaScript code.

Exercise Description

- Create new project in SAP Web IDE.
- Define the application data model and service.
- Create the SAP HANA DB schema.
- Expose the OData Service.
- Create UI using SAP Fiori elements.

7.1 Enable the New Tools

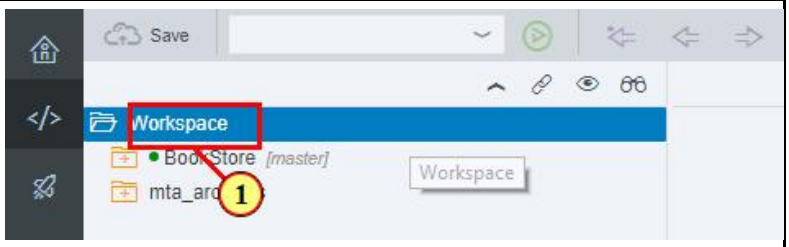
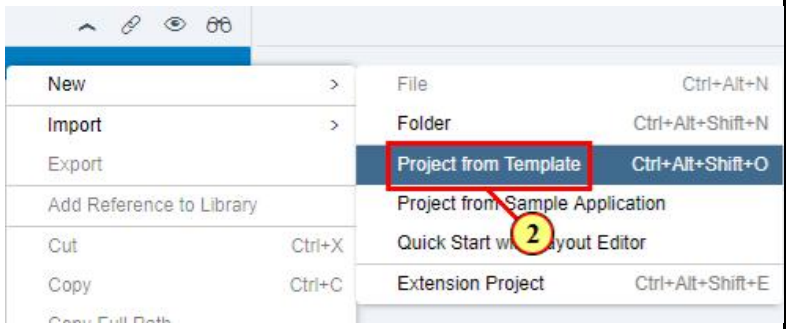
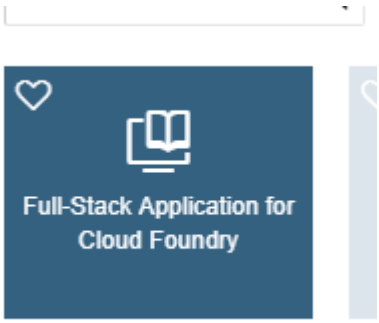
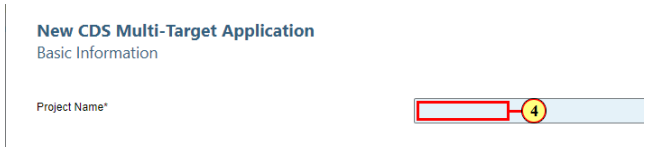
Explanation	Screenshot
1. Click  to open the preferences perspective.	
2. Click **Features**. 3. Enter **cloud** in the search field.	
4. Look for the **Tools for SAP Cloud Platform App Development** feature and click the **ON/OFF** toggle button to enable it. 5. Click  .	

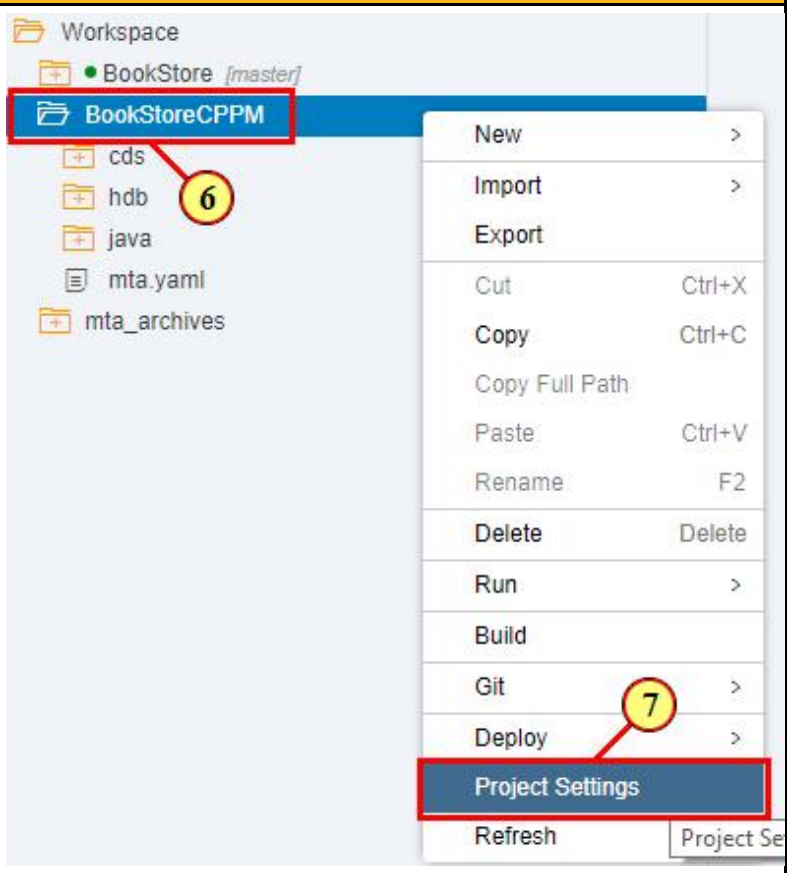
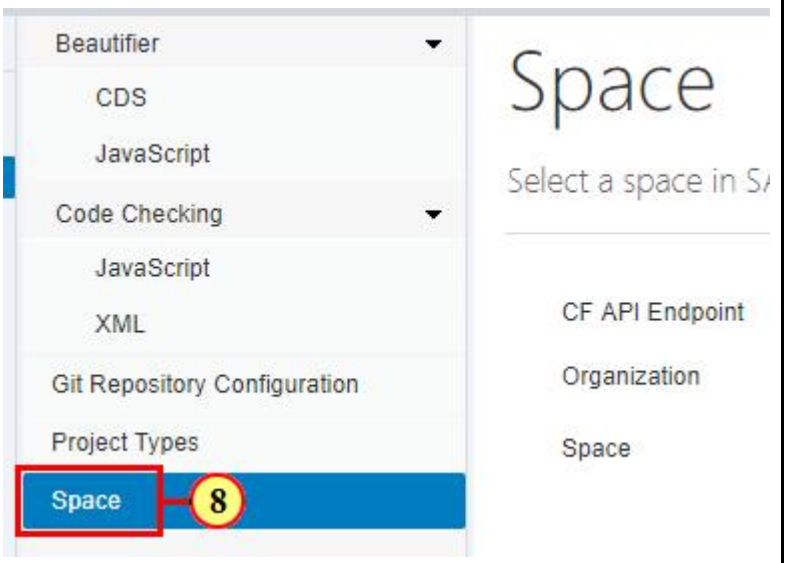
Explanation	Screenshot
6. Click Refresh to reload SAP Web IDE with the new features.	

7.2 Create a New Project in SAP Web IDE

In this step, you will create a new MTA project using the **Full-Stack Application for Cloud Foundry** template.

As opposed to the previous exercise where you had to create the modules for your MTA, with this template, the **cds**, **java** and **hdb** modules are automatically generated within the MTA, including the relationships between them.

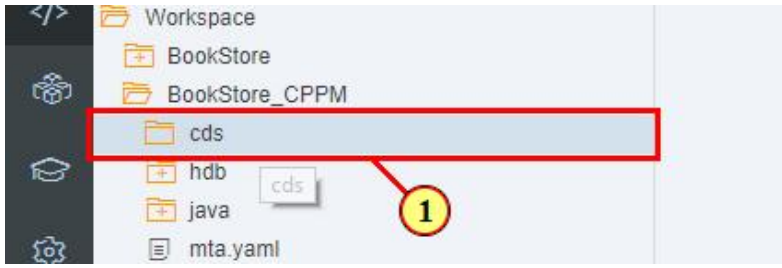
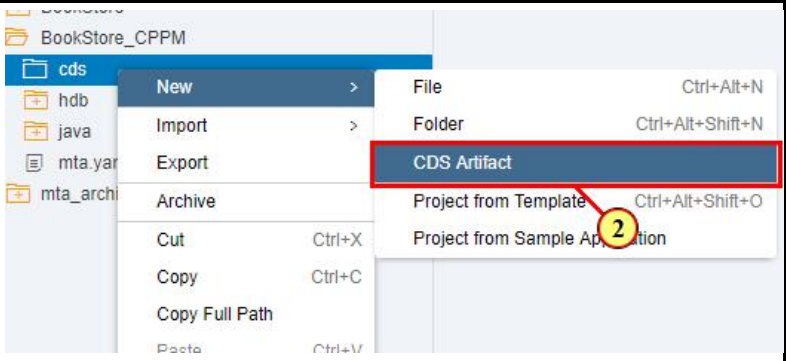
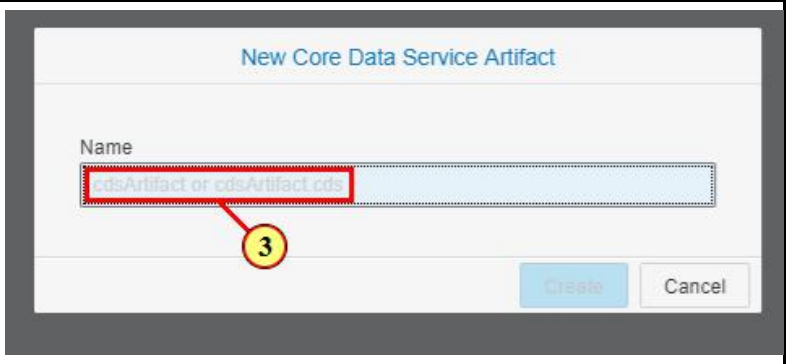
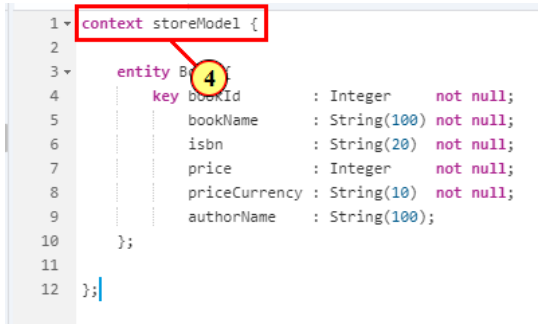
Explanation	Screenshot
1. Right-click **Workspace**.	
2. Click Project from Template .	
3. Select the **Full-Stack Application for Cloud Foundry** template, and click **Next**.	
4. Enter **BookStoreCPPM** in the **Project Name** field, and click **Next**.	
5. Click Finish .	

Explanation	Screenshot
6. Right-click BookStoreCPPM. 7. Click Project Settings.	
8. Click Space.	
9. Make sure the CF API Endpoint, organization and space are updated, and click **Save**. 10. Click Close.	

7.3 Define the Application using CDS

In this step, you will use the CDS to define the application data model, the OData service that is being exposed using Java, and UI annotations for the SAP Fiori elements application.

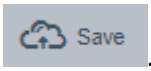
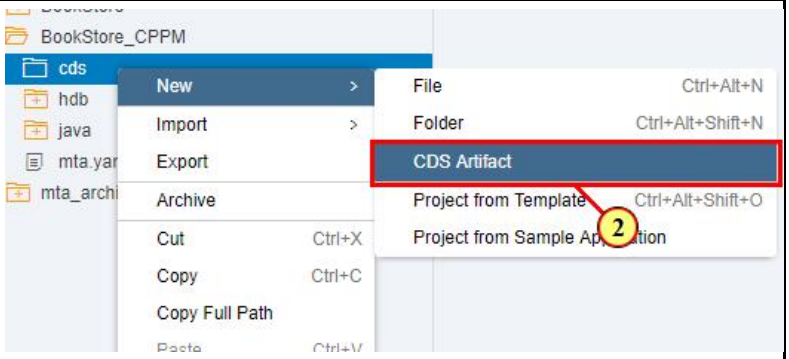
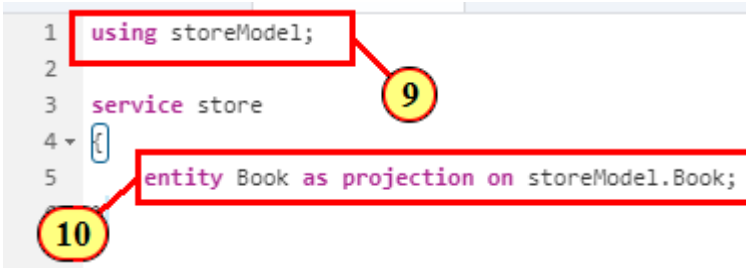
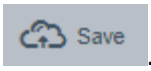
After you build the CDS module, the HDB CDS artifacts are copied to the HDB module and the OData metadata file, together with the **csn.json** file (a JSON representation of the model) is added to the **edmx** folder in the Java module.

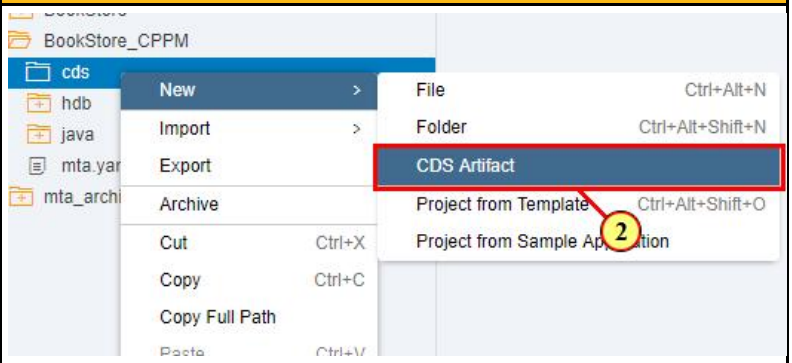
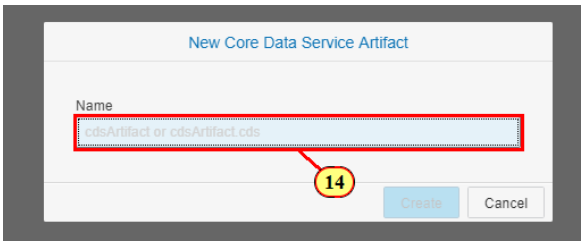
Explanation	Screenshot
1. Right-click the **cds** folder.	
2. Go to **BookStore_CPPM > cds**. Rigkt-click and select **New > CDS Artifact**.	
3. Enter **storeModel** as the artifact name and click **Create**.	
4. In the **datamodel.cds** file, create the **Book** entity using the following code:	

```
context storeModel {

      entity Book {
```

Explanation	Screenshot

`key bookId : Integer not null;`
`bookName : String(100) not null;`
`isbn : String(20) not null;`
`price : Integer not null;`
`priceCurrency : String(10) not null;`
`authorName : String(100);`
`};`
`};`	

5. Click  .	
6. Go to **BookStore_CPPM > cds**. Rigkt-click and select **New > CDS Artifact**.	
8. Enter **store** as the artifact name and click **Create**.	
9. Add the **using storeModel** that was previously defined. 10. Expose the **Book** entity as follows:	
`entity Book as projection on storeModel.Book;`	
11. Click  .	

Explanation	Screenshot
12. Go to **BookStore_CPPM > cds**. Rigkt-click and select **New > CDS Artifact**.	
14. Enter **annotation** as the artifact name and click **Create**.	

15. Using SAP Fiori elements, you will display the Book data in a table.

For this table, you need to annotate the Book fields using the

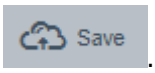
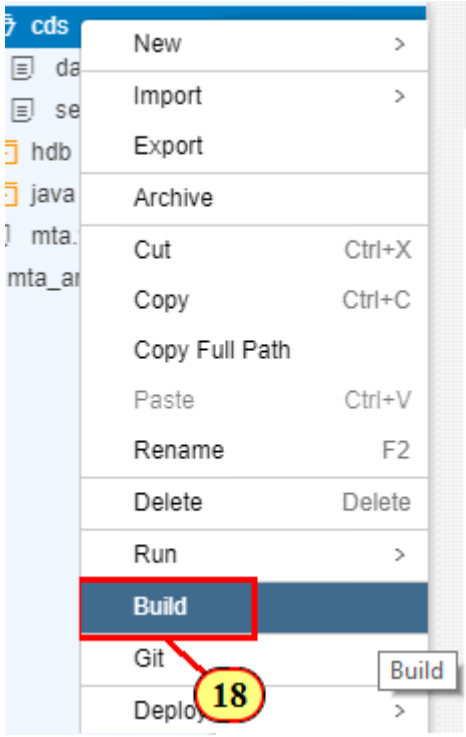
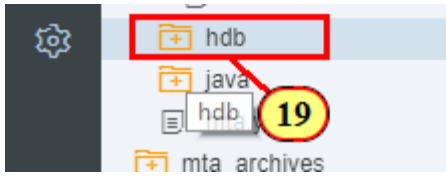
UI.LineItems.

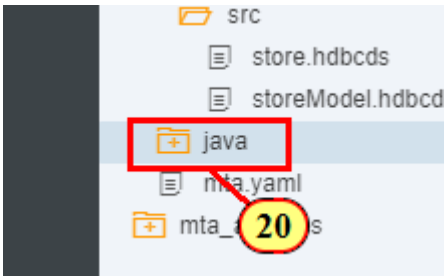
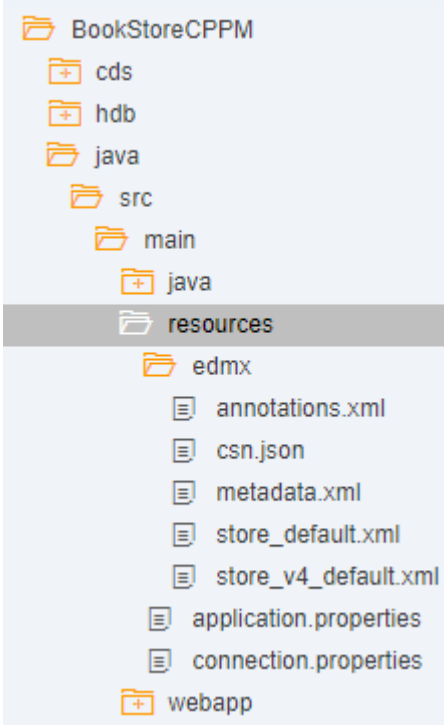
See sample code below.

```
using store;

annotate store.Book with
{
      @UI.HiddenFilter
      @Common.Label: 'Book'
      @Common.SemanticObject: 'EBookt'
      @Common.Text: name
      @Core.Immutable
      @UI.Hidden: true
      bookId;
      @Common.Label: 'Book Name'
      bookName;
      @Common.Label: 'Author Name'
      authorName;
      @Common.Label: 'Price'
      @Measures.ISOCurrency: priceCurrency
      price;
      @Common.Label: 'Currency'
      priceCurrency;
};

annotate store.Book with @(
      Common.SemanticKey: [bookId],

      UI.LineItem: [
            {$Type: 'UI.DataField', Value: bookName, "@UI.Importance":
#High},
      {$Type: 'UI.DataField', Value: authorName, "@UI.Importance": #High},
      {$Type: 'UI.DataField', Value: price, "@UI.Importance": #High},
```

Explanation	Screenshot

`{$Type: 'UI.DataField', Value: priceCurrency, "@UI.Importance": #High}`
`],`
 	
`UI.HeaderInfo: {`
`TypeName:'Book',`
`TypeNamePlural:'Books',`
`Title:{Value:bookName},`
`},`
 	
`UI.SelectionFields: [bookName,authorName]`
 	
`);`	

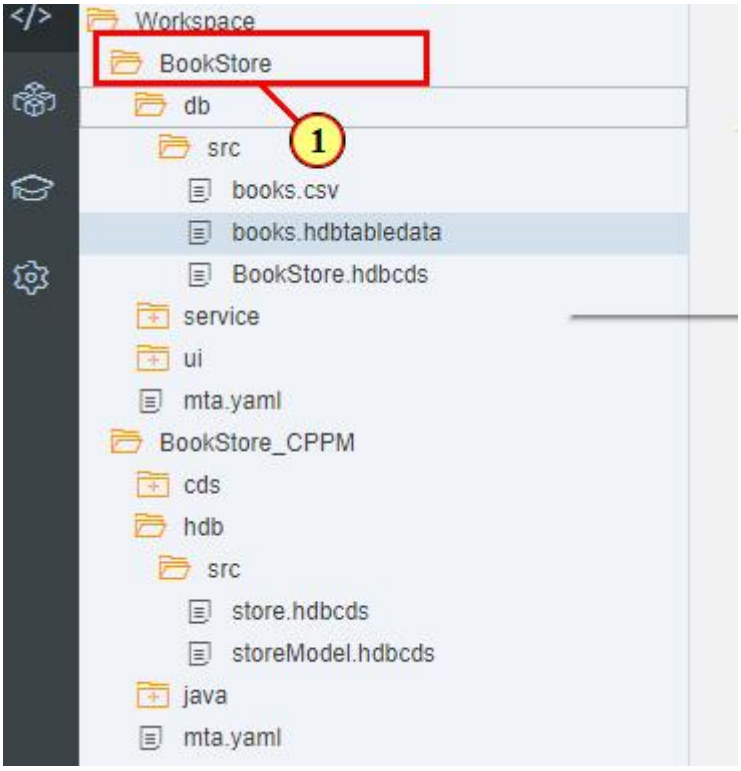
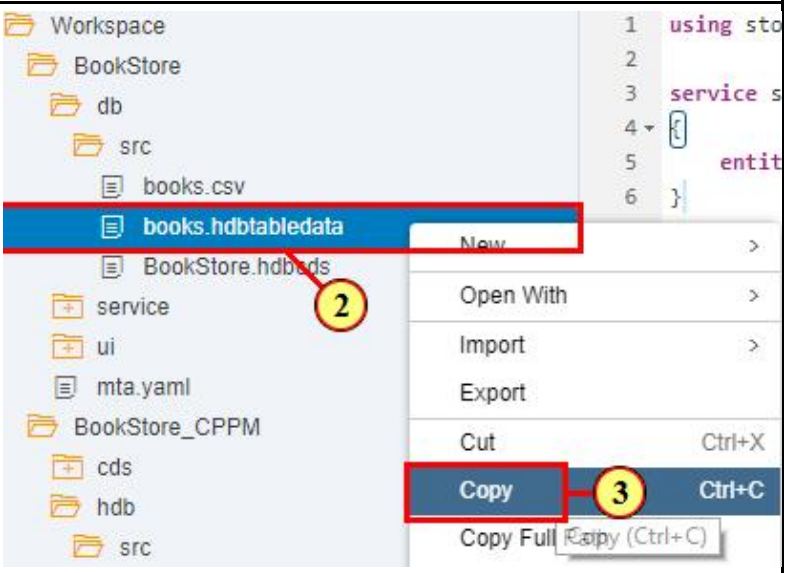
16. Click  .	
18. Go to **BookStore_CPPM > cds**. Rigkt-click and select **Build**. This will create the HDB CDS artifacts and the metadata of the OData service.	
19. Go to **BookStore_CPPM > hdb**. Open the **hdb** folder to see the generated **hdbcds** files.	

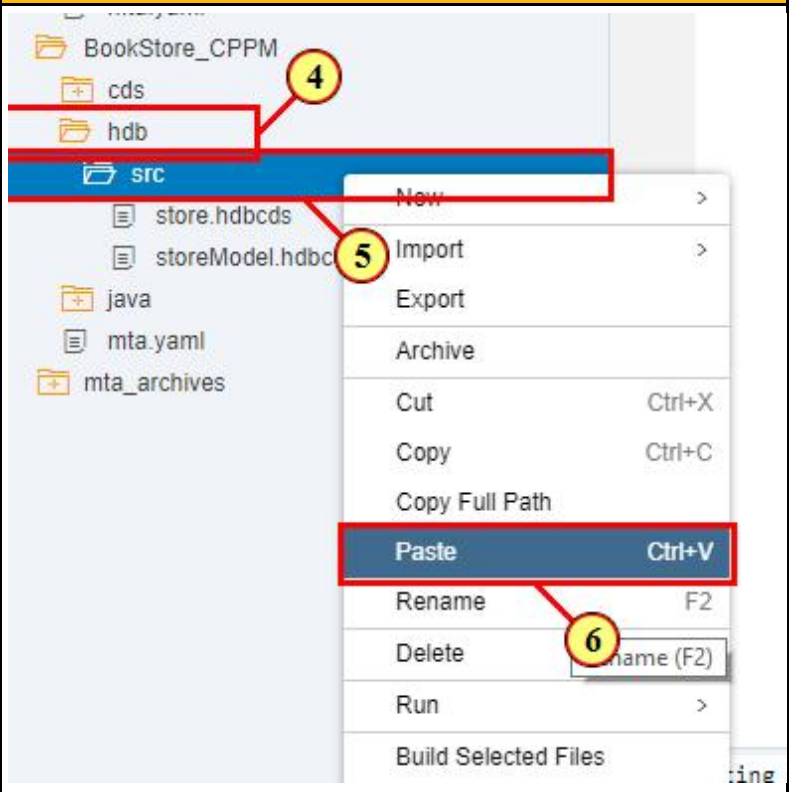
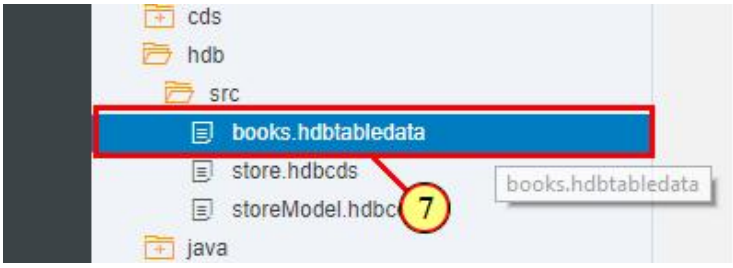
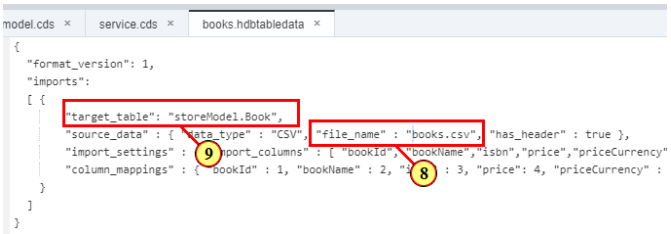
Explanation	Screenshot
20. Go to **BookStore_CPPM > java**. 21. Open the **java** folder and click **resources**.	
22. Open the **edmx** folder to see the generated OData metadata file.	

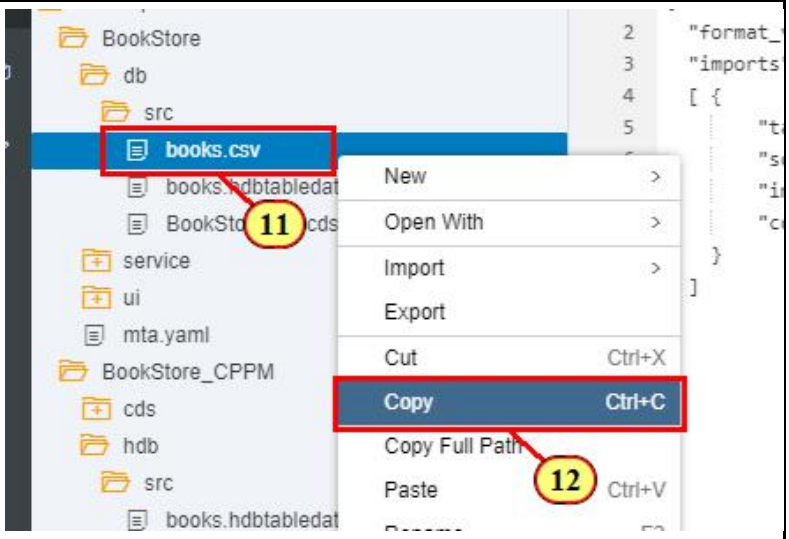
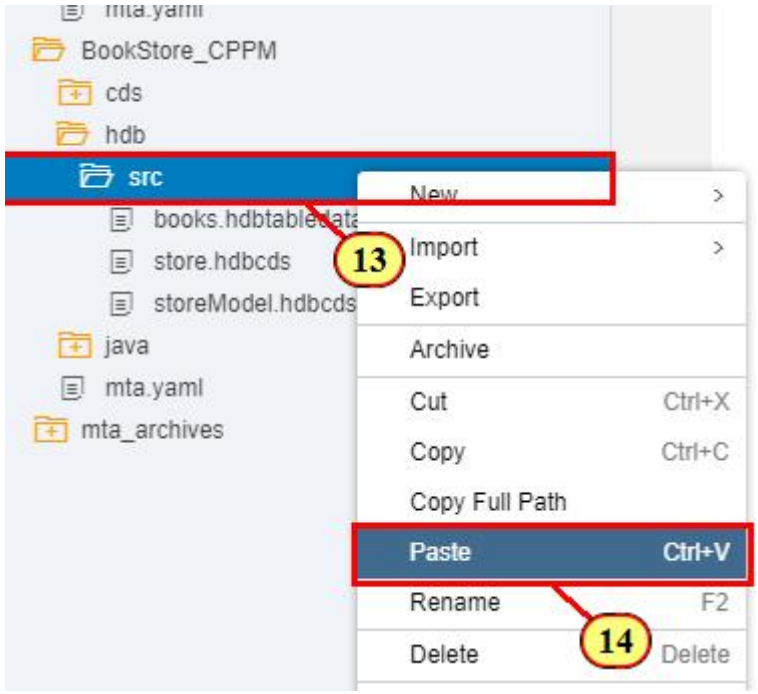
7.4 Create the SAP HANA DB Schema

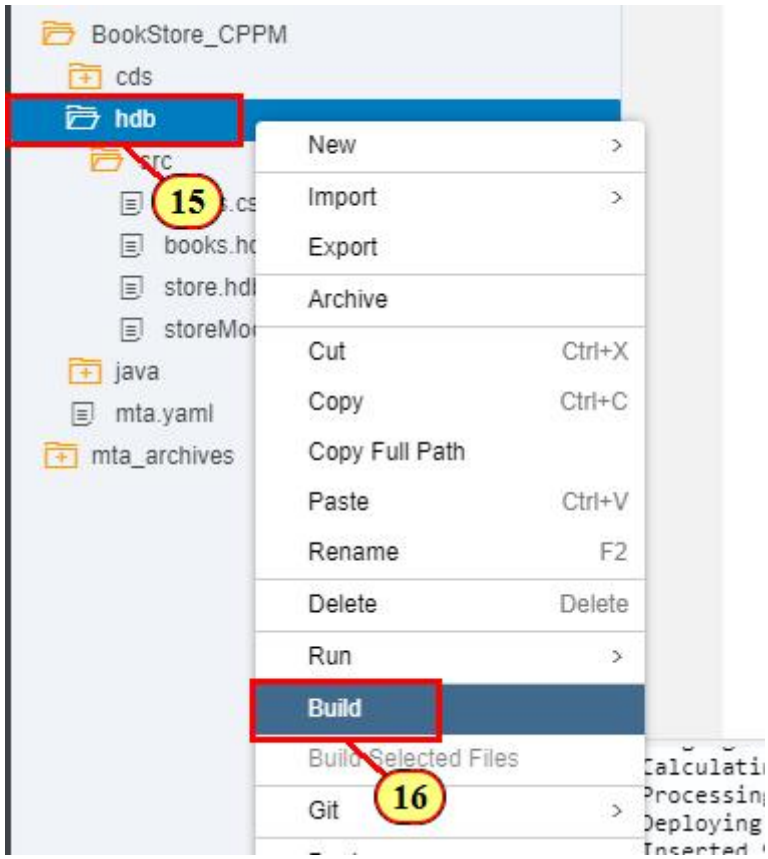
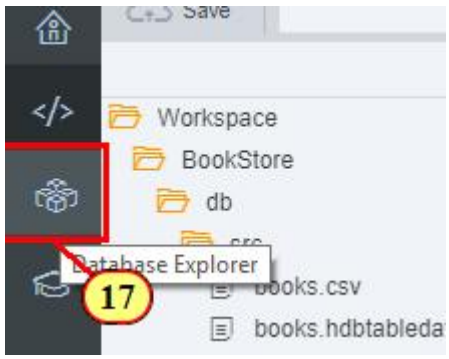
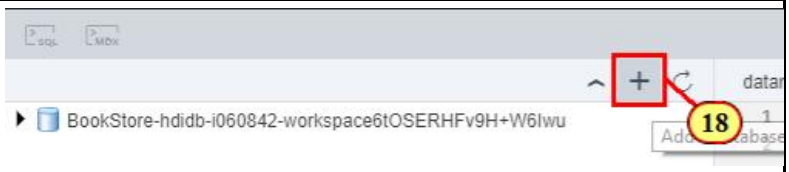
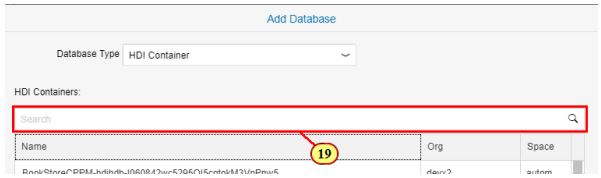
In this step, you will activate, in the SAP HANA data base, the schema and tables you defined in the previous step.

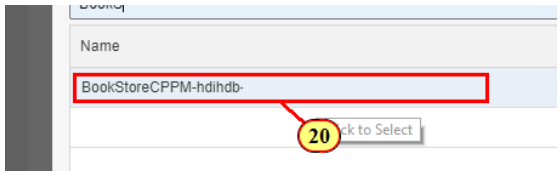
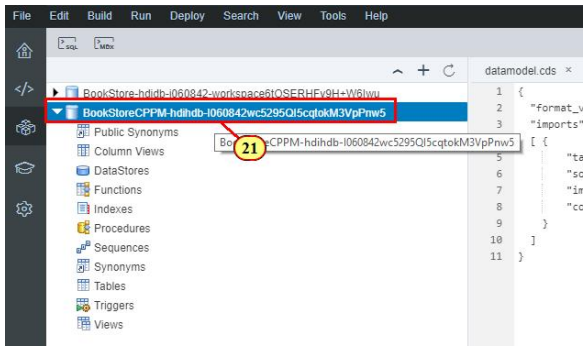
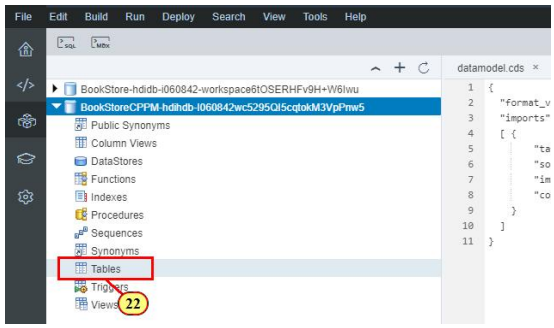
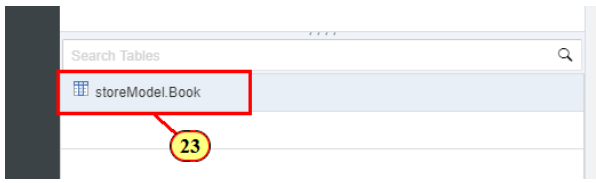
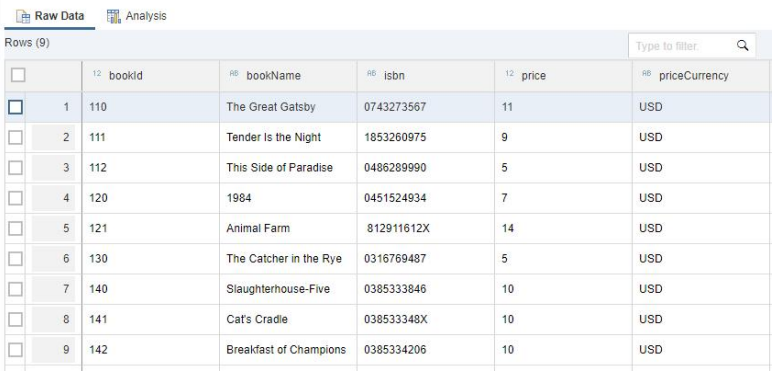
In order to populate the tables with data, you will reuse the data in the **hdbtabledata** and the **csv** files of the previous project.

Explanation	Screenshot
1. Open the **Bookstore** project you created in the previous exercises. Go to **BookStore > db > src**.	
2. Right-click on the **books.hdbtabledata** file. 3. Click **Copy**.	

Explanation	Screenshot
4. In the new project, open the **BookStore_CPPM > hdb** folder. 5. Right-click the **src** folder. 6. Click **Paste**.	
7. Go to **BookStore_CPPM > hdb > src**. 8. Double-click the **books.hdbtabedata** file to open it.	
9. Update the table name to **storeModel.Book** 10. Click  Save.	

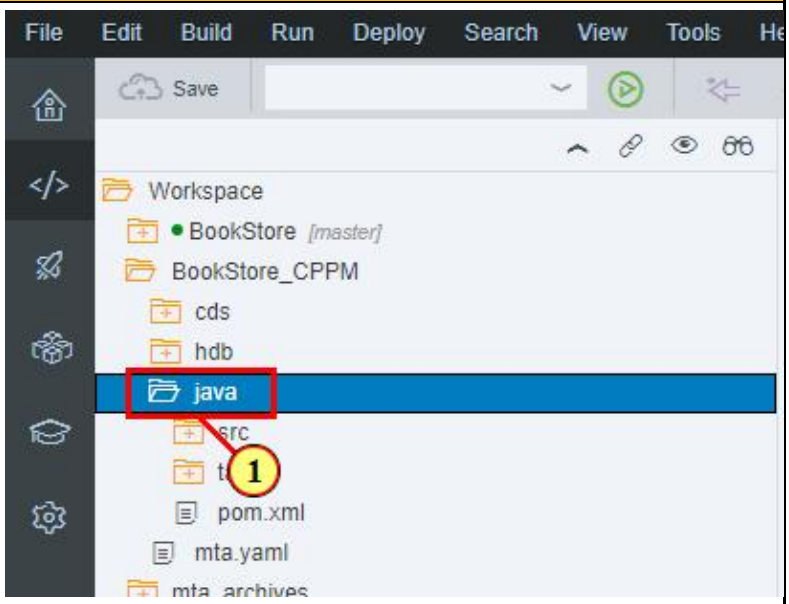
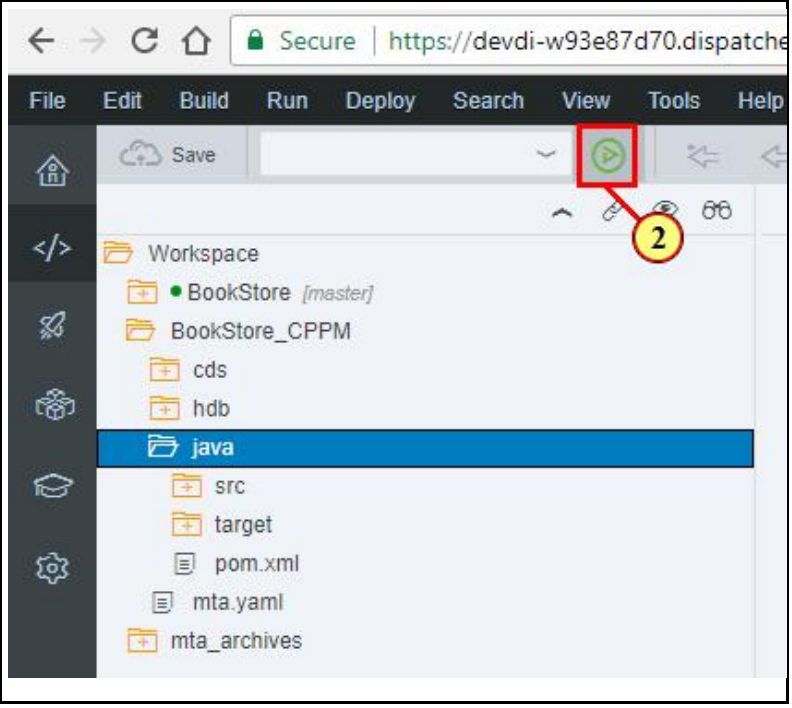
Explanation	Screenshot
11. Go to **BookStore > db > src** and right-click the **books.csv** file. 12. Click **Copy**.	
13. Go to **BookStore_CPPM > db**. Right-click the **src** folder. 14. Click **Paste**.	

Explanation	Screenshot
15. Right-click the **hdb** folder. 16. Click **Build**.	
17. Once the build finishes, click  to open the Database Explorer perspective. In the Database Explorer you can view information about your database's catalog objects and execute queries.	
18. Click  to add the newly created data base.	
19. Enter **BookStore_CPPM** in the search field.	

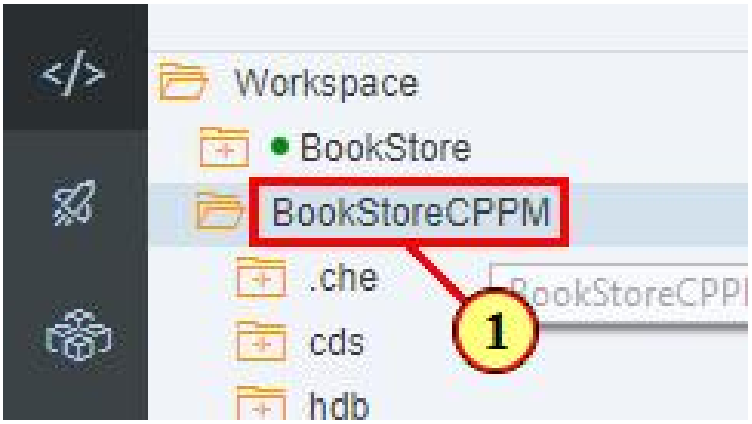
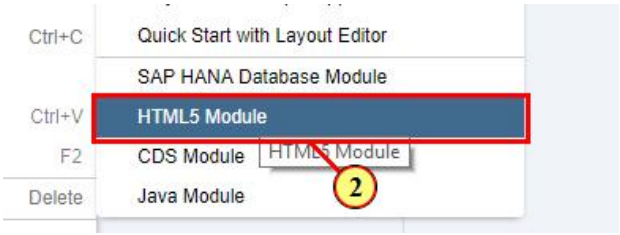
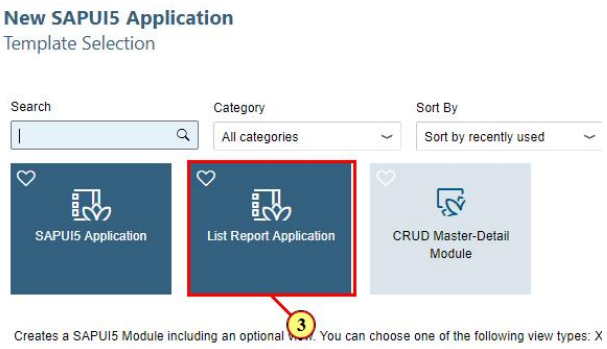
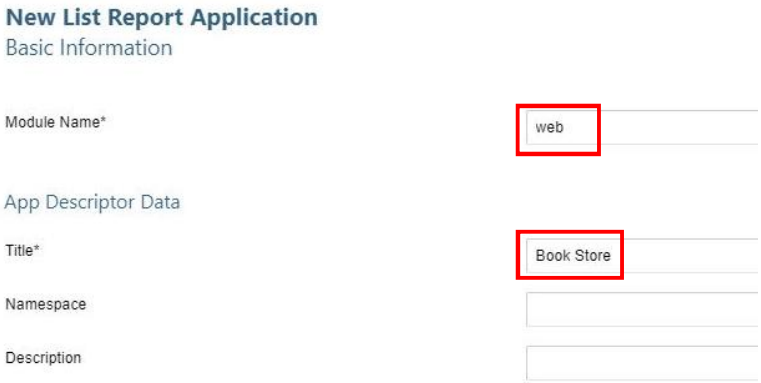
Explanation	Screenshot
20. Click the schema in the list and click **OK**.	
21. Click the HDI container.	
22. Click **Tables**.	
23. Click **storeModel.Book** to see the table's properties.	
24. Click Open Data.	
25. The data is displayed.	

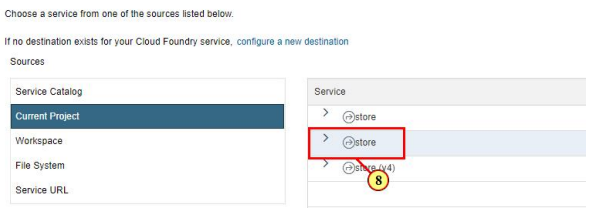
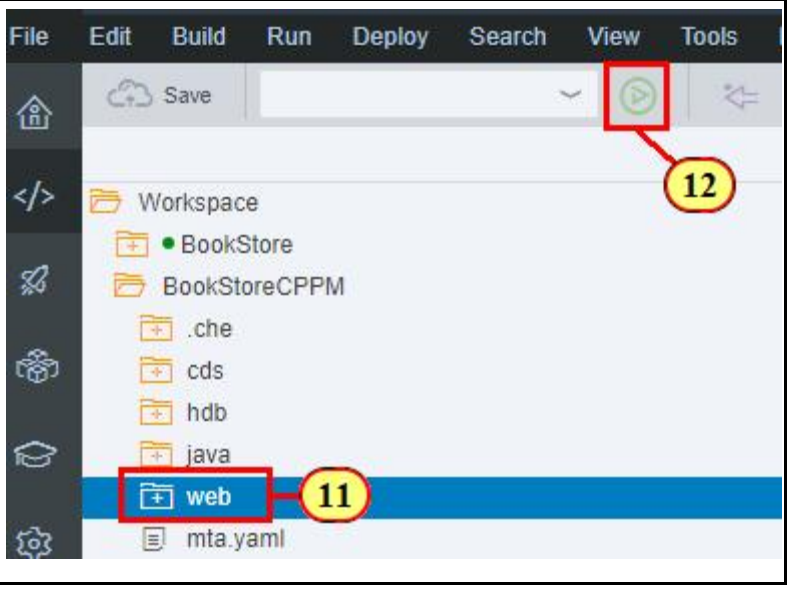
	bookId	bookName	isbn	price	priceCurrency
1	110	The Great Gatsby	0743273567	11	USD
2	111	Tender Is the Night	1853260975	9	USD
3	112	This Side of Paradise	0486289990	5	USD
4	120	1984	0451524934	7	USD
5	121	Animal Farm	812911612X	14	USD
6	130	The Catcher in the Rye	0316769487	5	USD
7	140	Slaughterhouse-Five	0385333846	10	USD
8	141	Cat's Cradle	038533348X	10	USD
9	142	Breakfast of Champions	0385334206	10	USD

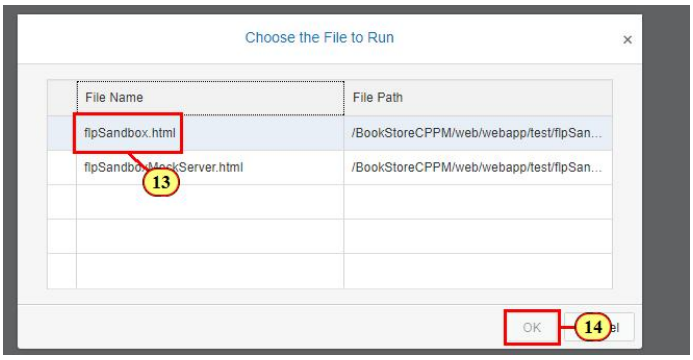
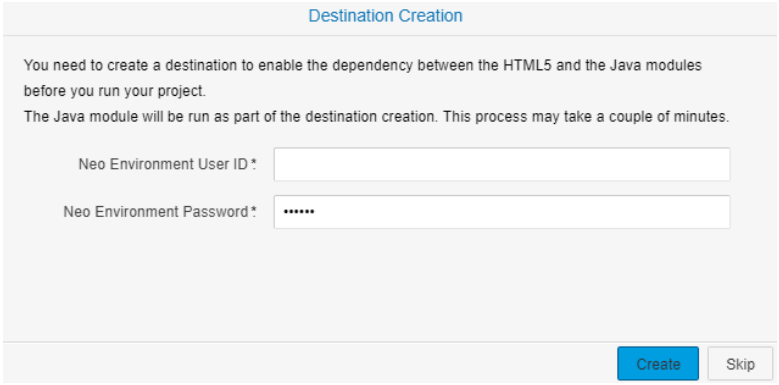
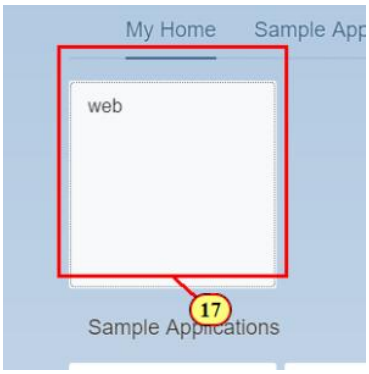
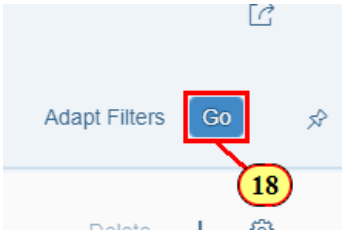
7.5 Expose the OData Service

Explanation	Screenshot
1. Go to **BookStore_CPPM > java**.	
2. Click  (Run).	

7.6 Create UI using SAP Fiori elements

Explanation	Screenshot
1. Right-click the **BookstoreCPPM** folder.	
2. Click **New > HTML5 Module**.	
3. Select the **List Report Application** template and click **Next**.	
4. Enter **web** as the module name. 5. Enter **Book Store** as the title.	

Explanation	Screenshot
6. Click **Current Project**.	
7. Click **store**. 8. Click Next 9. Click Next once again.	
10. From the **OData Collection** dropdown list, select **Book**. Click **Finish**. The HTML5 module is added to your project.	
11. Go to **BookStore_CPPM > web**. 12. Click (Run).	

Explanation	Screenshot
13. Click **flpSandbox.html** to run the application in the SAP Fiori Launchpad Sandbox. 14. Click OK.	
15. You need to create a destination to enable the dependency between the HTML5 and the Java modules. Enter your Neo Environment credentials. 16. Click Create.	
17. Click on the application tile in the SAP Fiori Launchpad Sandbox.	
18. Click Go.	

Summary

You have completed the exercise!

You are now acquainted with the upcoming SAP Cloud Platform programming model.

With it, you experienced how fast and easy it is to create full-stack apps which include CRUD capabilities.

8 INTRODUCTION TO ANALYTICAL SAP HANA DATABASE DEVELOPMENT

Overview

Estimated time: 10 minutes

Objective

In the following exercise, you will learn how to create an SAP HANA Calculation View to show advanced sales data.

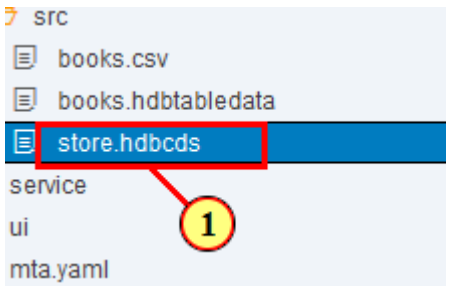
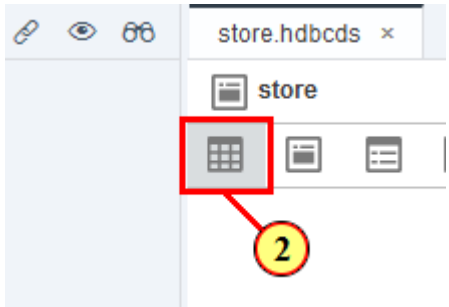
You will add the raw sales data to your SAP HANA database and then use a Calculation View to create joins, aggregations, and calculations on this data.

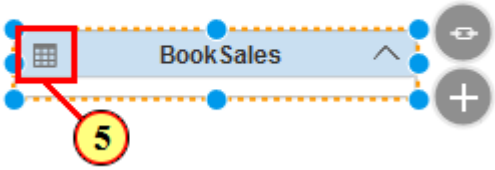
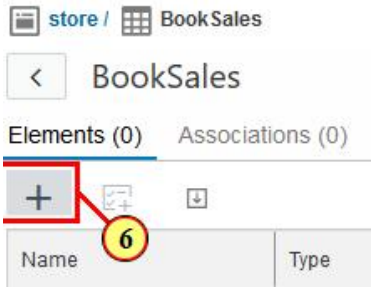
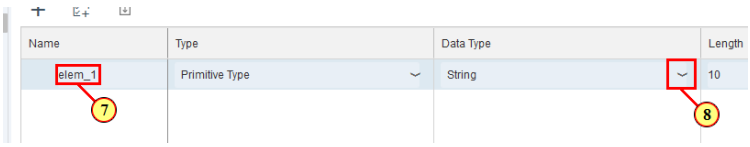
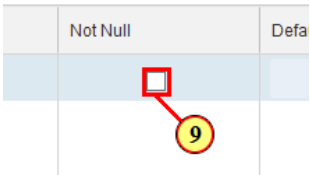
Exercise Description

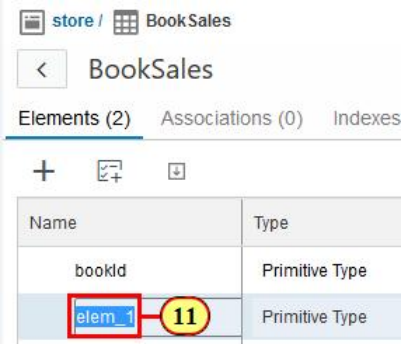
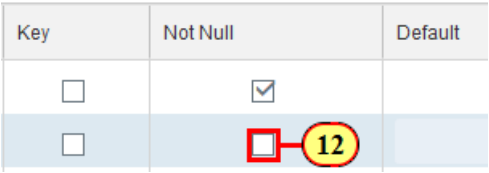
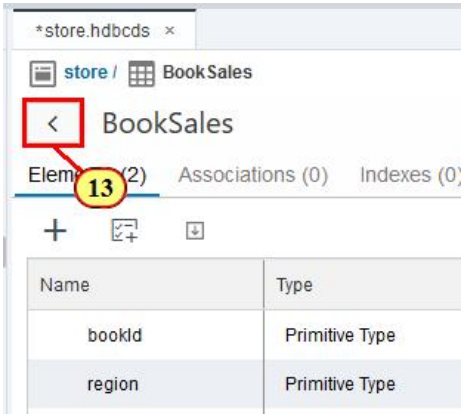
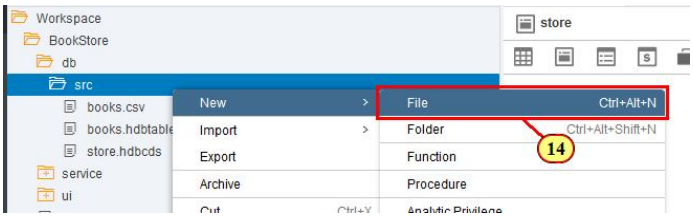
- Add a new entity to the SAP HANA CDS mode.l
- Create a Calculation View to calculate sales data.
- Build the SAP HANA module and review the runtime data.

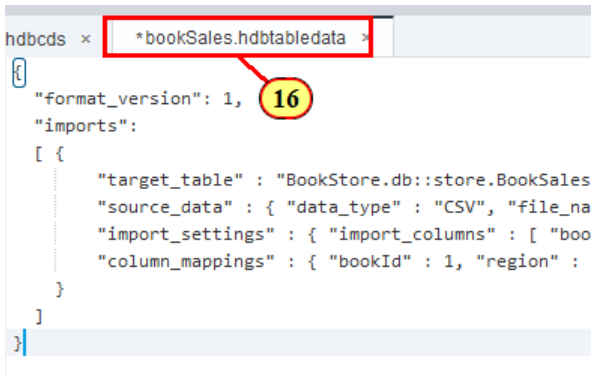
8.1 Add Sales Info to Model

In this step, you will add sales information to the model you created in the previous exercises.

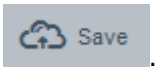
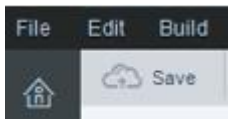
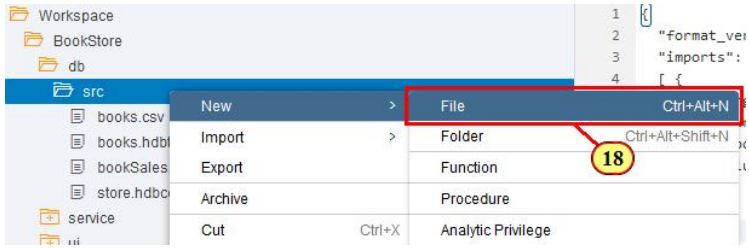
Explanation	Screenshot
1. Go to **Bookstore > db > src** and double-click to open the **HDBCDS** file.	
2. Click  to create a new entity. 3. Click on the canvas. A new entity is created.	

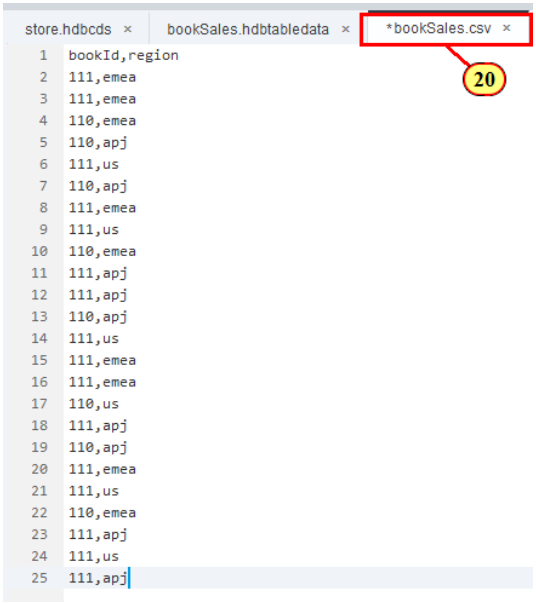
Explanation	Screenshot
4. Change the name of Entity_1 to **BookSales**.	
5. Double-click to open the view the entity's definitions.	
6. Click + to add a ngew element.	
7. Enter **bookId** in the **Name** field. 8. Open the **Data Type** dropdown list and select **Integer**.	
9. Select the checkbox to mark the property as **Not Null**.	
10. Click +.	

Explanation	Screenshot
11. Enter **region** in the **Name** field.	
12. Select the checkbox to mark the property as **Not Null**. Click **Save**.	
13. Click < to return to the model.	
14. Go to **BookStore > db**. Right-click **src** and select **New > File**.	
15. Name the new file **bookSales.hdbtabledata** and click **OK**.	

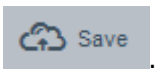
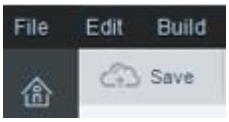
Explanation	Screenshot
16. Add the code snippet below to the new file:	

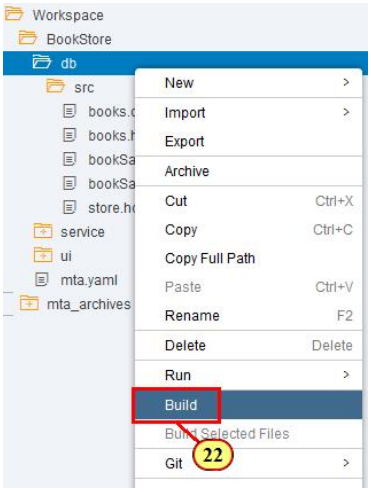
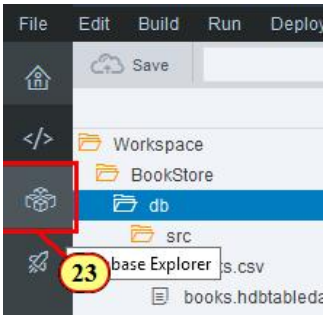
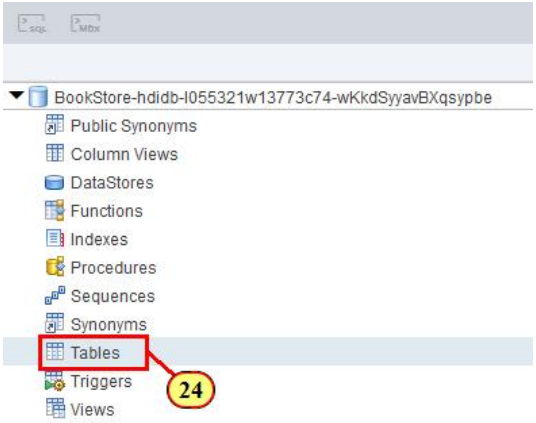
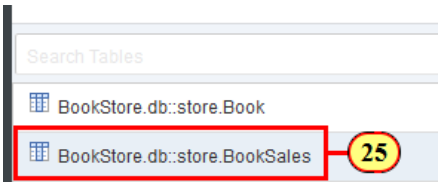
```
{
"format_version": 1,
"imports":
[ {
"target_table" : "store.BookSales",
"source_data" : { "data_type" : "CSV", "file_name" : "bookSales.csv",
"has_header" : true },
"import_settings" : { "import_columns" : [ "bookId", "region" ] },
"column_mappings" : { "bookId" : 1, "region" : 2 }
}
]
}
```

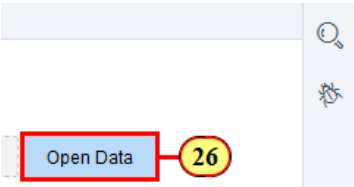
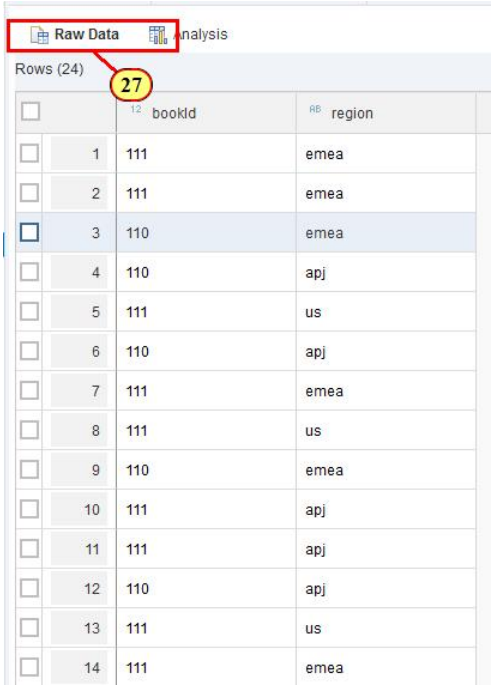
17. Click  .	
18. Go to **BookStore > db**. Right-click **src** and select **New > File**.	
19. Call the new file **bookSales.csv** and click **OK**.	

Explanation	Screenshot
20. Add the sample data below to the new file:	

```
bookId,region
111,emea
111,emea
110,emea
110,apj
111,us
110,apj
111,emea
111,us
110,emea
111,apj
111,apj
110,apj
111,us
111,emea
111,emea
110,us
111,apj
110,apj
111,emea
111,us
110,emea
111,apj
111,us
111,apj
```

21. Click  .	

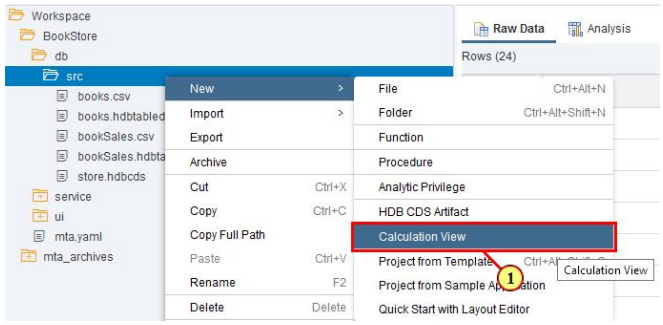
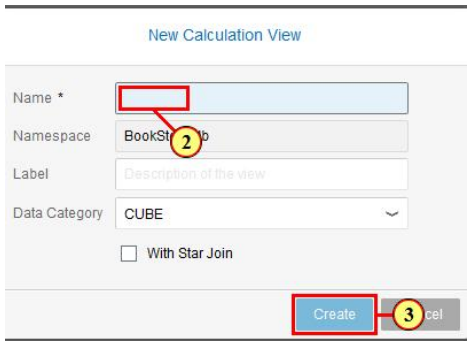
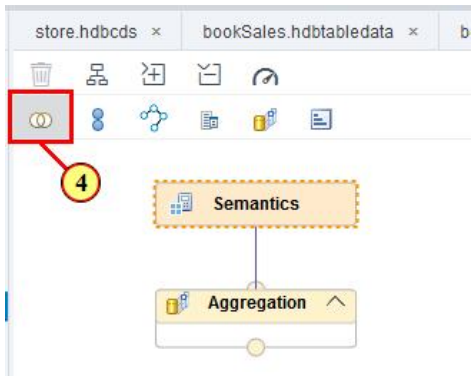
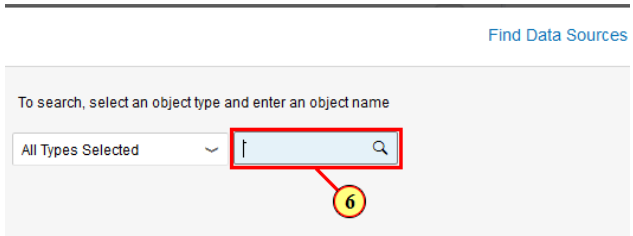
Explanation	Screenshot
22. Under **BookStore**, right-click **db** and click Build.	
23. Once build finishes, click  to open the Database Explorer perspective. In the Database Explorer you can view information about your database's catalog objects and execute queries.	
24. Open the database and select **Tables**.	
25. Select the **BookSales** table.	

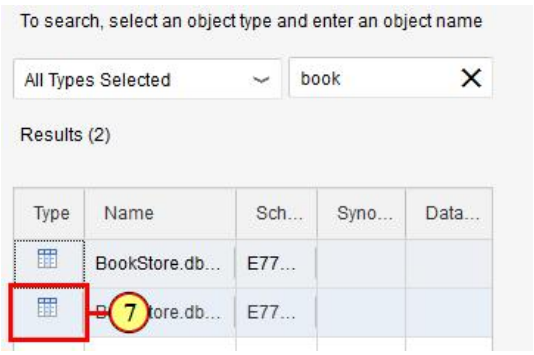
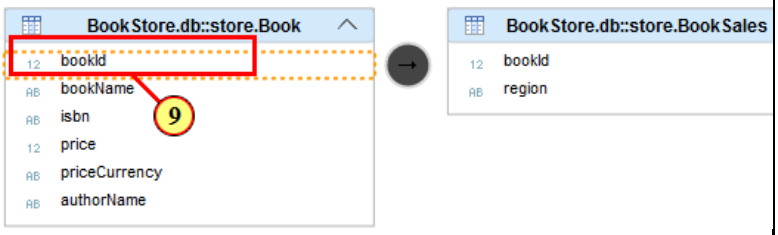
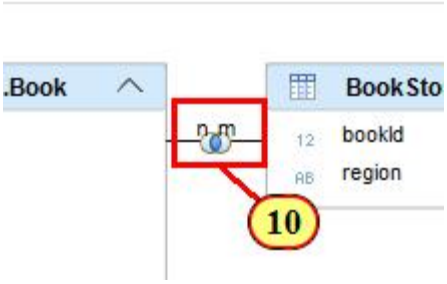
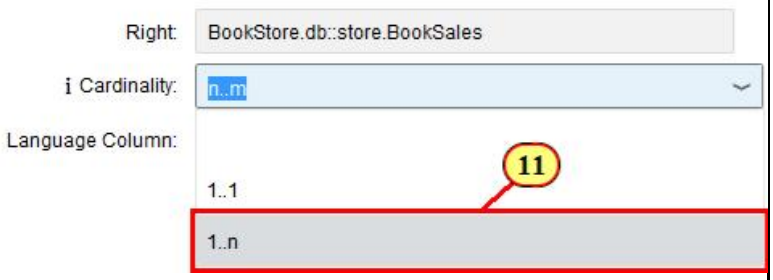
Explanation	Screenshot
26. Click Open Data.	
27. Your data is displayed.	

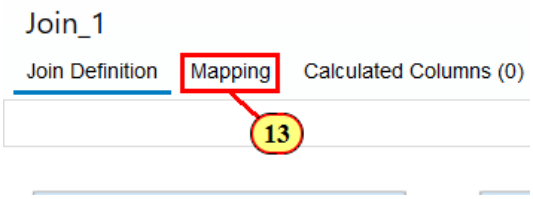
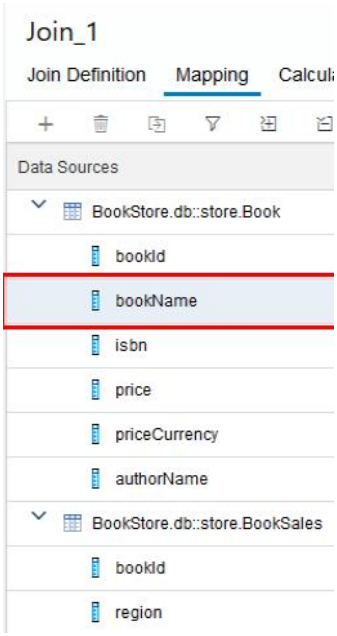
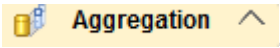
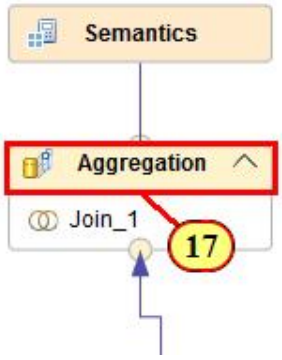
	bookId	region
1	111	emea
2	111	emea
3	110	emea
4	110	apj
5	111	us
6	110	apj
7	111	emea
8	111	us
9	110	emea
10	111	apj
11	111	apj
12	110	apj
13	111	us
14	111	emea

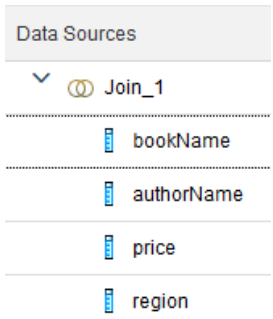
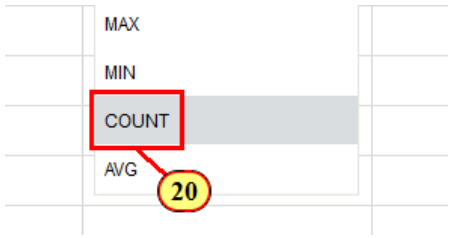
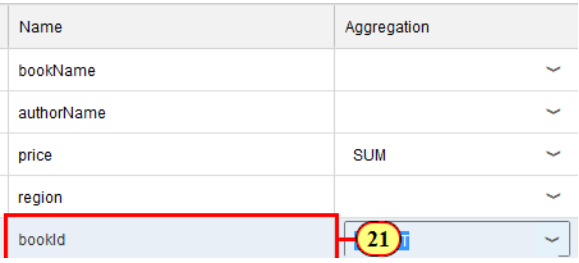
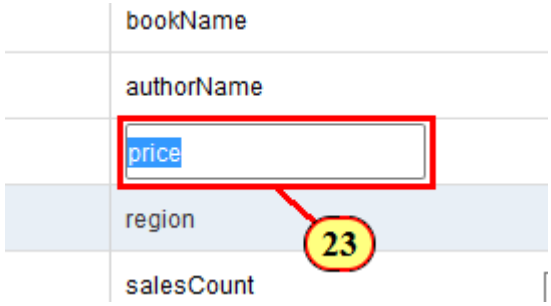
8.2 Create a Calculation View

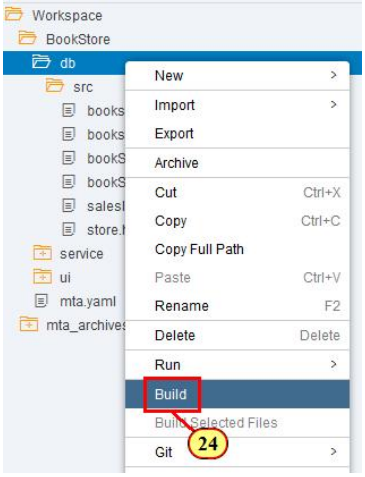
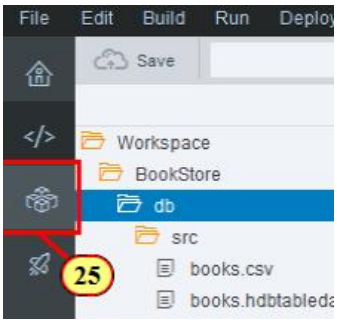
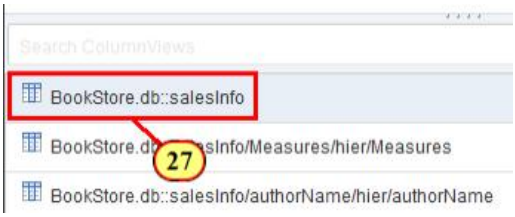
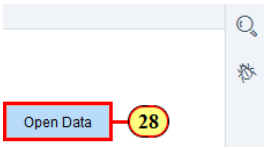
In this step, you will learn how to add a calculation view to your SAP HANA database module and how to display an aggregated view of your data.

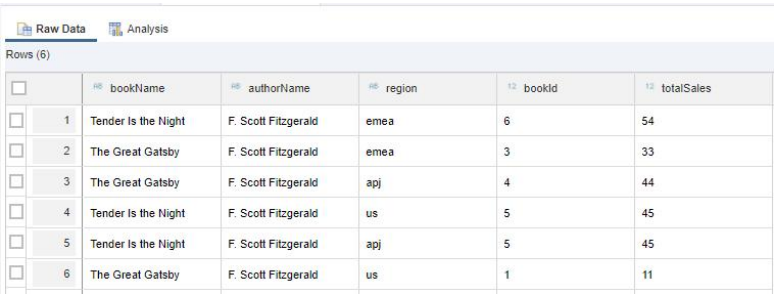
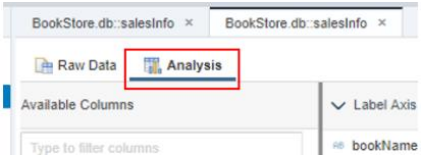
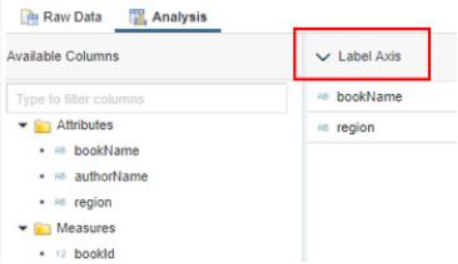
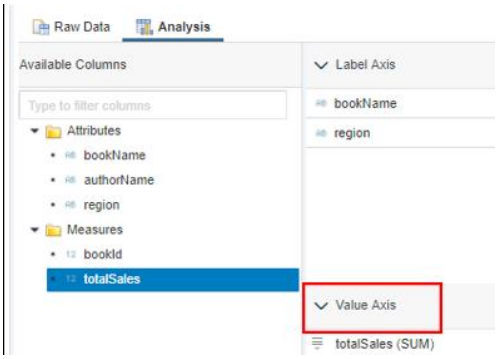
Explanation	Screenshot
1. Go to **Bookstore > db**. Right-click **src** and select **New > Calculation View**.	
2. Name the Calculation View **SalesInfo**. 3. Click Create.	
4. Click to create a new Join and then click on the canvas.	
5. Click .	
6. Enter **book** in the text field.	

Explanation	Screenshot
7. Click  on both rows to select the 2 tables. 8. Click  .	
9. Click **bookId** in the **Book** entity and drag to **bookId** in the **BookSales** entity.	
10. Click  .	
11. From the **Cardinality** dropdown list, select **1..n**	
12. In the **Join_1** element, click  and drag to the **Aggregation** element above.	

Explanation	Screenshot
13. Open the **Mapping** tab of the **Join_1** element.	
14. Drag the following elements to the **Output Column**: · bookId · bookName · authorName · price · region	
15. Click **Columns**.	
16. Clear the **Aggregation** field for the **price** and **bookId** elements.	
17. Click  .	

Explanation	Screenshot
18. Drag all elements in the data source column to the **Output Column** column. Click **Save**.	
19. Open the **Columns** tab of the **Aggregation**.	
20. Change the **bookId** aggregation to COUNT.	
21. Click bookId.	
22. Rename **bookId** to **salesCount**.	
23. Rename price to **totalSalesAmount**. Click **Save**.	

Explanation	Screenshot
24. Under **BookStore**, right-click **db** and select **Build**.	
25. Once the build finishes, click  to open the Database Explorer perspective. In the Database Explorer you can view information about your database's catalog objects and execute queries.	
26. Open the database and select **Column Views**.	
27. Click **salesInfo**.	
28. Click Open Data.	

Explanation	Screenshot
29. The sales information contained in the database is displayed. It shows the book sales information by region.	
30. Open the **Analysis** tab.	
31. Drag the **bookName** and **region** attributes to the **Label Axis** section.	
32. Drag the **totalSalesAmount** to the **Value Axis** section. The data is displayed in a chart.	
33. Choose the desired chart type.	
34. Click to export the chart or the data.	

Summary

You have completed the exercise!

You are now able to:

- Add a Calculation View to your SAP HANA database module.
- Display an aggregated view of your data.

9 COPYRIGHT