UX410

Developing SAP Fiori UIs

EXERCISES AND SOLUTIONS

Course Version: 20
Course Duration: 8 Hours 20 Minutes

SAP Copyrights, Trademarks and Disclaimers

Typographic Conventions

American English is the standard used in this handbook.

The following typographic conventions are also used.

This information is displayed in the instructor's presentation	
Demonstration	
Procedure	
Warning or Caution	
Hint	
Related or Additional Information	
Facilitated Discussion	
User interface control	*Example text*
Window title	*Example text*

Contents

Unit 4
Exercise 1

Create a Prototype with SAP BUILD

Note:
SAP Build is a cloud service. Due to the nature of cloud software, *step* procedures and names such as *buttons* or *fields* may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

Business Scenario

In this exercise, you create a prototype with **SAP BUILD** using the **Clone** function.

1. Create a new account at https://www.build.me.
2. Log in to your newly created account.

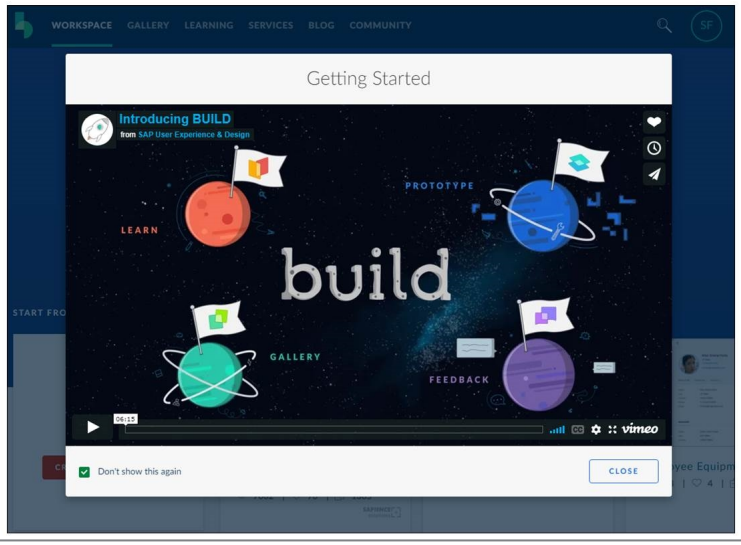


Figure 7: Getting Started

You are now logged into SAP BUILD.

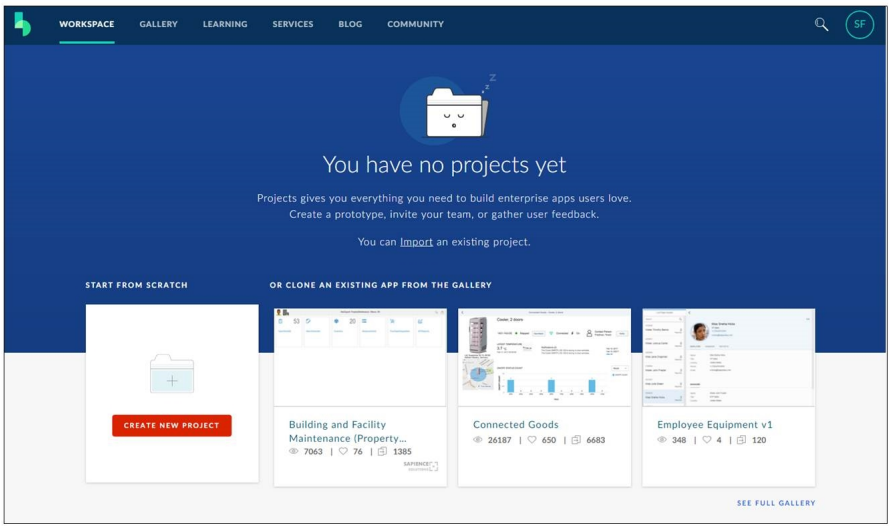


Figure 8: Logged into SAP Build

3. Clone the template project *Manage Products* from the gallery.
4. Open the cloned project to familiarize yourself with the design of the newly-created prototype.

Unit 4
Solution 1

Create a Prototype with SAP BUILD

Note:
SAP Build is a cloud service. Due to the nature of cloud software, *step* procedures and names such as *buttons* or *fields* may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

Business Scenario

In this exercise, you create a prototype with **SAP BUILD** using the **Clone** function.

1. Create a new account at https://www.build.me.
   a) Open `Google Chrome` and insert **`https://www.build.me`** into the address bar.

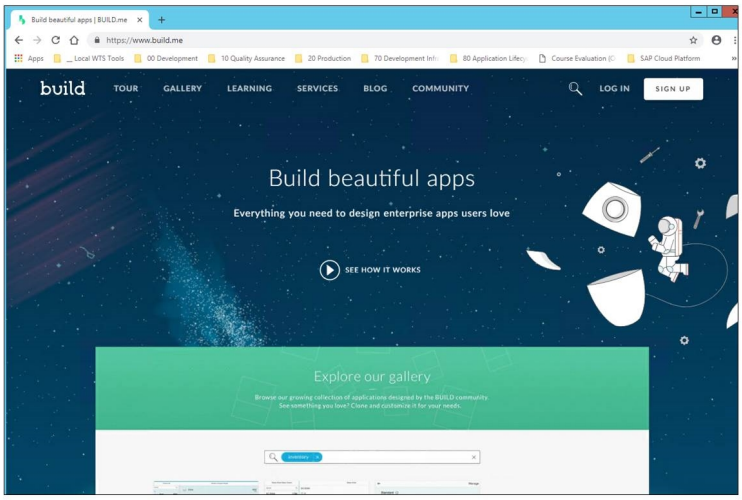


Figure 1: Big Beautiful Apps

   b) Choose the button labelled *SIGN UP.*

   c) Enter an email address in the email address field and choose *Proceed*.

Note:
To activate the trial account, you need to access the entered email account. If you do not have an email account that you can access during training, you can create one, for example at https://www.google.com/gmail/.

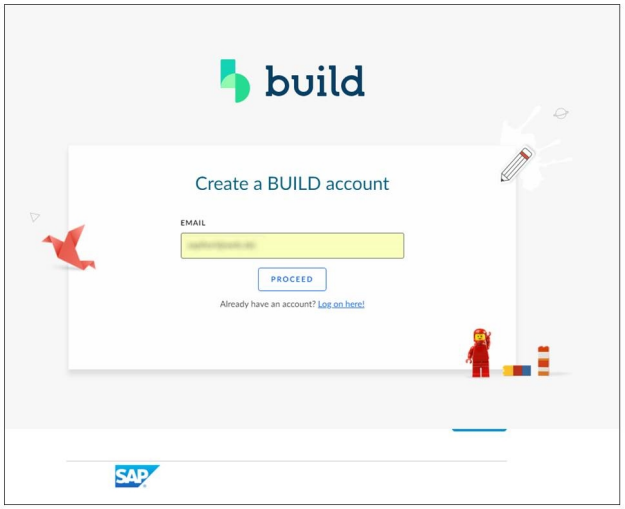


Figure 2: Create a Build Account

d) Fill in the fields using a valid email address to receive an activation link and choose *Register*.

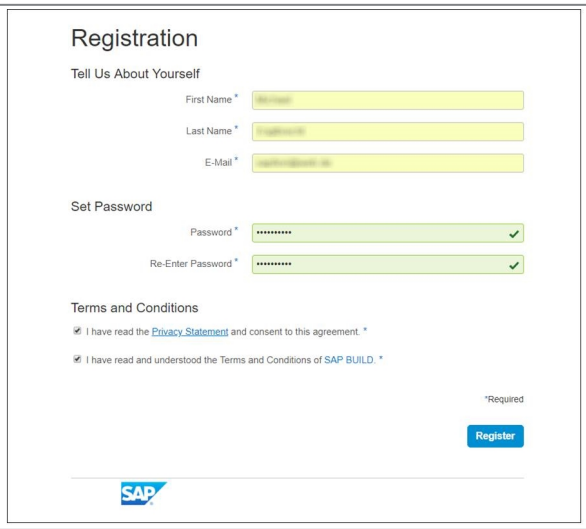


Figure 3: Registration SAP Build

e) Open the activation link you receive to your submitted email address. This activates your account.

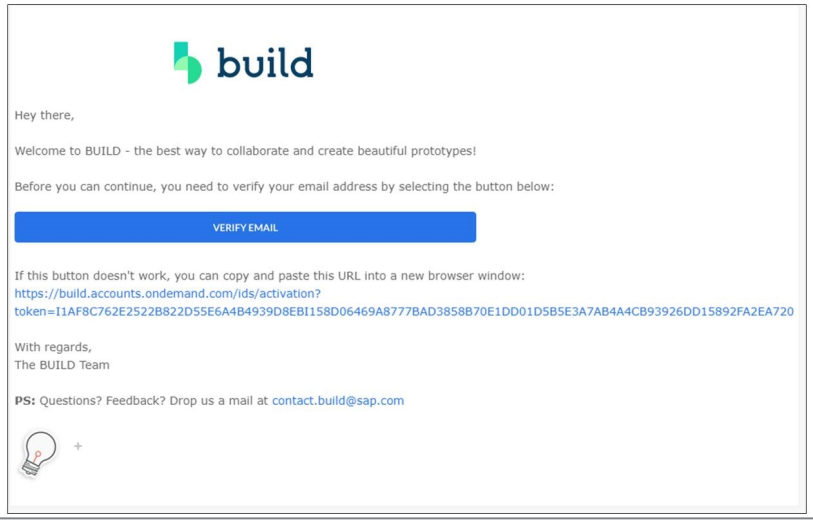


Figure 4: Verification Email

f) Choose *Continue*.

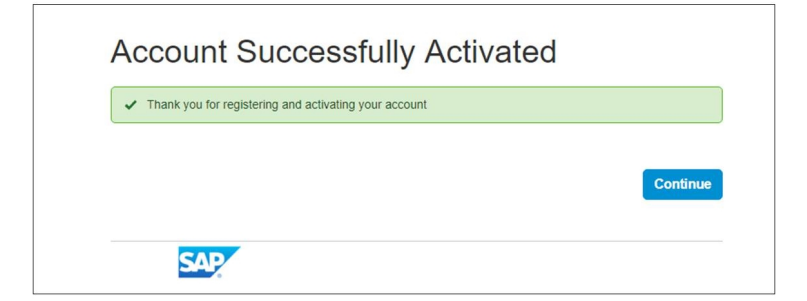


Figure 5: Account Activated

2. Log in to your newly created account.

a) Choose *login*.

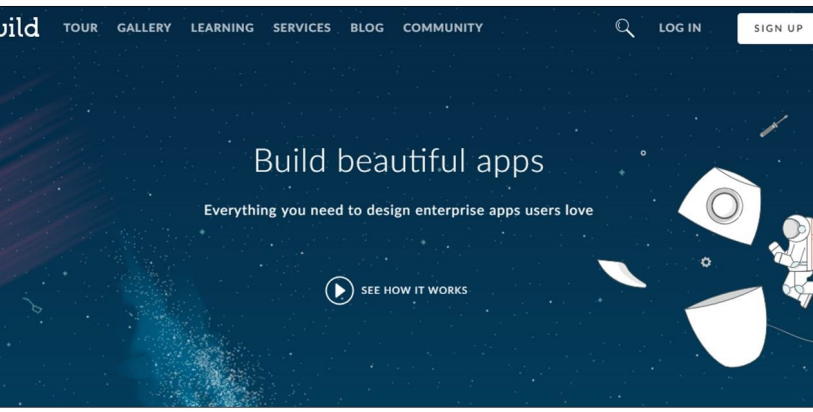


Figure 6: Login

b) Enter your email address and choose *Proceed*.

c) Optionally, select the *Checkbox* to 'Don't show this again' and choose *Close*.

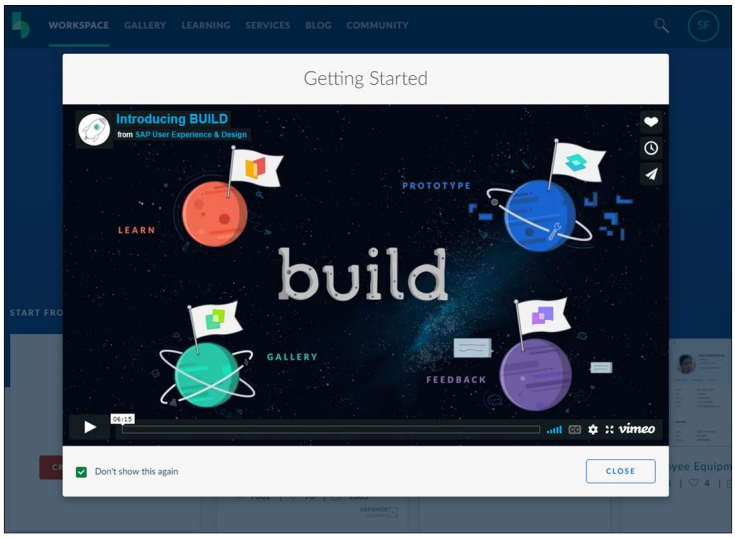


Figure 7: Getting Started

You are now logged into SAP BUILD.

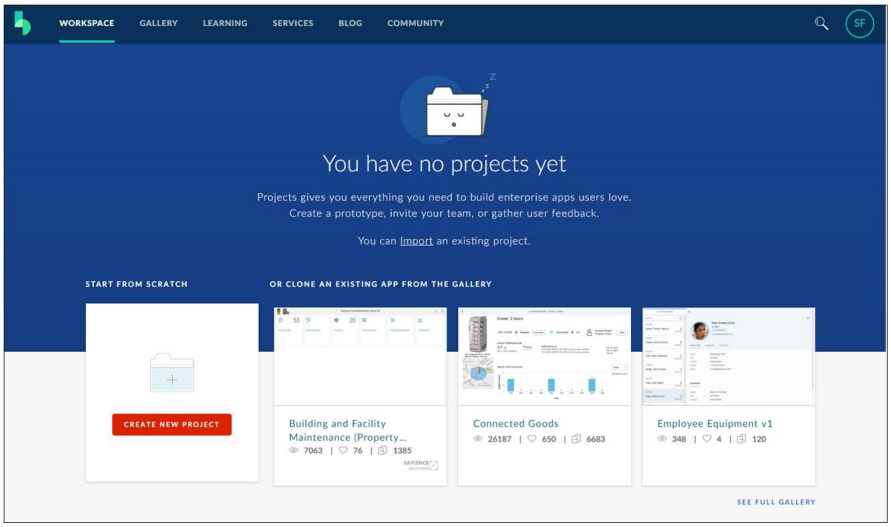


Figure 8: Logged into SAP Build

3. Clone the template project *Manage Products* from the gallery.
   a) Choose the button *See Full Gallery*.

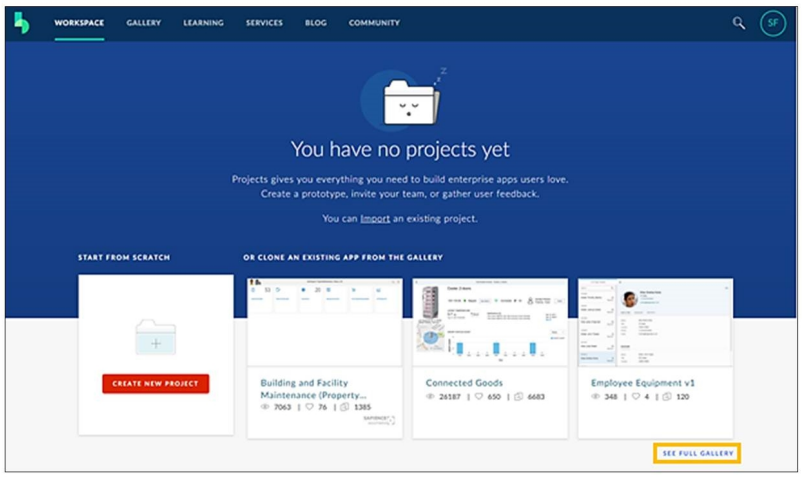


Figure 9: See Full Gallery

b) Browse the Gallery.

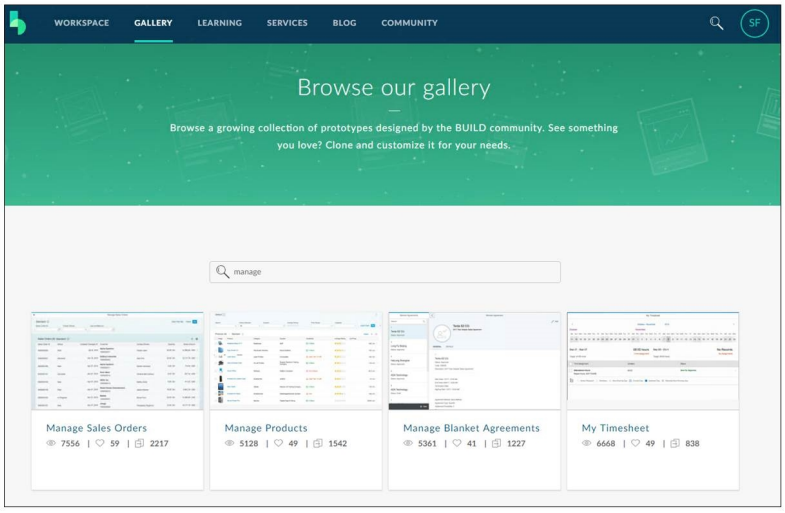


Figure 10: Browse our Gallery

c) Search for the prototype of *Manage products* by entering **`Manage`** into the search field.

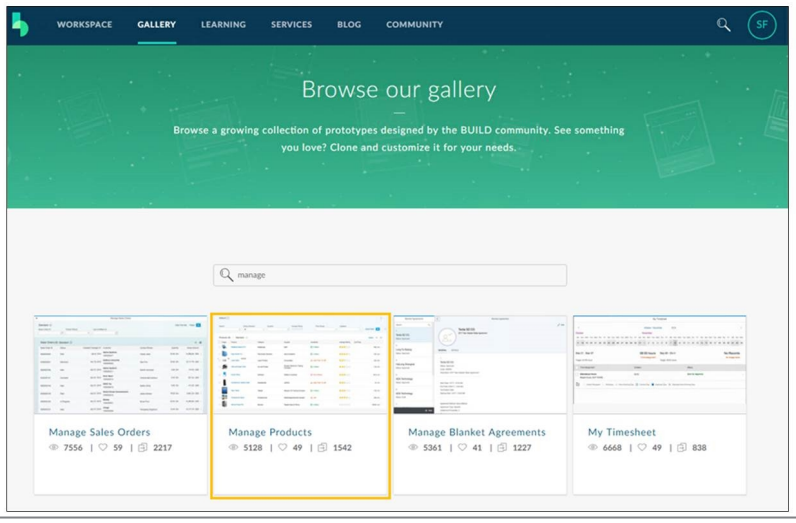


Figure 11: Template for Manage Products

d) Choose the template *Manage Products*.

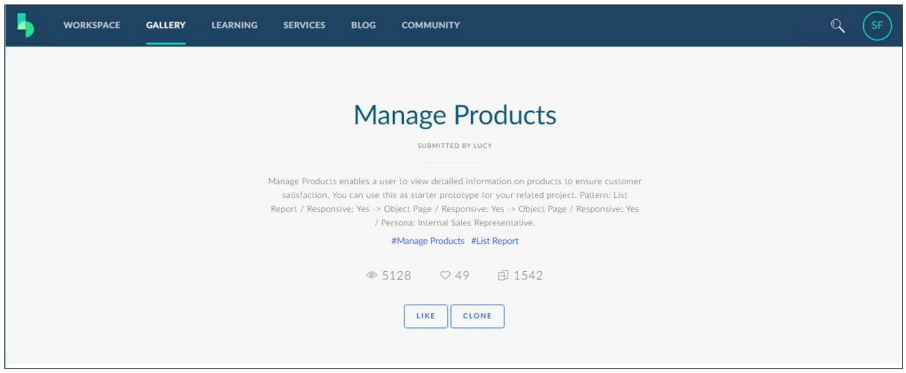


Figure 12: Manage Products

e) Choose the *Clone* button.

f) Choose *Clone to a new project* from the *CLONE TO* options.

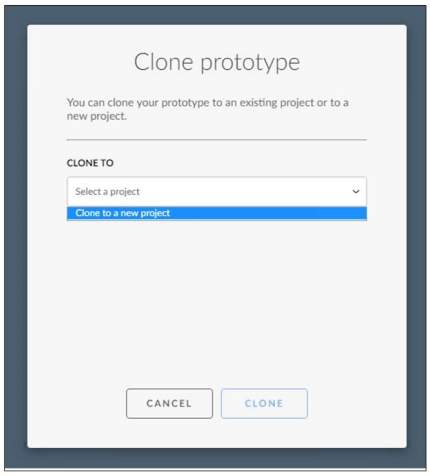


Figure 13: Clone Prototype

g) Choose the *CLONE* button to start the clone process.

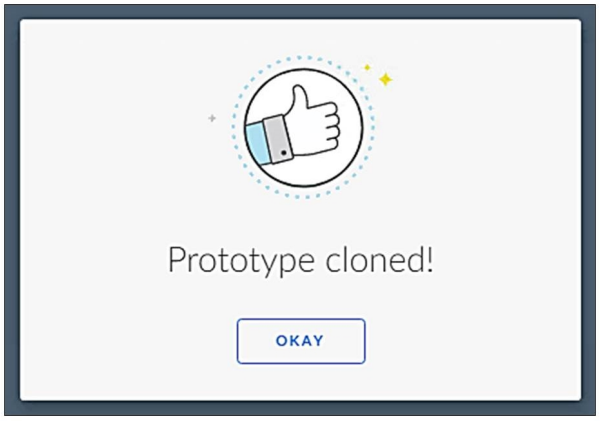


Figure 14: Prototype Cloned

h) After successfully cloning the template, choose *OK*.

4. Open the cloned project to familiarize yourself with the design of the newly-created prototype.

a) To open the newly-created prototype choose *Title*.

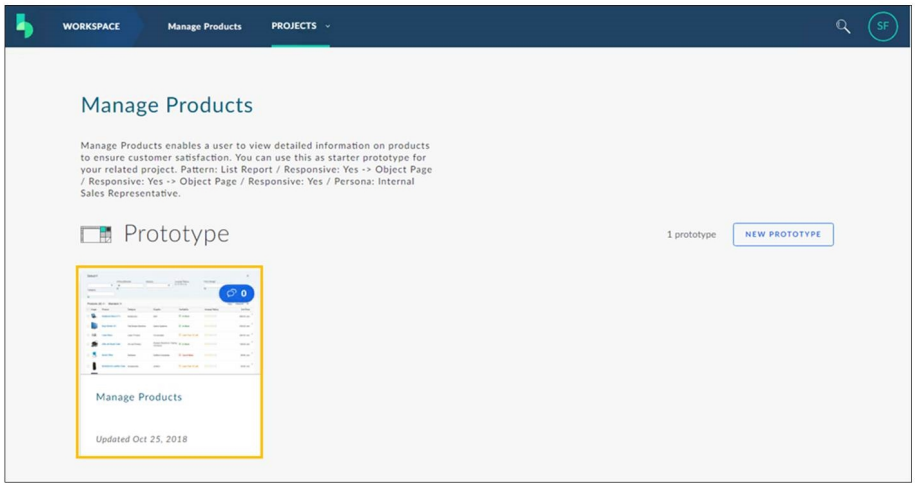


Figure 15: Manage Products Prototype

b) Open *Page 1* in *Edit* mode to investigate the new prototype.

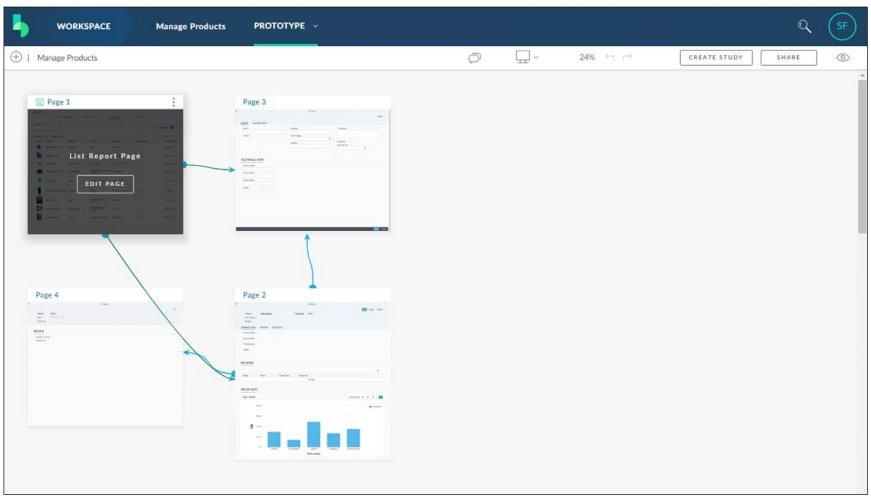


Figure 16: Edit Page 1

c) Open the *Data Editor* to take a look at the implementation.

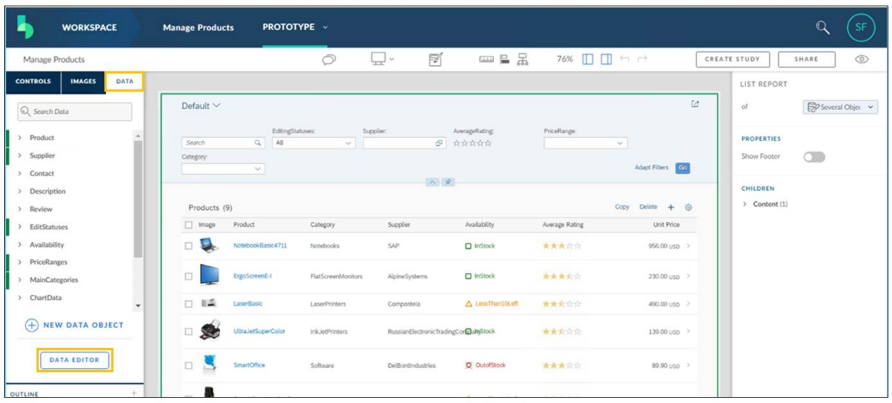


Figure 17: Data Editor

d) After the review, select the *Preview* button in the upper toolbar to start a preview of the cloned project.

Figure 18: Preview Button

e) Explore the application.

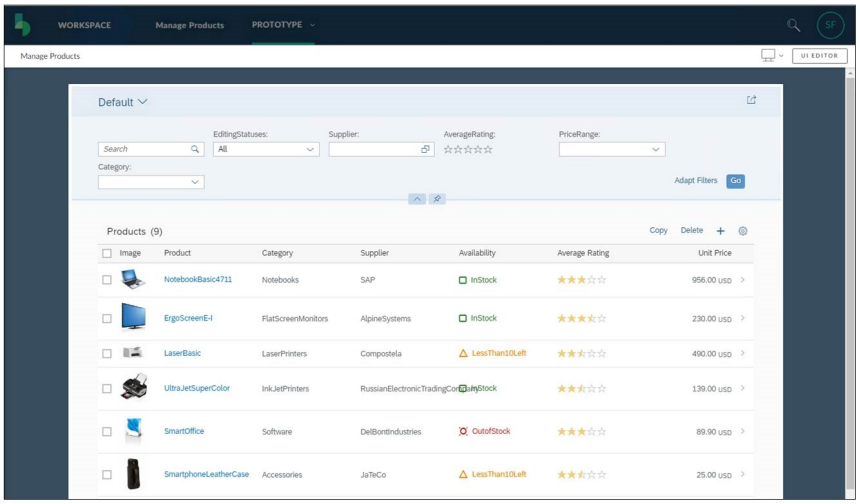


Figure 19: Application

f) Choose the *UI EDITOR* button in the upper toolbar to exit preview mode.

Figure 20: UI Editor Button

g) You finished the exercise successfully. Log out from `SAP BUILD`.

Figure 21: Logout

Unit 6
Exercise 2

Create a Trial Account for SAP Cloud Platform and Start the SAP Web IDE

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, step procedures and naming of fields and buttons may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

Objective:

Create a trial account for the SAP Cloud Platform. Start the *SAP Web IDE*.

Business Example

The *SAP Web IDE* is used for the exercises that follow. In this exercise, you access the *SAP Web IDE* via a trial account on the SAP Cloud Platform:

1. Create a trial account for the SAP Cloud Platform.

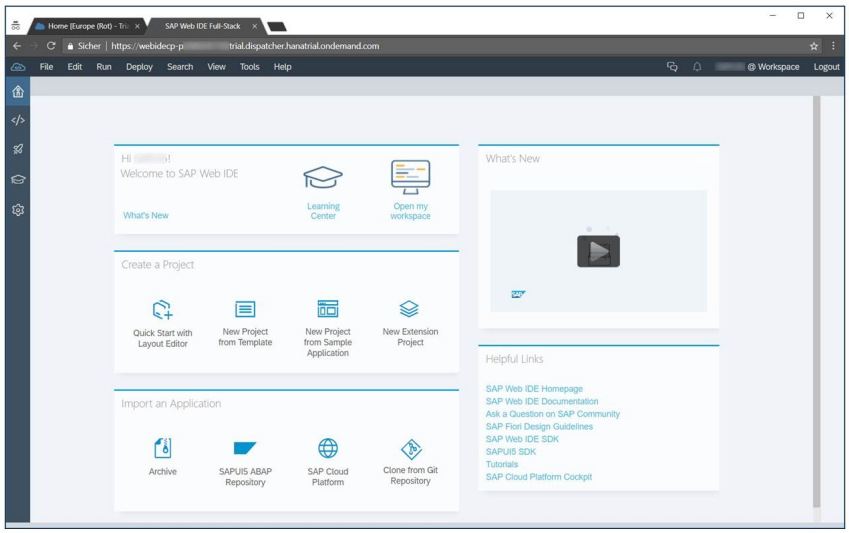


Figure 22: Trial Account Window

2. Use the account created to start the *Full Stack SAP Web IDE* using this account.

Create a Trial Account for SAP Cloud Platform and Start the SAP Web IDE

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, step procedures and naming of fields and buttons may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

Objective:

Create a trial account for the SAP Cloud Platform. Start the *SAP Web IDE*.

Business Example

The *SAP Web IDE* is used for the exercises that follow. In this exercise, you access the *SAP Web IDE* via a trial account on the SAP Cloud Platform:

1. Create a trial account for the SAP Cloud Platform.

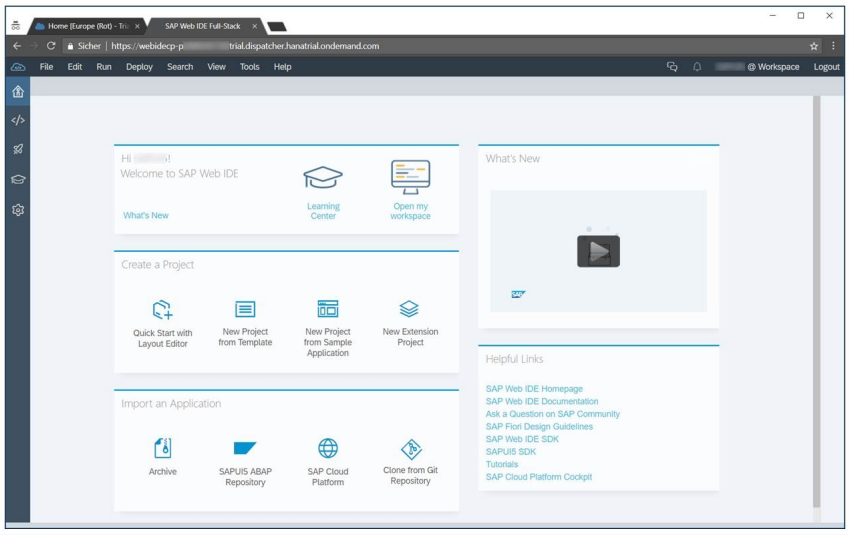


Figure 22: Trial Account Window

a) Open the browser window and enter the following URL: https://account.hanatrial.ondemand.com.

b) Choose *Register* on the web page to create a new account.

c) At a minimum, enter your last name and e-mail address. Assign a password for your account.

Note:
To activate the trial account, you need to access the entered email account. If you do not have an email account that you can access during training, you can create one, for example at https://www.google.com/gmail/.

d) Enter your preferences for contact by SAP and select the two *check boxes* to accept *SAP's Privacy Statement* and *Terms and Conditions*.

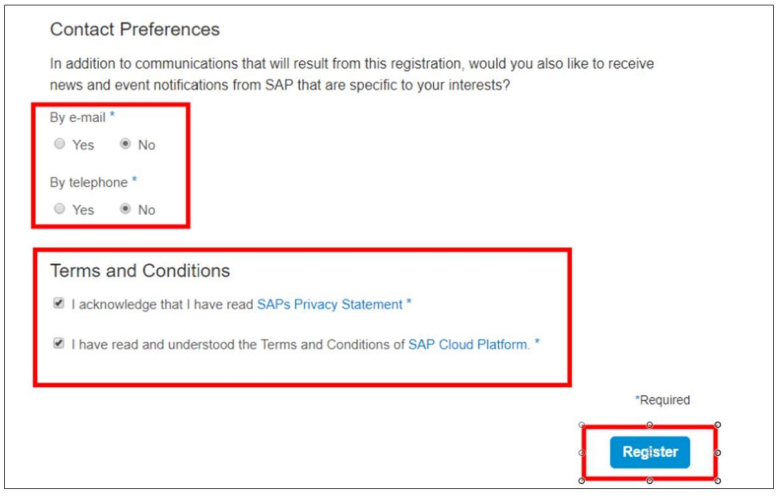


Figure 23: Contact Preferences

e) Confirm your input with the *Register* button.

f) Select the correct images from a *captcha* dialog that appears and choose *Verify*.

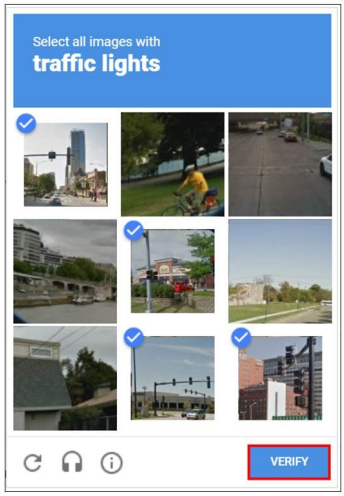


Figure 24: Select Captcha Images

g) Close the window that confirms your registration.

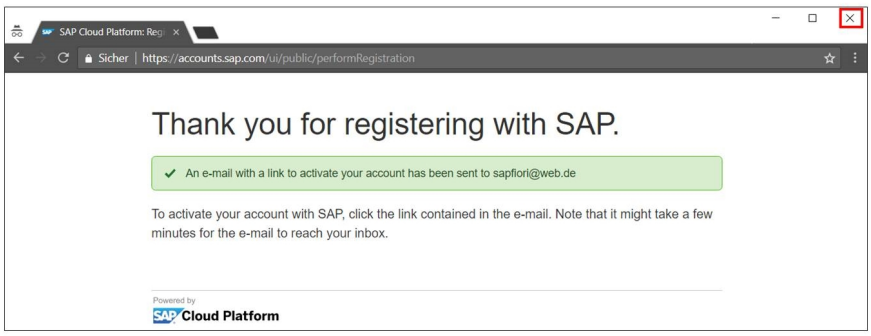


Figure 25: Thank You for Registering

2. Use the account created to start the *Full Stack SAP Web IDE* using this account.

   a) Check your registered email account and select *Click here to activate your account*. The link opens to activate your free trial.

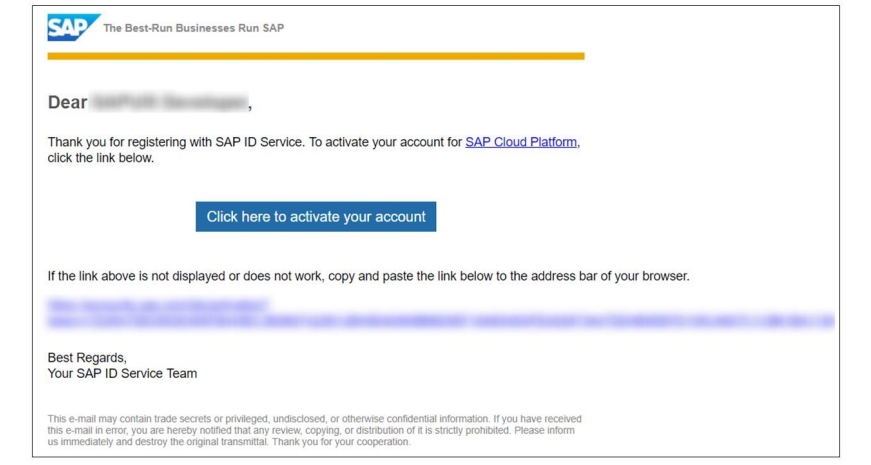


Figure 26: Activation Link

   b) To close the confirmation window, choose *Continue*.

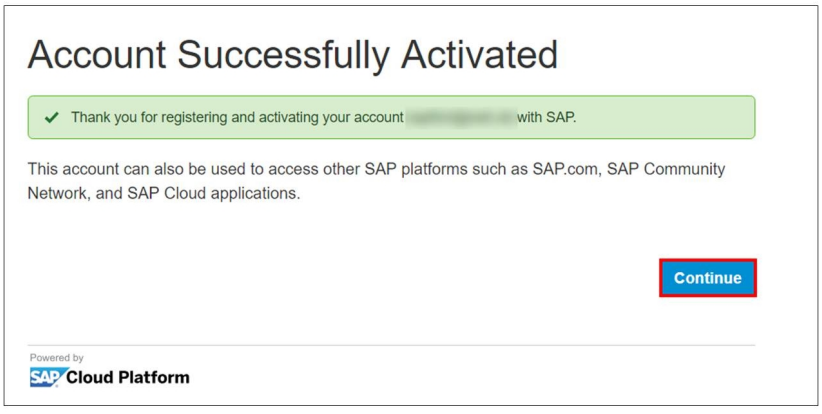


Figure 27: Activate and Continue

   c) Check that you are automatically logged into your new account and confirm any popups.

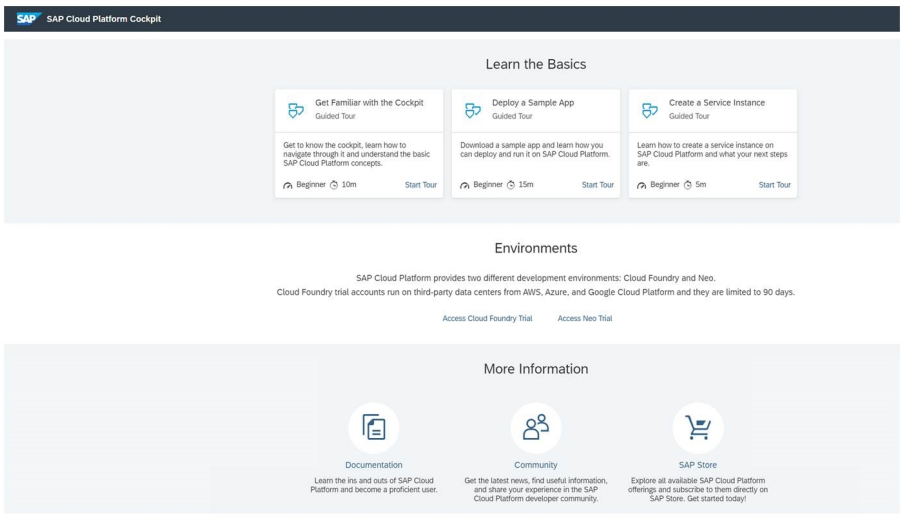


Figure 28: Access New Trial

d) Scroll down to the environments and choose *Access New Trial*.

e) To start the *SAP Web IDE*, select the *Neo Trial* hyperlink.

f) Choose *Services* on the *SAP Cloud Platform Cockpit*.

g) In the text input field, enter **`web`** and, in the DevOps screen area, choose the *SAP Web IDE Full-Stack* tile.

h) Select the *Go to Service* link to start the *SAP Web IDE* and display in the browser. Look for your trial account window.

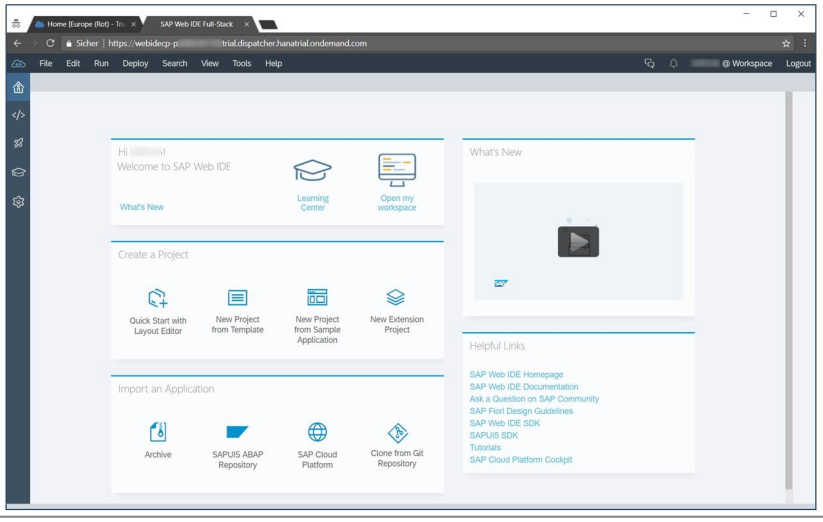


Figure 29: Trial Account Window

Unit 6
Exercise 3

Create a Simple SAPUI5 Application with the help of the Layout Editor

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, *step* procedures and names such as *buttons* and *fields* may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises. The graphical layout editor of the SAP Web IDE is only available in Google Chrome.

Objective: Create and test a simple SAPUI5 application in the SAP Web IDE. Use the graphical layout editor to edit the application views.

Business Scenario:

In this exercise, you create an SAPUI5 project with an XML view in the SAP Web IDE. In the generated view, you enter the text **"Hello World!"** using a **Text UI** element. You also set the title of the displayed page to **SAP Web IDE**. To do this, you edit the view with the graphical Layout Editor. You then test the application in the browser:

1. Open the development perspective of the SAP Web IDE.
2. In the *SAP Web IDE* create a new project. Use the following data and replace ## with your student number:

Property	Value
Project Name:	helloworld
Namespace:	student##.sap.training
View type:	XML
View name:	Main
SAPUI5 version:	1.60

3. Open the *Main* view in the graphical Layout Editor generated by the wizard. Add a *Text* UI element to the view and set its **Text** property to the value `Hello World`. Also set the title of the page to *SAP Web IDE*.
4. Adjust the bootstrap script so that SAPUI5 is loaded from a Content Delivery Network (CDN) externally and no longer loaded from the web server locally. Use the following URL https://sapui5.hana.ondemand.com/resources/sap-ui-core.js.

Note:
The relative paths for loading the SAPUI5 are different in the cloud environment than on an ABAP server. In the next exercise, you deploy your SAPUI5 application on an ABAP server and test it there. You will load the SAPUI5 externally via a CDN, so that the application can run without path adjustment in both the cloud environment and on an ABAP server.

Note:
The latest SAPUI5 version is now loaded via the URL you defined. This procedure can have an impact on the stability of an application in new releases, so only follow this procedure for test purposes. You can load a specific SAPUI5 version using a version URL, for example https://sapui5.hana.ondemand.com/1.38.8/resources/sap-ui-core.js

5. Test your application by starting it from the *SAP Web IDE*.

Note:
In some environments the pop-up blocker of your browser may prevent the start of the application. Please allow pop-ups so the application can run.

Create a Simple SAPUI5 Application with the help of the Layout Editor

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, *step* procedures and names such as *buttons* and *fields* may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises. The graphical layout editor of the SAP Web IDE is only available in Google Chrome.

Objective: Create and test a simple SAPUI5 application in the SAP Web IDE. Use the graphical layout editor to edit the application views.

Business Scenario:

In this exercise, you create an SAPUI5 project with an XML view in the SAP Web IDE. In the generated view, you enter the text **"Hello World!"** using a **Text UI** element. You also set the title of the displayed page to **SAP Web IDE**. To do this, you edit the view with the graphical Layout Editor. You then test the application in the browser:

1. Open the development perspective of the SAP Web IDE.
   a) Start the *SAP Web IDE*.
   b) Click on the < > icons and the development perspective will appear.

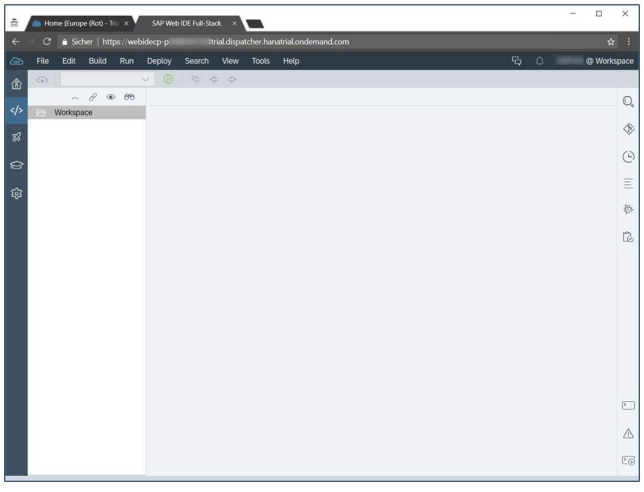


Figure 30: Development Perspective

2. In the *SAP Web IDE* create a new project. Use the following data and replace ## with your student number:

Property	Value
Project Name:	helloworld
Namespace:	student##.sap.training
View type:	XML
View name:	Main
SAPUI5 version:	1.60

a) Choose the following path *File → New → Project from Template* from the *SAP Web IDE* menu.

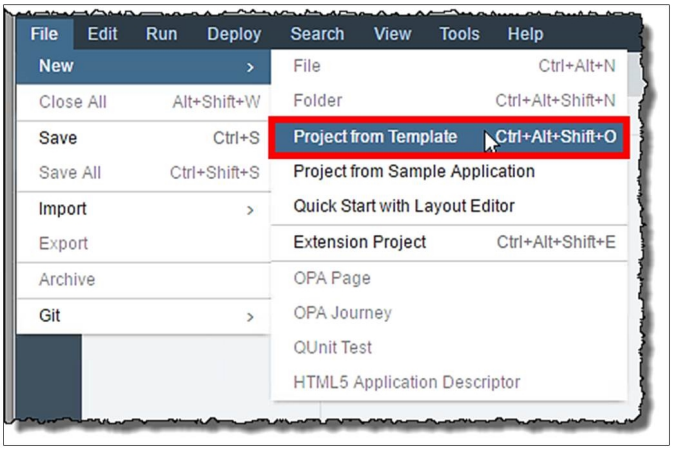


Figure 31: Follow Path

b) Select the *SAPUI5 Application Project* template, and choose *Next*.

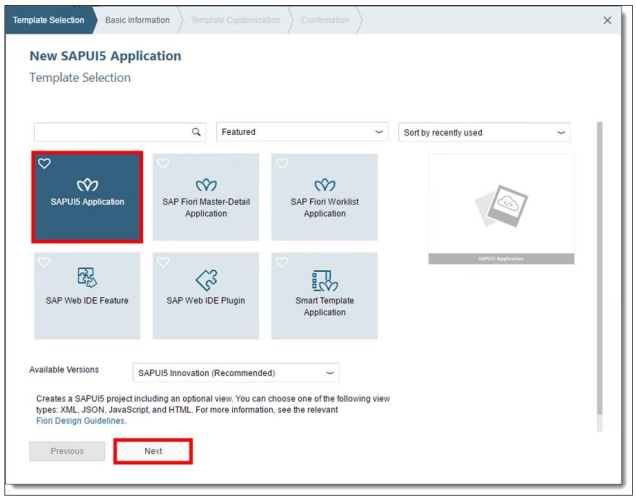


Figure 32: SAPUI5 Application Template

c) Enter **`helloworld`** in the *project name* field and **`student##.sap.training`** in the *namespace* field. Replace the group number **`00`** with your assigned group number and choose the *next*.

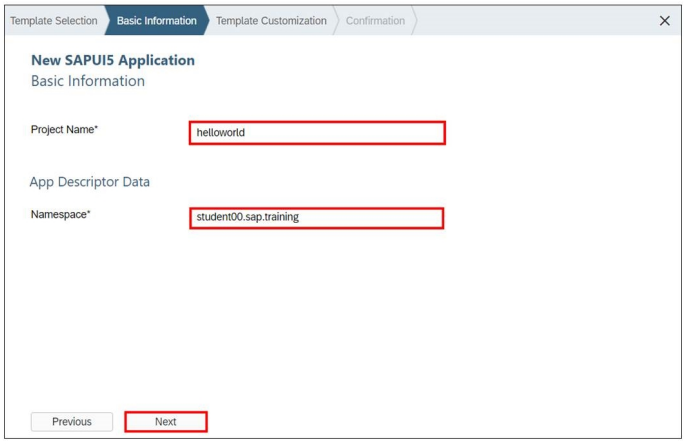


Figure 33: New SAPUI5 Application

d) Choose XML as the *View Type* to be generated. Enter **`Main`** as the name of the view. Choose *next*.

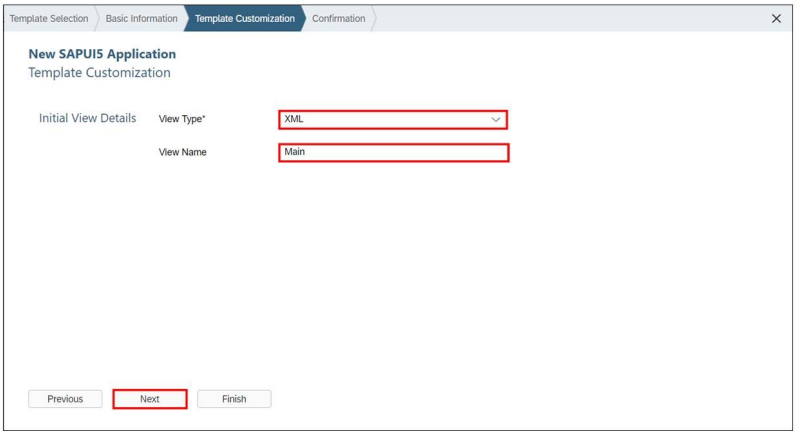


Figure 34: XML View Type

e) Close the wizard by choosing *Finish*.

3. Open the *Main* view in the graphical Layout Editor generated by the wizard. Add a *Text* UI element to the view and set its **`Text`** property to the value `Hello World`. Also set the title of the page to *SAP Web IDE*.

   a) Open the *Main.view.xml* in the layout editor by choosing: *Open With* → *Layout Editor*.

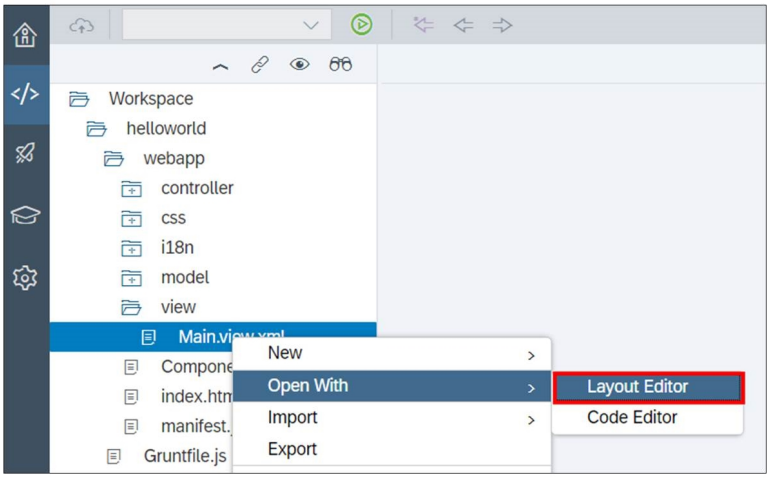


Figure 35: Main View Layout Editor

b) Expand the display section on the *Control tab* and drag and drop the *Text* UI element from this section to the *Main view*.

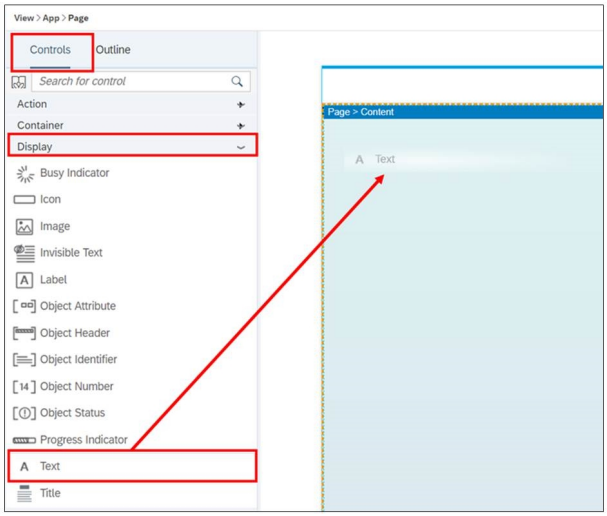


Figure 36: Text UI Path

c) In the *Properties Pane* enter the value **`Hello World!`** for the text property of the *Text* UI element.

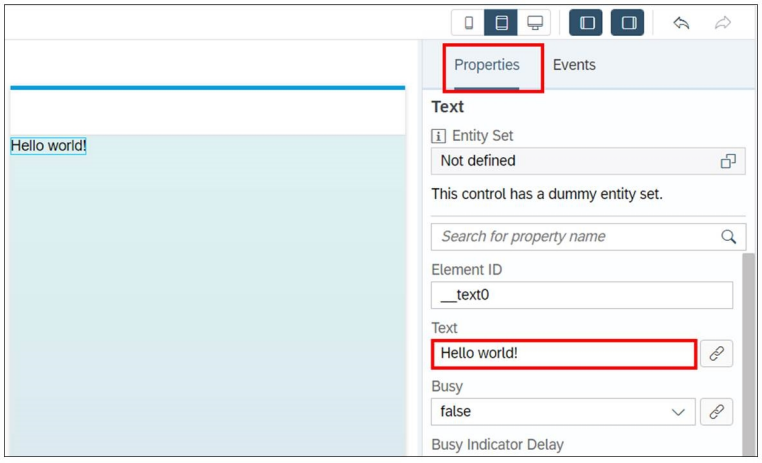


Figure 37: Hello World

d) Select the associated *Page UI* element in the *Toolbar* to adjust the title of the page.

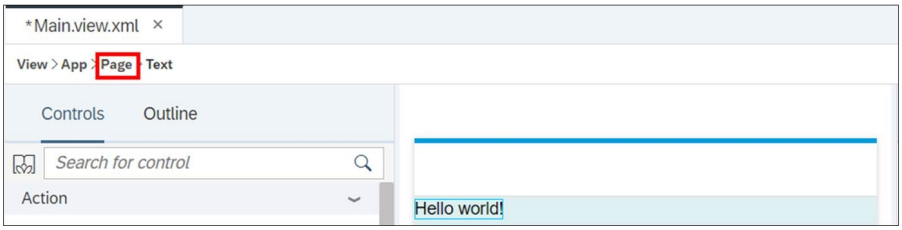


Figure 38: Page Link on Toolbar

e) In the *Properties Pane*, enter **`SAP Web IDE`** for the *Title* property of the *Page* UI element and *Save* your changes.

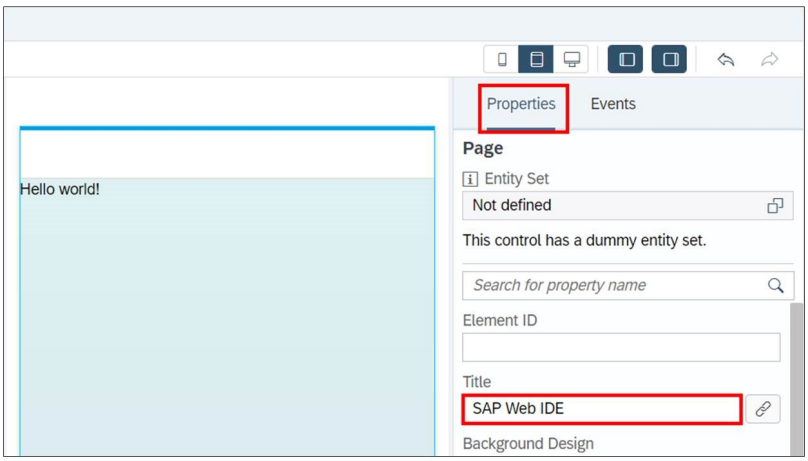


Figure 39: Property Pane Title SAP Web IDE

4. Adjust the bootstrap script so that **SAPUI5** is loaded from a Content Delivery Network (CDN) externally and no longer loaded from the web server locally. Use the following URL https://sapui5.hana.ondemand.com/resources/sap-ui-core.js.

> Note:
> The relative paths for loading the SAPUI5 are different in the cloud environment than on an ABAP server. In the next exercise, you deploy your SAPUI5 application on an ABAP server and test it there. You will load the SAPUI5 externally via a CDN, so that the application can run without path adjustment in both the cloud environment and on an ABAP server.

a) Open the *index.html* file. In the bootstrap script, change the value of the *src* attribute to the following URL: https://sapui5.hana.ondemand.com/resources/sap-ui-core.js

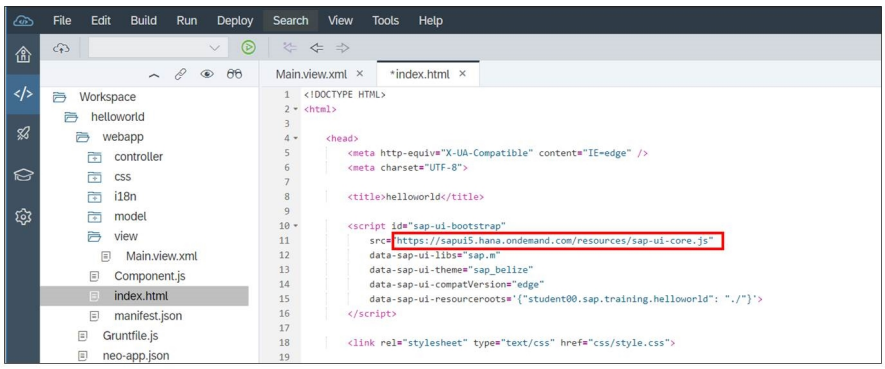


Figure 40: Change Source

b) Save your changes.

> Note:
> The latest SAPUI5 version is now loaded via the URL you defined. This procedure can have an impact on the stability of an application in new releases, so only follow this procedure for test purposes. You can load a specific SAPUI5 version using a version URL, for example https://sapui5.hana.ondemand.com/1.38.8/resources/sap-ui-core.js

5. Test your application by starting it from the *SAP Web IDE*.

> Note:
> In some environments the pop-up blocker of your browser may prevent the start of the application. Please allow pop-ups so the application can run.

a) Select the index.html file and start the application by choosing the *Run* button in the toolbar of the *SAP Web IDE*.

b) Check whether the adjustments you made on the *Main View* appear in the preview screen.

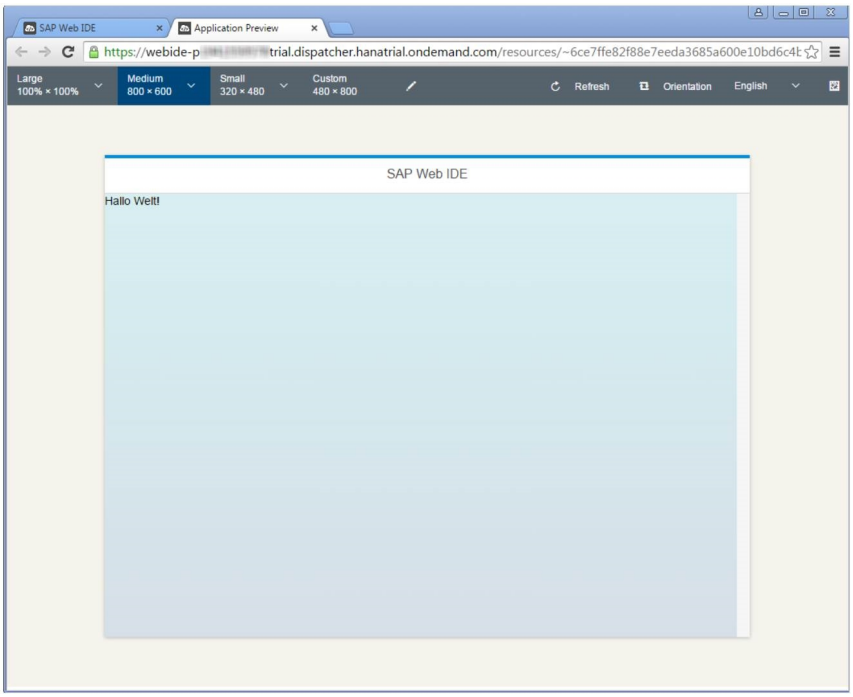


Figure 41: Preview Screen Hello World

Unit 6
Exercise 4

Set up the SAP Cloud Connector

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, step procedures and names of fields and buttons may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

Business Scenario

Set up the SAP Cloud Connector to connect the SAP Cloud Platform to your on-premise SAP System.

Task 1: Set up the SAP Cloud Connector.

1. Start the *SAP Cloud Connector Administration* site by browsing in Chrome to the URL https://localhost:8443 or choose the bookmarked Local WTS Tools folder.
2. Complete the *SAP Cloud Connector* login dialog box.
3. Map the virtual host from the destination to the ABAP training system.
4. Specify the URL paths to be called in the training system.

Task 2: Run the Initialize Course script

For several exercises, you are required to download files. Additional resources are also available.

1. To access the files and additional resources for the exercises in this course, run the Initialize Course script.

Unit 6
Solution 4

Set up the SAP Cloud Connector

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, step procedures and names of fields and buttons may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

Business Scenario

Set up the SAP Cloud Connector to connect the SAP Cloud Platform to your on-premise SAP System.

Task 1: Set up the SAP Cloud Connector.

1. Start the *SAP Cloud Connector Administration* site by browsing in Chrome to the URL https://localhost:8443 or choose the bookmarked Local WTS Tools folder.
   a) To start the *SAP Cloud Connector Administration* site, choose the *LocalWTSTools* from the bookmarks bar of Chrome.
   b) Choose *SAP Cloud Connector*.
   c) Accept warnings that the site is untrusted. The site is hosted locally without a valid signed certificate and this can be expected. Choose *Advanced* and *Proceed to localhost*.
2. Complete the *SAP Cloud Connector* login dialog box.
   a) Enter **`Administrator`** in the *User Name field.*
   b) Enter **`Welcome`** in the *Password* field.
   c) Choose *Login.*

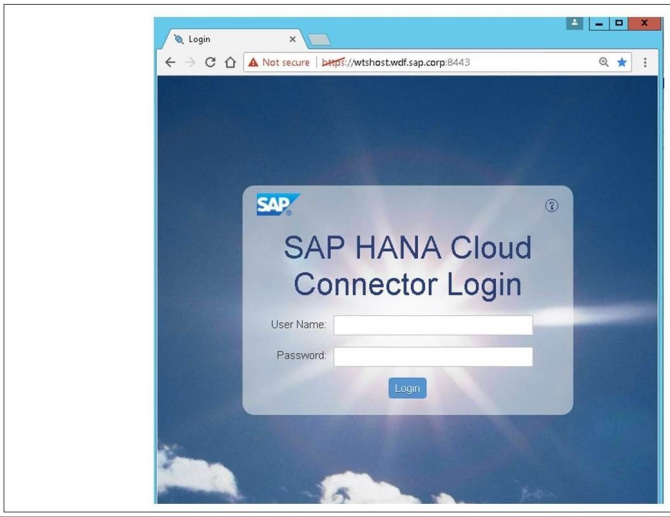


Figure 42: SAP Cloud Connector Login

d) Choose *Add Subaccount* and the *Add Subaccount* dialog box opens.

Figure 43: Add Sub Account

e) Choose **`Europe(Rot)-Trial`** as the region.

f) Enter your SCP account information.

g) Choose *Save* to confirm your values.

> Note:
> The Account Name is shown in the top left corner of the SAP Cloud Platform Cockpit.

3. Map the virtual host from the destination to the ABAP training system.

a) Choose the *Access Control* Tab.

b) Choose *Cloud To On-Premise* in the Cloud-Connector front end for your SAP Cloud Platform Account.

c) Choose *Add* on the toolbar in the *Mapping Virtual To Internal System* section of the screen.

d) Select ABAP System in the *Back-end Type* field and choose *Next*.

e) Select HTTPS in the protocol field.

f) Enter **`fsdhost.wdf.sap.corp`** in the *Internal Host* field and **`44310`** in the *Internal Port* field and then choose *Next*.

g) Enter **`fsd.virtual`** in the *Virtual Host* field and **`8000`** in the *Virtual Port* field.

h) Choose **`None`** in the *Principal Type* field, and then *Next*.

i) Choose *Use Virtual Host* in the *Host In Request Header* field and choose *Next*.

Note:
You can also optionally enter a description.

j) Choose *Next* and *Finish* and exit the wizard.

4. Specify the URL paths to be called in the training system.
   a) Select the created entry in the *Mapping Virtual To Internal System* section of the screen.
   b) Choose to *Add* on the *Resource Accessible On fsd.virtual:8000* section of the screen.
   c) Enter **`/sap`** in the *URL Path* field, and choose the *Path and all sub-paths* radio button and *Save*.

Task 2: Run the Initialize Course script

For several exercises, you are required to download files. Additional resources are also available.

1. To access the files and additional resources for the exercises in this course, run the Initialize Course script.
   a) In your training view, choose *Start* → *More*.
   b) Choose *Initialize UT4.2*.
      The initialization script starts. It may take several seconds to finish.
   c) Choose *OK* when the information dialog is displayed.
   d) Open File Explorer to navigate to the resources and choose: *S:/Courses/ UX410_20*.

Unit 6
Exercise 5

Deploy an SAPUI5 Application via the SAP Web IDE on an ABAP Server

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, naming of fields and buttons as well as steps may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

In this exercise, you deploy the SAPUI5 application you developed in the last exercise, onto an ABAP server from the SAP Web IDE. To do this, the ABAP system used for this training course must be made accessible for your SAP Cloud Platform account through the SAP HANA Cloud Connector. The connection to your account is established through a destination, which you create in the SAP Cloud Platform Cockpit.

1. In the SAP Cloud Platform cockpit, create a destination with the following properties:

   Table 1: Destination Table Configuration

Configuration	Value
Name	FDS_100
Type	HTTP
Description	FSD
URL	http://fsd.virtual:8000
Proxy Type	OnPremise
Authentication	NoAuthentication

   Table 2: Add These Properties to the Destination

Properties	Value
WebIDEEnabled	true
WebIDEUsage	dev_abap,ui5_execute_abap,bsp_execute_abap,odata_abap

2. Deploy the SAPUI5 application you developed in the previous exercise from the SAP Web IDE to the ABAP system used in the training course. To do this, create a new application called **`ZUI5_WEBIDE_##`**, where ## stands for your group number. Assign the application to the existing package ZTRAIN_## (## = group number) and to the transport request that already exists for the training course.

3. Track the deployment process that now starts in the SAP Web IDE console.
4. After deployment, call up the application on the ABAP server using the following URL (replace ## with your group number):
   http://fsdhost.wdf.sap.corp:8000/sap/bc/ui5_ui5/sap/zui5_webide_##/index.html

Hint:
You can also start transaction SICF on the frontend server, search for your application. Select the found entry and choose from the context menu Test Service.

Unit 6
Solution 5

Deploy an SAPUI5 Application via the SAP Web IDE on an ABAP Server

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, naming of fields and buttons as well as steps may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

In this exercise, you deploy the SAPUI5 application you developed in the last exercise, onto an ABAP server from the SAP Web IDE. To do this, the ABAP system used for this training course must be made accessible for your SAP Cloud Platform account through the SAP HANA Cloud Connector. The connection to your account is established through a destination, which you create in the SAP Cloud Platform Cockpit.

1. In the SAP Cloud Platform cockpit, create a destination with the following properties:

Table 1: Destination Table Configuration

Configuration	Value
Name	FDS_100
Type	HTTP
Description	FSD
URL	http://fsd.virtual:8000
Proxy Type	OnPremise
Authentication	NoAuthentication

Table 2: Add These Properties to the Destination

Properties	Value
WebIDEEnabled	true
WebIDEUsage	dev_abap,ui5_execute_abap,bsp_execute_abap,odata_abap

a) To create a new destination, on the left side of the SAP Cloud Platform cockpit screen, choose *Connectivity* → *Destinations*.

b) In the screen that appears on the right, choose the *New Destination* button.

c) In the displayed form, enter the configuration values for the destination from the Destination Table.

> Note:
> The entered URL defines a virtual host and is therefore freely definable. In the SAP Cloud Connector, this virtual host is mapped to the ABAP system to be used.

d) Choose the *New Property* button to add further properties to the destination.

e) In the displayed input fields, enter **`WebIDEEnabled`** as the property name and **`true`** as the property value.

> Note:
> When maintaining the properties, pay attention to upper and lower case.
>
> The individual values for the `WebIDEUsage` property are separated by a comma only (no spaces).
>
> The figure does not show all the values to be entered for the `WebIDEUsage` property.
>
> To deploy an SAPUI5 application on an ABAP server, not all the entered values for the `WebIDEUsage` property are required. To keep things simple, all the values needed during the training course are included. For example, value `odata_abap` is not needed until you access the OData services of the ABAP system.

f) Choose the *New Property* button again to add a further property.

g) In the added input fields, enter **`WebIDEUsage`** as the property name and **`dev_abap,ui5_execute_abap,bsp_execute_abap,odata_abap`** as the property value.

h) Save your destination by choosing *Save*.

2. Deploy the SAPUI5 application you developed in the previous exercise from the SAP Web IDE to the ABAP system used in the training course. To do this, create a new application called **`ZUI5_WEBIDE_##`**, where ## stands for your group number. Assign the application to the existing package ZTRAIN_## (## = group number) and to the transport request that already exists for the training course.

a) In the context menu of your SAPUI5 project *helloworld*, choose the following path: *Deploy → Deploy to SAPUI5 ABAP Repository*.

b) Ensure the option *Deploy a new application* is selected on the displayed form, and select the system *FSD_100 - FSD* configured above from the drop-down menu. In the dialog box that is displayed, enter the user name and password for the ABAP system, remembering to replace ## with your group number. Confirm your entries by choosing *Sign in*.

c) On successful login, choose the *Next* button.

d) Enter **`ZUI5_WEBIDE_##`** (where ## is your group number), as the name of the new application and enter **`SAPUI5`** in SAP Web IDE as the description. Browse to select the existing package *ZTRAIN_##*, (where ## is your group number), and exit editing of the window by choosing *Next*.

e) On the next window, ensure the transport request relating to the training course is selected, and choose *Next*.

f) Close the wizard by choosing *Finish*.

3. Track the deployment process that now starts in the SAP Web IDE console.
   a) Select the following path from the SAP Web IDE menu: *View* → *Console*.
4. After deployment, call up the application on the ABAP server using the following URL (replace ## with your group number):
   http://fsdhost.wdf.sap.corp:8000/sap/bc/ui5_ui5/sap/zui5_webide_##/index.html

Hint:
You can also start transaction SICF on the frontend server, search for your application. Select the found entry and choose from the context menu Test Service.

Unit 6
Exercise 6

Import an SAP BUILD Prototype into SAP Web IDE

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, naming of fields and buttons as well as steps may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

Objective:

Import a prototype created with SAP Build into the SAP Web IDE.

Business Scenario

You import a prototype created with SAP Build to the SAP Web IDE to undertake further development.

You require the Prototype created in exercise 1.

Download a SAP Build Prototype to the SAP Web IDE.

1. Download the SAP Build prototype created in exercise 1 as a *SAP Web IDE* project.
2. Import the downloaded prototype into the *SAP Web IDE Full-Stack* edition. Use SAPBuildPrototype as the project name for the import.
3. Test the imported project with mock data. Create a runtime configuration with the name SAP Build prototype and use the html file with the name `testFLPServiceMockServer.html` located in the folder of your test project.

Unit 6
Solution 6

Import an SAP BUILD Prototype into SAP Web IDE

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, naming of fields and buttons as well as steps may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

Objective:

Import a prototype created with SAP Build into the SAP Web IDE.

Business Scenario

You import a prototype created with SAP Build to the SAP Web IDE to undertake further development.

You require the Prototype created in exercise 1.

Download a SAP Build Prototype to the SAP Web IDE.

1. Download the SAP Build prototype created in exercise 1 as a *SAP Web IDE* project.
   a) Open https://www.build.me in Google Chrome.
   b) Log in with your email address and choose *Proceed*.
   c) Enter your email address and password in the corresponding fields and log in.
   d) On log in, view your current projects in the *SAP Build* workspace.
   e) Open the *Manage Products* project by double-clicking the project tile or selecting the three dots of the tile, and choosing *Open* to see the content of the prototype.
   f) Select the *Manage Products* prototype to open it.
   g) Select the *Share* button in the page overview.
   h) In the new dialog box, choose *Download Zip of Prototype* in the *DOWNLOAD* tab.
   i) Close the dialog window when the file has downloaded to your browser window.
2. Import the downloaded prototype into the *SAP Web IDE Full-Stack* edition. Use SAPBuildPrototype as the project name for the import.
   a) Start the SAP Web IDE as shown in the previous exercise.

b) Choose from the context menu *Import → File or Project* from the Workspace folder of your project overview.

c) Browse and select the file you downloaded in the previous task and insert the value **`SAP Build Prototype`** in the *Import to* field.

d) After the import finishes successfully, you will see the newly imported project in the *SAP Web IDE*. Investigate the files of the imported project.

3. Test the imported project with mock data. Create a runtime configuration with the name SAP Build prototype and use the html file with the name `testFLPServiceMockServer.html` located in the folder of your test project.

a) Select the project root folder *SAPBuildPrototype* and choose from the context menu *Run → Run Configuration*.

b) Select the *+ symbol* to add a new run-configuration in the new dialog box.

c) Choose the option to *Run as Web Application* and a new default runtime configuration is created in the context menu.

d) Change the value in the *Name* field to **`SAP Build prototype`**.

e) Choose to execute the option *testFLPServiceMockServer.html* from the list of available files in the test folder of your project.

f) Select the checkbox *Run with mock data.*

g) Choose *Save* and *Run* to start the configuration.

h) Choose the *Manage Products* tile in the *My Home* section to view the application executed with mock data.

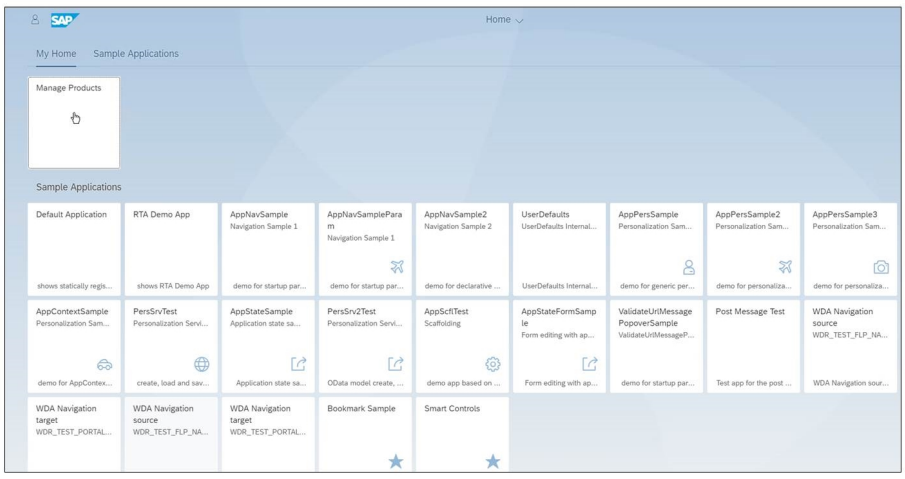


Figure 44: My Home Tile Manager

i) Explore the application. Not all aspects of the prototype are fully implemented.

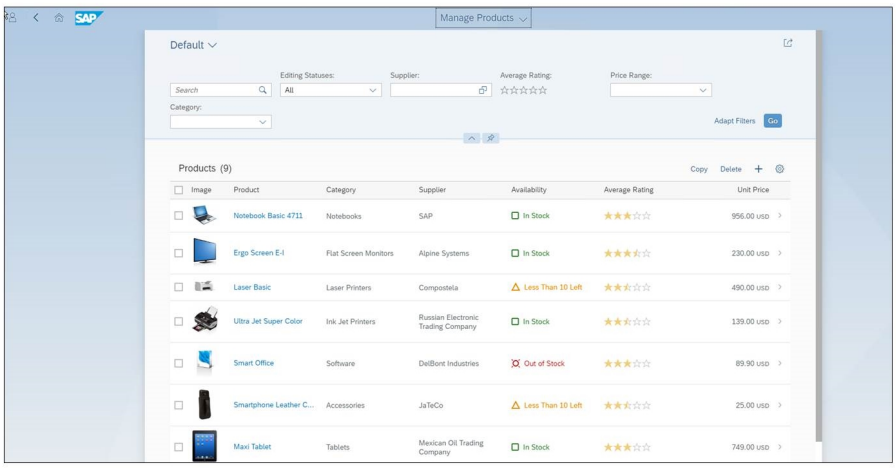


Figure 45: Executed with Mock Data

Unit 7
Exercise 7

Work with Diagrams

Note:
Please be aware that in some screenshots the code of previous implementation steps is not shown. Please do not delete any code that you have added in previous steps in the following steps.

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, step procedures and naming of fields and buttons may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

Objective:

Create a SAPUI5 project to view data with graphs and tables.

Business Scenario

Implement a simple analytical application to visualize your data with graphs and tables. Use selection criteria to choose the data to display.

Follow the guidelines to create a full-screen application with a *Main* view.

Task 1: Create a SAPUI5 project to view data with graphs and tables

1. Create a SAPUI5 project with the following settings:

Parameter	Value
Project name	diagram
Namespace	student##.sap.training
Viewtype	XML
View name	Main
SAPUI5 Version	1.60

Task 2: Implement the model

1. Create a folder with the name localService inside the webapp folder.
2. Add a new DataSource configuration by choosing the `metadata.xml` file located at `S:\Courses\UX410_20\templates\diagram\metadata.xml` as a service document and assigning a model with the name SalesModel to the newly created DataSource.

3. Enter mock data.

Task 3: Implement a line graph

1. Display data with a line graph by replacing the generated Page control in the *Main.view.xml* with a control of type *sap.f.DynamicPage*. Add the following attributes to the newly created control.

Attribute	Value
id	dynamicPageId
headerExpanded	true
toggleHeaderOnTitleClick	true

2. Add a *sap.f.DynamicPageTitle* control to the title aggregation of the newly added *DynamicPage* control.
3. Add a *sap.m.Title* control to the heading aggregation of the newly added *DynamicPageTitle* control.

Attribute	Value
text	Sales figures

Note:
Add the value as an attribute of your **i18n** property file. Use the value `pageTitle` as the key of your **i18n** file.

4. Add a *sap.suite.ui.commons.ChartContainer* to the content aggregation of the *DynamicPage* control. Use **`suite`** as a *XML-namespace* alias for the sap.suite.ui.commons-SAPUI5-namespace. Add the table attributes to the *ChartContainer-content* control.

Field	Value
id	idChartContainer
autoAdjustHeight	true
contentChange	onContentChange
showFullScreen	true
showLegend	true
showPersonalization	false
title	Sales figures

Note:
Add the title of the *ChartContainerContent* as an **i18n** property and use the id **`diagramTitle`**.

5. Add a *sap.suite.ui.commons.ChartContainerContent* control to the content aggregation of the *ChartContainer* control. Use the attributes listed in the table. Add the content aggregation to the newly added *sap.suite.ui.commons.ChartContainerContent* control.

Attribute	Value
icon	sap-icon://line-chart
title	Line Chart

Note:
Assign the title **`ChartContainerContent`** to the i18n property file and use the id `lineChartTitle`.

6. Add a *sap.ui.viz.Popover* control to the content aggregation of the *ChartContainerContent* control with the id `idLineChartPopover`.

7. Add a *sap.ui.viz.VizFrame* control after the newly added *Popover* control. Use the attributes in the following table.

Attribute	Value
id	sap-icon://line-chart
height	9.1rem
width	100%
uiConfig	{applicationSet:'fiori'}

8. Implement the `onInit` function in the controller of *Main* view. Declare a member variable with the name `sCurrentVizFrame` and assign the value **`idLineChartVizFrame`** as a string to the variable. Declare a member variable **`sCurrentSelectedDimension`** and assign the value **`0`** as type string.

9. Add a reference of `sap.viz.ui5.controls.common.feeds.FeedItem` class to the controller implementation.

10. Implement a function called `_createFeedMap`.

```
_createFeedMap: function () {
    this.feedMap = {};
    this.feedMap.salesAmount = new FeedItem({
        "uid": "valueAxis",
        "type": "Measure",
        "values": ["SalesAmount"]
    });

    this.feedMap.products = new FeedItem({
        "uid": "categoryAxis",
        "type": "Dimension",
        "values": ["Products"]
    });

    this.feedMap.subregion = new FeedItem({
        "uid": "categoryAxis",
        "type": "Dimension",
        "values": ["Sub_Region_Name"]
    });

    this.feedMap.productsSubregion = new FeedItem({
        "uid": "categoryAxis",
        "type": "Dimension",
        "values": ["Products", "Sub_Region_Name"]
    });

}
```

Figure 61: Add Reference

11. Add a reference to the `sap.viz.ui5.data.MeasureDefinition` and to the class `sap.viz.ui5.data.DimensionDefinition` to your controller implementation and implement the below function `_createDataSetMap` in your *Main* controller.

12. Invoke two new functions `_createDataSetMap` and `_createFeedMap` from inside the `onInit` function of your controller.

Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/viz/ui5/controls/common/feeds/FeedItem",
    "sap/viz/ui5/data/DimensionDefinition",
    "sap/viz/ui5/data/MeasureDefinition"
], function (Controller,FeedItem,DimensionDefinition,MeasureDefinition) {
    "use strict";

    return Controller.extend("student00.sap.training.diagram.controller.Main", {
        onInit: function () {
            this.sCurrentVizFrame  = "idLineChartVizFrame";
            this.sCurrentSelectedDimension = "0";
            this._createFeedMap();
            this._createDataSetMap();
        },
        _createFeedMap: function () {},

        _createDataSetMap: function () {}
    });
});
```

Figure 64: Invoke Functions

13. Implement a function `_createDataSet` inside the `Main.controller.js` file. Declare a variable *oDataSet* and assign an object of type `sap.viz.ui5.data.FlattenedDataset` . Bind the Dataset to the `EntitySet SalesFigures` of your SalesModel. Invoke the newly created function in the `onInit` function of your controller. Assign the newly created dataset to a member variable with the name *dataSet*.

14. Create a function `_handleSelection` to handle the assignment of dimension and measure to the current selected diagram. The function should take a parameter with the name `selectedItem`. The implementation of the function should do the following: store

the reference of the currently visible *VizFrame* control using the stored value from variable *sCurrentVizFrame* and store the reference to the control as a local variable.

15. Create a function with name `_createLineDiagram`.

16. Test the application once the line graph is implemented. Execute using the *MockServer* option.

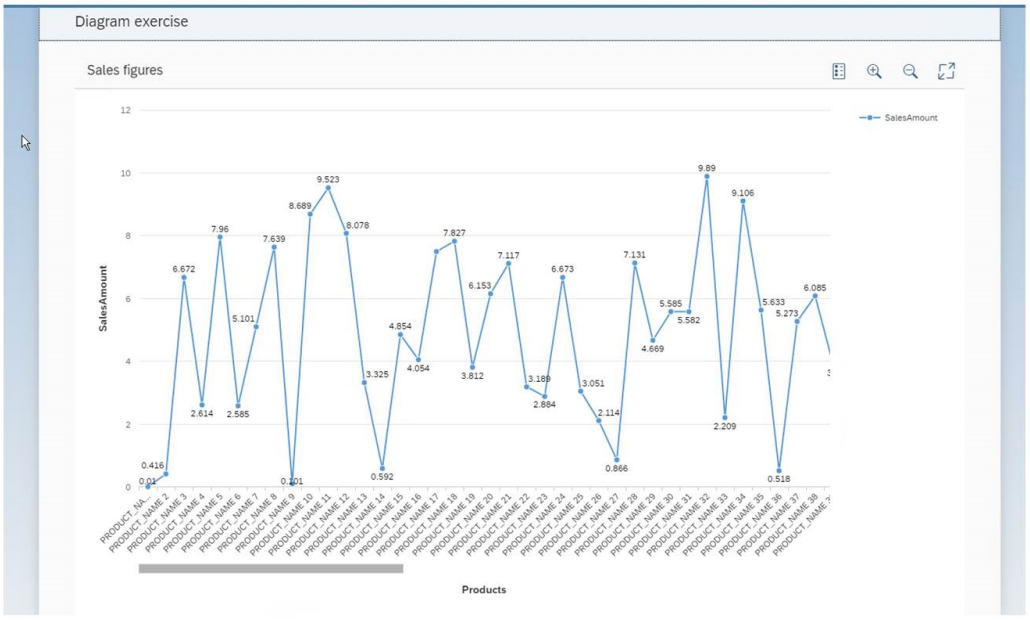


Figure 71: Implementation with Mock Data

Task 4: Implement a bar chart

1. Implement a bar chart. The surrounding `ChartContainerContent` control requires an attribute with the title Column Chart. Use file `i18n`.

Attribute	Value
id	idBarChartVizFrame
height	9.1rem
width	100%
uiConfig	{applicationSet:'fiori'}

2. Implement the function `_createColumnChart`.

3. Develop the event handler `contentChange` to switch between diagram views with the function `onContentChange`.

Task 5: Implement a table view

1. Create a `ChartContainerContent` with the *table-chart* icon and Sales figures as table title using the i18n file.

2. Create a function `_createTable`. in the controller of the *Main* view with dependencies.

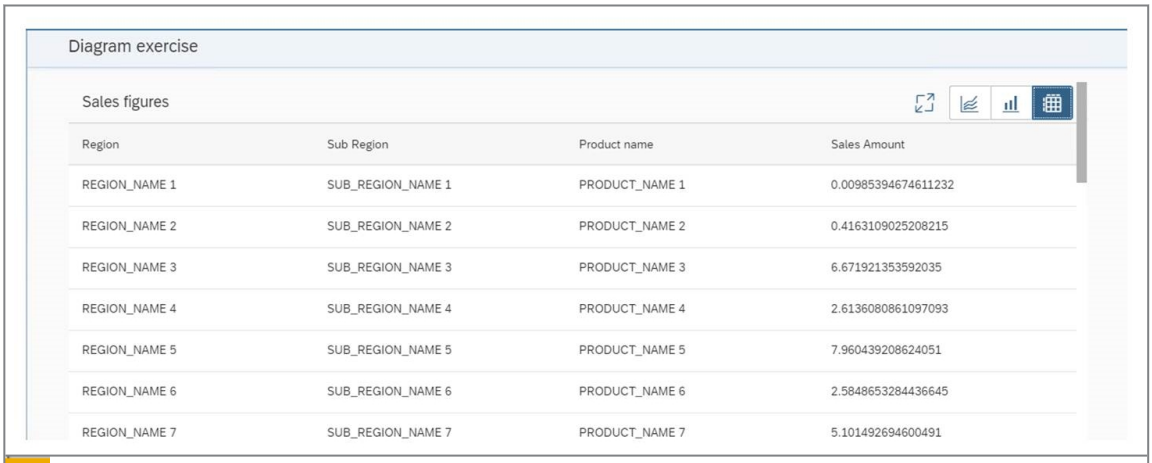


Figure 81: Test Application

Task 6: Implement a dimension selection

1. Implement a dimension selection. Use a control of type sap.m.Select with the following attribute and values:

Parameter	Value
id	idDimSelector
visible	true
change	onChange

2. Create a function `_createSelector`. Use the values in the table below.

   The function fills the previously created selection list with values. Extend the implementation of the function handleSelection to handle the different visualization facettes.

Item-Key	Item-Text
0	per Product
1	per Region
2	per Product and Region

3. To display more meaningful data in the charts rename the file `SalesFigures.json` located in the localService folder to `SalesFigures_Gen.json`. Import the file `SalesFigures.json` located at `s:\courses\UX410_20\templates\diagram\` into the localService folder. Change the Mock Data configuration of your project so that the data from the`SalesFigures.json` file is used.

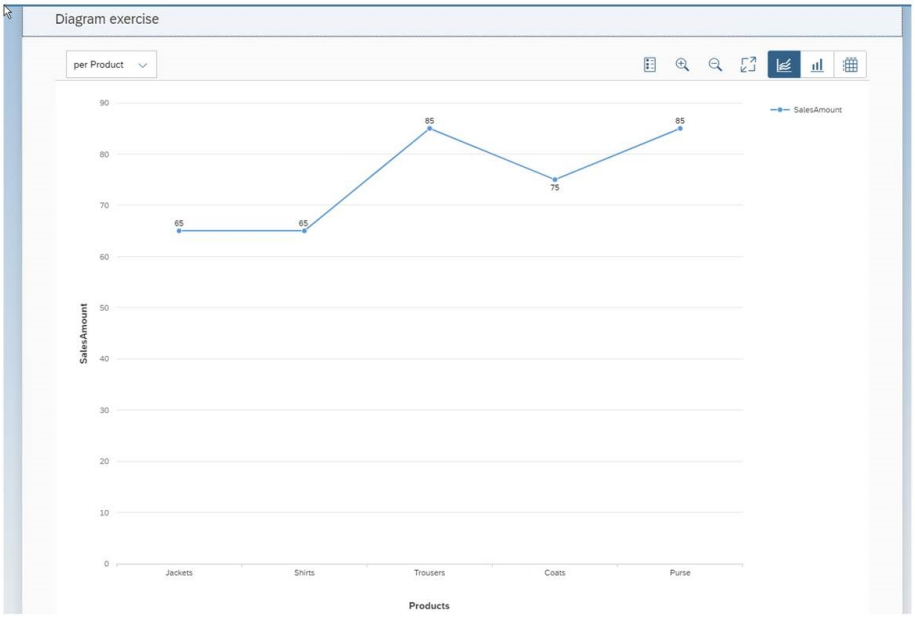


Figure 94: Dimension Selection Implementation

Unit 7
Solution 7

Work with Diagrams

Note:
Please be aware that in some screenshots the code of previous implementation steps is not shown. Please do not delete any code that you have added in previous steps in the following steps.

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, step procedures and naming of fields and buttons may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

Objective:

Create a SAPUI5 project to view data with graphs and tables.

Business Scenario

Implement a simple analytical application to visualize your data with graphs and tables. Use selection criteria to choose the data to display.

Follow the guidelines to create a full-screen application with a *Main* view.

Task 1: Create a SAPUI5 project to view data with graphs and tables

1. Create a SAPUI5 project with the following settings:

Parameter	Value
Project name	diagram
Namespace	student##.sap.training
Viewtype	XML
View name	Main
SAPUI5 Version	1.60

   a) Log on to the *SAP Web IDE Full-Stack.*

   b) Choose *File* → *New* → *Project from Template* → *SAPUI5 Application Project*, select the *SAPUI5-Version 1.60* and choose *Next.*

   c) Name your project diagram and in the *Namespace* field enter **`student##.sap.training`** and select *Next.*

Note:
Use your assigned student and group number.

d) In the *View Name* field enter **`Main`**.

e) Choose *Finish* and the newly created project diagram appears in your workspace.

Task 2: Implement the model

1. Create a folder with the name localService inside the webapp folder.
   a) Select the webapp folder of your project and choose *New* → *Folder* from the context menu.
   b) Name the folder localService in the dialog box and choose *OK*.
2. Add a new DataSource configuration by choosing the `metadata.xml` file located at `S:\Courses\UX410_20\templates\diagram\metadata.xml` as a service document and assigning a model with the name SalesModel to the newly created DataSource.
   a) Open the *manifest.json* file in the *Descriptor Editor* view.
   b) Choose the *Data Sources* tab and with the *plus symbol* select *File System* as a Source.
   c) Browse and navigate to the `metadata.xml` file located at `S:\Courses\UX410_20\templates\diagram.`
   d) Select the `metadata.xml` file with a double click and choose *Next* to confirm the data connection step in the data source dialog.
   e) Choose the option *Create new model* in the model selection and insert **`SalesModel`** in the *Model Name* field and choose *Next*.
   f) Choose *Finish* to create the new data source and model.
   g) Save the changes.
3. Enter mock data.
   a) Select your project and choose *Project* → *Project Settings* from the context menu.
   b) Choose *Mock Data* and *Generated data* and save the selection.
   c) Select the *metadata.xml* file, and from the context menu choose the option *Edit Mock Data*.
   d) Select the *Use the data above as my mock data source* checkbox.
   e) Select the *Generate Random Data* button.
   f) Confirm the selection.
   The *SAP Web IDE* generates a file with the name **`SalesFigures.json`** and adds the newly created file to a folder named mockdata.
   g) Open the file and analyse the content.

Task 3: Implement a line graph

1. Display data with a line graph by replacing the generated Page control in the *Main.view.xml* with a control of type *sap.f.DynamicPage*. Add the following attributes to the newly created control.

Attribute	Value
id	dynamicPageId
headerExpanded	true
toggleHeaderOnTitleClick	true

   a) Open the file *Main.view.xml*.

   b) Add the **`namespace`** alias to the view control.

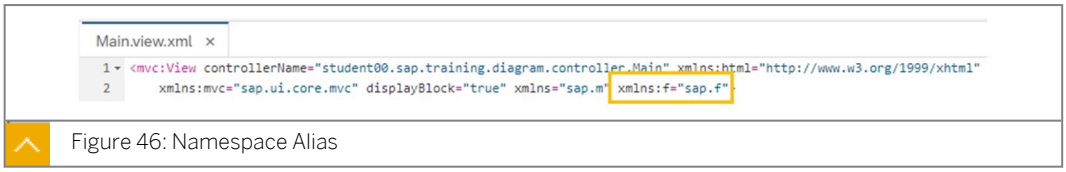


Figure 46: Namespace Alias

   c) Remove the generated Page control from the App control.

   d) Add the *DynamicPage* control with the attributes **`XML-Namespace alias f`** for *sap.f-UI5-namespace* to the app control.

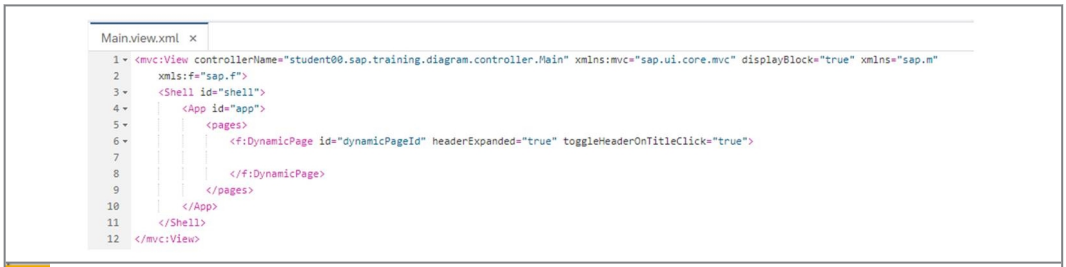


Figure 47: Add Control and Attributes

2. Add a *sap.f.DynamicPageTitle* control to the title aggregation of the newly added *DynamicPage* control.

   a) Add the title Aggregation to the *DynamicPage* control.

   b) Add the UI control *sap.f.DynamicPageTitle* to the title aggregation.

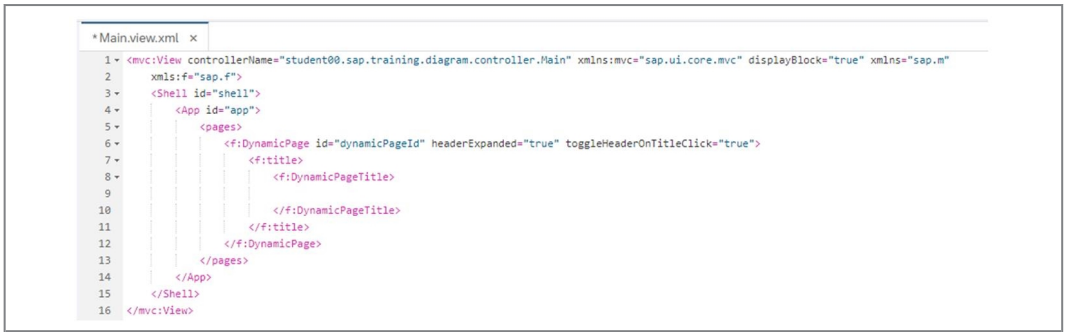


Figure 48: Add UI Control

3. Add a *sap.m.Title* control to the heading aggregation of the newly added *DynamicPageTitle* control.

Attribute	Value
text	Sales figures

Note:
Add the value as an attribute of your **i18n** property file. Use the value `pageTitle` as the key of your **i18n** file.

a) Add the heading aggregation of the *DynamicPageTitle* control and add a *sap.m.Title* control with the attribute *text* to the newly created *heading* aggregation.

Figure 49: Page Title

b) Use the following attribute values for the *sap.m.Title* control, type **`text`** by putting the cursor between the "" of the text attribute and choose *Create i18n String from the context menu*.

c) Add **`pageTitle`** as the *String key* field and **`Diagram Exercise`** as the value.

d) Choose *XHED* as type from the selection box *Type*.

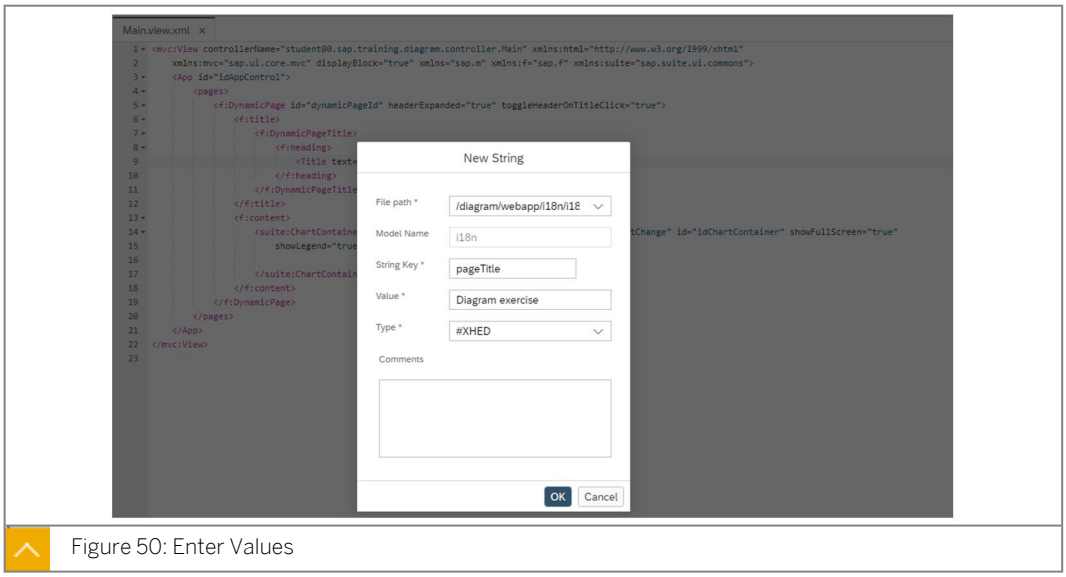


Figure 50: Enter Values

e) Press *OK* to display your implementation.

Figure 51: Implementation

4. Add a *sap.suite.ui.commons.ChartContainer* to the content aggregation of the *DynamicPage* control. Use **`suite`** as a *XML-namespace* alias for the sap.suite.ui.commons-SAPUI5-namespace. Add the table attributes to the *ChartContainer-content* control.

Field	Value
id	idChartContainer
autoAdjustHeight	true
contentChange	onContentChange
showFullScreen	true
showLegend	true
showPersonalization	false
title	Sales figures

Note:
Add the title of the *ChartContainerContent* as an i18n property and use the id **`diagramTitle`**.

a) Add the content aggregation to the *DynamicPage* control.

b) Add the XML-namespace alias suite to view control and assign the UI5-namespace *sap.suite.ui.commons* to the alias suite.

c) Create a new UI element of the type *ChartContainer* with the table attributes.

Figure 52: Add UI Element

d) Add the title of the *ChartContainer* control to the i18n properties file of your project by placing the cursor on the tile *property* and choose *Create i18n String* from the context menu,

e) Insert the table data and choose *OK* to display your implementation.

Figure 53: Implementation

5. Add a *sap.suite.ui.commons.ChartContainerContent* control to the content aggregation of the *ChartContainer* control. Use the attributes listed in the table. Add the content aggregation to the newly added *sap.suite.ui.commons.ChartContainerContent* control.

Attribute	Value
icon	sap-icon://line-chart
title	Line Chart

Note:
Assign the title **`ChartContainerContent`** to the i18n property file and use the id `lineChartTitle`.

a) Add the content aggregation to the *ChartContainer* control.

b) Add a *ChartContainerContent* control using the table attributes.

Figure 54: Use the Table Attributes

c) Add the content aggregation to the newly added *ChartContainerContent* control.

6. Add a *sap.ui.viz.Popover* control to the content aggregation of the *ChartContainerContent* control with the id `idLineChartPopover`.

a) Add a XML-namespace viz pointing to sap.ui.viz to your view implementation.

b) Add the *sap.ui.viz.Popover* control to the content aggregation of the *ChartContainerContent* control.

Figure 55: Add Control

7. Add a *sap.ui.viz.VizFrame* control after the newly added *Popover* control. Use the attributes in the following table.

Attribute	Value
id	sap-icon://line-chart
height	9.1rem
width	100%
uiConfig	{applicationSet:'fiori'}

a) Add a UI control of type *sap.ui.viz.VizFrame* to the content aggregation of the *ChartContainerContent* control.

Figure 56: Add UI Control

b) Save your changes.

8. Implement the `onInit` function in the controller of *Main* view. Declare a member variable with the name `sCurrentVizFrame` and assign the value **`idLineChartVizFrame`** as a string to the variable. Declare a member variable **`sCurrentSelectedDimension`** and assign the value **`0`** as type string.

a) Open the file Main.controller located in the controller folder of your project.

b) Add an implementation of the function `onInit` to the `Main.controller.js` implementation.

Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller"
], function (Controller) {
    "use strict";

    return Controller.extend("student00.sap.training.diagram.controller.Main", {
        onInit: function () {

        }
    });
});
```

Figure 57: Add the Function

c) Declare a member variable *sCurrentVizFrame* and assign the value **`idLineChartVizFrame`** as a string.

Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller"
], function (Controller) {
    "use strict";

    return Controller.extend("student00.sap.training.diagram.controller.Main", {
        onInit: function () {
            this.sCurrentVizFrame  = "idLineChartVizFrame";
        }
    });
});
```

Figure 58: Add String

d) Declare a member variable *sCurrentSelectedDimension* and assign the value **0** as a string.

Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller"
], function (Controller) {
    "use strict";

    return Controller.extend("student00.sap.training.diagram.controller.Main", {
        onInit: function () {
            this.sCurrentVizFrame  = "idLineChartVizFrame";
            this.sCurrentSelectedDimension = "0";
        }
    });
});
```

Figure 59: Declare Variable

e) Save your changes.

9. Add a reference of `sap.viz.ui5.controls.common.feeds.FeedItem` class to the controller implementation.

a) Add a reference to the `sap.viz.ui5.controls.common.feeds.FeedItem class.`

Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/viz/ui5/controls/common/feeds/FeedItem"
], function (Controller,FeedItem) {
    "use strict";
```

Figure 60: Add Reference

10. Implement a function called `_createFeedMap`.

```
_createFeedMap: function () {
    this.feedMap = {};
    this.feedMap.salesAmount = new FeedItem({
        "uid": "valueAxis",
        "type": "Measure",
        "values": ["SalesAmount"]
    });

    this.feedMap.products = new FeedItem({
        "uid": "categoryAxis",
        "type": "Dimension",
        "values": ["Products"]
    });

    this.feedMap.subregion = new FeedItem({
        "uid": "categoryAxis",
        "type": "Dimension",
        "values": ["Sub_Region_Name"]
    });

    this.feedMap.productsSubregion = new FeedItem({
        "uid": "categoryAxis",
        "type": "Dimension",
        "values": ["Products", "Sub_Region_Name"]
    });

}
```

Figure 61: Add Reference

11. Add a reference to the `sap.viz.ui5.data.MeasureDefinition` and to the class `sap.viz.ui5.data.DimensionDefinition` to your controller implementation and implement the below function `_createDataSetMap` in your *Main* controller.

a) Add the reference to `sap.viz.ui5.data.DimensionDefinition` and to `sap.viz.ui5.data.MeasureDefinition` of your controller implementation.

Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/viz/ui5/controls/common/feeds/FeedItem",
    "sap/viz/ui5/data/DimensionDefinition",
    "sap/viz/ui5/data/MeasureDefinition"
], function (Controller,FeedItem,DimensionDefinition,MeasureDefinition) {
    "use strict";
```

Figure 62: Add References

b) Add the implementation to your *Main* controller implementation.

Figure 63: Implementation of _createDataSetMap

12. Invoke two new functions `_createDataSetMap` and `_createFeedMap` from inside the `onInit` function of your controller.

Figure 64: Invoke Functions

13. Implement a function `_createDataSet` inside the `Main.controller.js` file. Declare a variable *oDataSet* and assign an object of type `sap.viz.ui5.data.FlattenedDataset` . Bind the Dataset to the `EntitySet SalesFigures` of your SalesModel. Invoke the newly created function in the `onInit` function of your controller. Assign the newly created dataset to a member variable with the name *dataSet*.

a) Open the file `Main.controller.js` and add a reference to the `sap.viz.ui5.data.FlattenedDataset`.

Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/viz/ui5/controls/common/feeds/FeedItem",
    "sap/viz/ui5/data/DimensionDefinition",
    "sap/viz/ui5/data/MeasureDefinition",
    "sap/viz/ui5/data/FlattenedDataset"
], function (Controller, FeedItem, DimensionDefinition, MeasureDefinition, FlattenedDataset) {
    "use strict";
```

Figure 65: Add a Reference

b) Implement the `_createDataSet` function.

Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/viz/ui5/controls/common/feeds/FeedItem",
    "sap/viz/ui5/data/DimensionDefinition",
    "sap/viz/ui5/data/MeasureDefinition",
    "sap/viz/ui5/data/FlattenedDataset"
], function (Controller, FeedItem, DimensionDefinition, MeasureDefinition, FlattenedDataset) {
    "use strict";

    return Controller.extend("student00.sap.training.diagram.controller.Main", {
        onInit: function () {},
        _createFeedMap: function () {},

        _createDataSetMap: function () {},

        _createDataSet: function () {
            var oDataset = new FlattenedDataset({
                data: {
                    path: "SalesModel>/SalesFigures"
                }
            });
            return oDataset;
        }
    });
});
```

Figure 66: Implement Function

c) Invoke the function in the `onInit` function of your controller. Assign the return value to a member variable *_dataSet*.

Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/viz/ui5/controls/common/feeds/FeedItem",
    "sap/viz/ui5/data/DimensionDefinition",
    "sap/viz/ui5/data/MeasureDefinition",
    "sap/viz/ui5/data/FlattenedDataset"
], function (Controller, FeedItem, DimensionDefinition, MeasureDefinition, FlattenedDataset) {
    "use strict";

    return Controller.extend("student00.sap.training.diagram.controller.Main", {
        onInit: function () {
            this.sCurrentVizFrame = "idLineChartVizFrame";
            this.sCurrentSelectedDimension = "0";
            this._createFeedMap();
            this._createDataSetMap();
            this._dataSet = this._createDataSet();
        },
```

Figure 67: Assign Return Value

14. Create a function `_handleSelection` to handle the assignment of dimension and measure to the current selected diagram. The function should take a parameter with the name `selectedItem`. The implementation of the function should do the following: store the reference of the currently visible *VizFrame* control using the stored value from variable *sCurrentVizFrame* and store the reference to the control as a local variable.

a) Remove all the dimensions, measures, and feeds from the dataset object of the currently visible `VizFrame` object.

b) Implement a switch-statement and check if the value of *selectedItem* is **0** as a string. When *selectedItem* is equal to "0" add the *productDim* and the *salesAmoutMeasure* of the *dataSetMap* to the dataset of the current *VizFrame*.

c) Add the corresponding feeds to the *VizFrame*.

d) Implement the function `_handleSelection`.

*Main.controller.js

```
    "sap/viz/ui5/data/FlattenedDataset"
], function (Controller, FeedItem, DimensionDefinition, MeasureDefinition, FlattenedDataset) {
    "use strict";

    return Controller.extend("student00.sap.training.diagram.controller.Main", {
        onInit: function () {},
        _createFeedMap: function () {},
        _createDataSetMap: function () {},
        _createDataSet: function () {},
        _handleSelection: function (selectedItem) {
            var oVizFrame = this.getView().byId(
                this.sCurrentVizFrame);
            var oDataSet = oVizFrame.getDataset();
            oDataSet.removeAllDimensions();
            oDataSet.removeAllMeasures();
            oVizFrame.removeAllFeeds();
            switch (selectedItem) {
            case "0":
                {
                    oDataSet.addDimension(this.dataSetMap.productDim);
                    oDataSet.addMeasure(this.dataSetMap.salesAmountMeasure);
                    oVizFrame.addFeed(this.feedMap.products);
                    oVizFrame.addFeed(this.feedMap.salesAmount);

                    break;
                }
            }
        }
    });
});
```

Figure 68: Implement Function

15. Create a function with name `_createLineDiagram`.

a) The function handles the creation of the line diagram. Read the reference of the *VizFrame* element with the ID `idLineChartVizFrame` and assign it to a variable. Provide a reference on the popover element with the ID `idLineChartPopover`.

b) Call the function `_createDataSet` and assign the result to the reference of the *VizFrame* object which was read in step a.

c) Call the function `_handleSelection` and pass the value of the member variable *sCurrentSelectedDimension* created in the `onInit` function.

d) Call the function `setVizProperties`. Pass an object to define that data shown in the diagram should be labelled. Define the title not to display in the diagram.

e) Pass a string literal with the value **`line`** to define the use of a line diagram by means of the function `setVizType` of the *VizFrameobject*.

f) Use the function `getVizUid` to call the function connect on the popover object of step a and pass the unique ID of the *VizFrame* object.

g) Check that the implementation is as expected.

Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/viz/ui5/controls/common/feeds/FeedItem",
    "sap/viz/ui5/data/DimensionDefinition",
    "sap/viz/ui5/data/MeasureDefinition",
    "sap/viz/ui5/data/FlattenedDataset"
], function (Controller, FeedItem, DimensionDefinition, MeasureDefinition, FlattenedDataset) {
    "use strict";

    return Controller.extend("student00.sap.training.diagram.controller.Main", {
        onInit: function () {},
        _createFeedMap: function () {},
        _createDataSetMap: function () {},
        _createDataSet: function () {},
        _handleSelection: function (selectedItem) {},
        _createLineDiagram: function () {
            var oVizFrame = this.getView().byId("idLineChartVizFrame");
            var oPop = this.getView().byId("idLineChartPopover");
            oVizFrame.setDataset(this._createDataSet());
            this._handleSelection(this.sCurrentSelectedDimension);
            oVizFrame.setVizProperties();
            oVizFrame.setVizType('line');
            oPop.connect(oVizFrame.getVizUid());
        },
    });
});
```

Figure 69: Implementation

h) Insert the `_createLineDiagram` function invocation at the end of your `onInit` function. Your `onInit` function should have the implementation required.

*Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/viz/ui5/controls/common/feeds/FeedItem",
    "sap/viz/ui5/data/DimensionDefinition",
    "sap/viz/ui5/data/MeasureDefinition",
    "sap/viz/ui5/data/FlattenedDataset"
], function (Controller, FeedItem, DimensionDefinition, MeasureDefinition, FlattenedDataset) {
    "use strict";

    return Controller.extend("student00.sap.training.diagram.controller.Main", {
        onInit: function () {
            this.sCurrentVizFrame = "idLineChartVizFrame";
            this.sCurrentSelectedDimension = "0";
            this._createFeedMap();
            this._createDataSetMap();
            this._dataSet = this._createDataSet();
            this._createLineDiagram();
        },
        _createFeedMap: function () {},
        _createDataSetMap: function () {},
        _createDataSet: function () {},
        _handleSelection: function (selectedItem) {},
        _createLineDiagram: function () {},
    });
});
```

Figure 70: Implementation

16. Test the application once the line graph is implemented. Execute using the *MockServer* option.

a) Select `index.html` file and choose from the context menu *Run → Run Configurations*.

b) Press the + Button to add a new run configuration.

c) Choose *Run as Web Application*.

d) Check *Run with mock data* and select *Save and Run.*

e) Select the `index.html` file and from the context menu choose *Run → Run with Mock Data*.

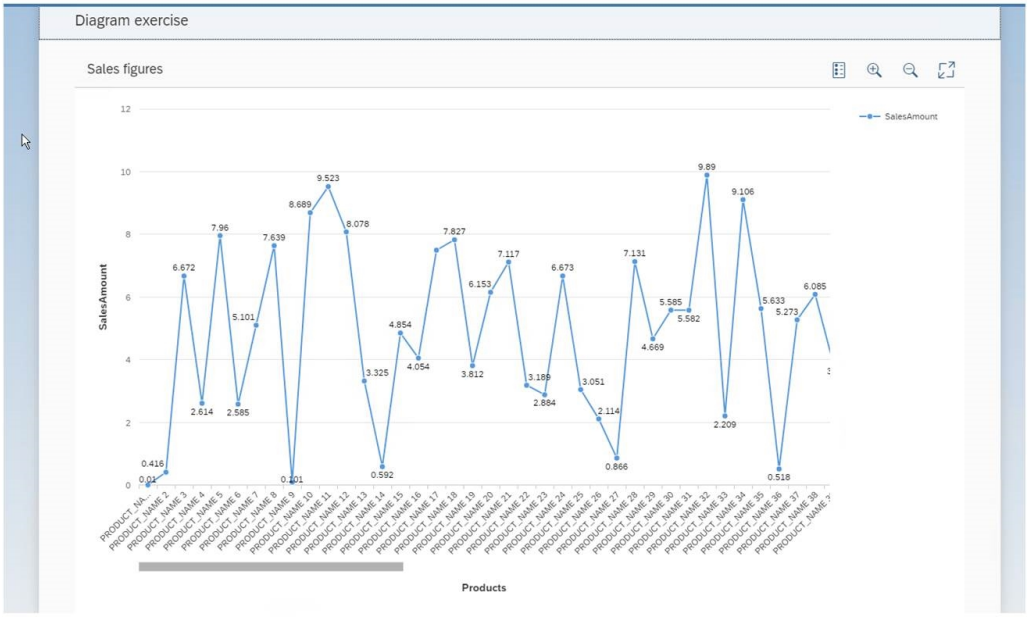


Figure 71: Implementation with Mock Data

Task 4: Implement a bar chart

1. Implement a bar chart. The surrounding `ChartContainerContent` control requires an attribute with the title Column Chart. Use file `i18n`.

Attribute	Value
id	idBarChartVizFrame
height	9.1rem
width	100%
uiConfig	{applicationSet:'fiori'}

   a) Create a new `ChartContainerContent` in the `Main.view.xml` file.

   b) The attribute icon of the new element, refers to the icon with the identifier vertical-bar-chart.

   c) Create an element with the type `VizFrame` in the content aggregation of the `ChartContainerContent` element.

   d) Assign the ID **`idBarChartVizFrame`** to the element, and assign an object with the variable **`applicationSet`** to the attribute `uiConfig`. The variable gets the value **`fiori.`**

   e) Review your implementation.

Main.view.xml

```
<mvc:View controllerName="student00.sap.training.diagram.controller.Main" xmlns:html="http://www.w3.org/1999/xhtml"
    xmlns:mvc="sap.ui.core.mvc" displayBlock="true" xmlns="sap.m" xmlns:f="sap.f" xmlns:suite="sap.suite.ui.commons"
    xmlns:viz="sap.viz.ui5.controls">
    <App id="idAppControl">
        <pages>
            <f:DynamicPage id="dynamicPageId" headerExpanded="true" toggleHeaderOnTitleClick="true">
                <f:title>
                    <f:DynamicPageTitle>
                        <f:heading>
                            <Title text="{i18n>pageTitle}"/>
                        </f:heading>
                    </f:DynamicPageTitle>
                </f:title>
                <f:content>
                    <suite:ChartContainer autoAdjustHeight="true" contentChange="onContentChange" id="idChartContainer"
                        showFullScreen="true" showLegend="true" showPersonalization="false" title="{i18n>diagramTitle}">
                        <suite:content>
                            <suite:ChartContainerContent icon="sap-icon://line-chart" title="{i18n>lineChartTitle}">
                            <suite:ChartContainerContent icon="sap-icon://vertical-bar-chart" title="{i18n>barChartTitle}">
                                <suite:content>
                                    <viz:VizFrame id="idBarChartVizFrame" height="9.1rem" width="100%" uiConfig="{applicationSet:'fiori'}"></viz:VizFrame>
                                </suite:content>
                            </suite:ChartContainerContent>
                        </suite:content>
                    </suite:ChartContainer>
                </f:content>
            </f:DynamicPage>
        </pages>
    </App>
</mvc:View>
```

Figure 72: Review Implementation

2. Implement the function `_createColumnChart`.
   a) Open the view controller `Main.controller.js`.
   b) Implement the function `_createColumnChart`.
   c) Create a reference to the VizFrame element already created.
   d) Assign a configuration for `VizProperties` to the VizFrame object. To do this specify that the chart title should not be displayed, and define the beams to display with attribute data labels. Use the `setVizType` function to specify a bar chart. You can use a string literal with value **`column`**.

*Main.controller.js

```
    "sap/viz/ui5/data/DimensionDefinition",
    "sap/viz/ui5/data/MeasureDefinition",
    "sap/viz/ui5/data/FlattenedDataset"
], function (Controller, FeedItem, DimensionDefinition, MeasureDefinition, FlattenedDataset) {
    "use strict";

    return Controller.extend("student00.sap.training.diagram.controller.Main", {
        onInit: function () {},
        _createFeedMap: function () {},
        _createDataSetMap: function () {},
        _createDataSet: function () {},
        _handleSelection: function (selectedItem) {},
        _createLineDiagram: function () {},
        _createColumnChart: function () {
            var oVizFrame = this.getView().byId("idBarChartVizFrame");
            oVizFrame.setDataset(this._createDataSet());
            oVizFrame.setVizProperties({
                plotArea: {
                    dataLabel: {
                        visible: true
                    }
                },
                title: {
                    visible: false
                }
            });
            oVizFrame.setVizType('column');
        }
    });
});
```

Figure 73: Assign a Configuration

3. Develop the event handler `contentChange` to switch between diagram views with the function `onContentChange`.
   a) Create an event handler function with the name `onContentChange`.
   b) Read out the parameter *selectedItemId*.

c) Check that the parameter *selectedItemId* contains the string `Table`. If not, split the parameter and assign it to the second value of the resulting array of member variables `sCurrentVizFrame`.

d) Call the function `_handleSelection` and hand it over to the *sCurrentSelectedDimension* variable.

```
onContentChange: function (oEvent) {
    var sSelectedVizFrame = oEvent.getParameter("selectedItemId");
    if (sSelectedVizFrame.indexOf("Table") === -1) {
        this.sCurrentVizFrame = sSelectedVizFrame.split("--")[2];
        this._handleSelection(this.sCurrentSelectedDimension);
    }
},
```

Figure 74: Specify Bar Chart

e) Add the call up `_createColumnChart` function at the end of your `onInit` implementation.

Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/viz/ui5/controls/common/feeds/FeedItem",
    "sap/viz/ui5/data/DimensionDefinition",
    "sap/viz/ui5/data/MeasureDefinition",
    "sap/viz/ui5/data/FlattenedDataset"
], function (Controller, FeedItem, DimensionDefinition, MeasureDefinition, FlattenedDataset) {
    "use strict";

    return Controller.extend("student00.sap.training.diagram.controller.Main", {
        onInit: function () {
            this.sCurrentVizFrame = "idLineChartVizFrame";
            this.sCurrentSelectedDimension = "0";
            this._createFeedMap();
            this._createDataSetMap();
            this._dataSet = this._createDataSet();
            this._createLineDiagram();
            this._createColumnChart();
        },
        _createFeedMap: function () {},
        _createDataSetMap: function () {},
        _createDataSet: function () {},
        _handleSelection: function (selectedItem) {},
        _createLineDiagram: function () {},
        _createColumnChart: function () {},
        onContentChange: function (oEvent) {}
    });
});
```

Figure 75: Call Function

f) Save your changes.

g) Test the applications. Switch between the line graph and bar chart.

Task 5: Implement a table view

1. Create a `ChartContainerContent` with the *table-chart* icon and Sales figures as table title using the i18n file.

   a) Add an `sap.m.Table` control to the content aggregation and add an id attribute to the *Table* control. Assign the value **`idTable`**.

   b) Create another `ChartContainerContent` element, assign the value for the table-chart icon to the icon attribute, and add a title attribute.

   c) Create a table of type `sap.m.Table` in the content aggregation of the `ChartContainerContent` and assign the ID **`idTable`** to the table control.

Figure 76: Implementation Result

d) Add the headerToolbar-aggregation to the newly created table. Add a control of type OverflowToolbar and add the title control to the content-aggregation of the OverflowToolbar.

```
<suite:ChartContainerContent icon="sap-icon://table-chart" title="{i18n>tableTitle}">
    <suite:content>
        <Table id="idTable">
            <headerToolbar>
                <OverflowToolbar>
                    <content>
                        <Title text="{i18n>tableheader}" level="H2"/>
                    </content>
                </OverflowToolbar>
            </headerToolbar>
        </Table>
    </suite:content>
</suite:ChartContainerContent>
```

Figure 77: Table with headerToolbar-aggregation and OverflowToolbar control

2. Create a function `_createTable`. in the controller of the *Main* view with dependencies.

a) Add the references as shown below.

Figure 78: Create Function

b) Create a function `_createTable` in the view controller of the *Main* view.

c) Create a reference to the first table in this function.

d) Add 4 columns to the table to represent the region, area, product name, and sales figures.

e) Create a template object of the type `sap.m.ColumnListItem` to display the values from the data model for the REGION_NAME, SUB_REGION_NAME, PRODUCT_NAME and SALES_AMOUNT fields in the table.

f) Bind the items in the table to the model element named sales by handing over the `sap.m.ColumnListItem` object already created to the used function `bindItem`.

Main.controller.js

```
73      _handleSelection: function (selectedItem) {},
92      _createLineDiagram: function () {},
110     _createColumnChart: function () {},
125     onContentChange: function (oEvent) {},
132     _createTable: function () {
133         var oTable = this.getView().byId("idTable");
134         oTable.addColumn(new Column({header: new Label({text: "Region",textAlign: "Right"})}));
135         oTable.addColumn(new Column({header: new Label({text: "Sub Region",textAlign: "Left"})}));
136         oTable.addColumn(new Column({header: new Label({text: "Product name"})}));
137         oTable.addColumn(new Column({header: new Label({text: "Sales Amount"})}));
138
139         var oTableTemplate = new ColumnListItem({
140             type: sap.m.ListType.Active,
141             cells: [new Label({
142                 text: "{SalesModel>REGION_NAME}"
143             }), new Label({
144                 text: "{SalesModel>SUB_REGION_NAME}",
145                 textAlign: "Left"
146             }), new Label({
147                 text: "{SalesModel>PRODUCT_NAME}",
148                 textAlign: "Right"
149             }), new Label({
150                 text: "{SalesModel>SALES_AMOUNT}",
151                 textAlign: "Center"
152
153             })]
154         });
155         oTable.bindItems("SalesModel>/SalesFigures",oTableTemplate, null, null);
156     },
157
158 });
```

Figure 79: Bind Items

g) Call the function `_createTable` in the `onInit` function.

Main.controller.js

```
1  sap.ui.define([
2      "sap/ui/core/mvc/Controller",
3      "sap/viz/ui5/controls/common/feeds/FeedItem",
4      "sap/viz/ui5/data/DimensionDefinition",
5      "sap/viz/ui5/data/MeasureDefinition",
6      "sap/viz/ui5/data/FlattenedDataset",
7      "sap/m/Column",
8      "sap/m/Label",
9      "sap/ui/core/Item",
10     "sap/m/ColumnListItem",
11 ], function (Controller, FeedItem, DimensionDefinition, MeasureDefinition, FlattenedDataset, Column, Label, Item, ColumnListItem) {
12     "use strict";
13     return Controller.extend("student00.sap.training.diagram.controller.Main", {
14         onInit: function () {
15             this.sCurrentVizFrame = "idLineChartVizFrame";
16             this.sCurrentSelectedDimension = "0";
17             this._createFeedMap();
18             this._createDataSetMap();
19             this._dataSet = this._createDataSet();
20             this._createLineDiagram();
21             this._createColumnChart();
22             this._createTable();
23         },
```

Figure 80: Call Function to Create Table

h) Test your application by switching between the line and bar chart and find another button in the toolbar table to show the newly implemented table.

Diagram exercise

Sales figures

Region	Sub Region	Product name	Sales Amount
REGION_NAME 1	SUB_REGION_NAME 1	PRODUCT_NAME 1	0.00985394674611232
REGION_NAME 2	SUB_REGION_NAME 2	PRODUCT_NAME 2	0.4163109025208215
REGION_NAME 3	SUB_REGION_NAME 3	PRODUCT_NAME 3	6.671921353592035
REGION_NAME 4	SUB_REGION_NAME 4	PRODUCT_NAME 4	2.6136080861097093
REGION_NAME 5	SUB_REGION_NAME 5	PRODUCT_NAME 5	7.960439208624051
REGION_NAME 6	SUB_REGION_NAME 6	PRODUCT_NAME 6	2.5848653284436645
REGION_NAME 7	SUB_REGION_NAME 7	PRODUCT_NAME 7	5.101492694600491

Figure 81: Test Application

Task 6: Implement a dimension selection

1. Implement a dimension selection. Use a control of type sap.m.Select with the following attribute and values:

Parameter	Value
id	idDimSelector
visible	true
change	onChange

   a) Open the `Main.view.xml` file and insert an element of type `dimensionSelector` into `ChartContainer`.

   b) Create an element of type `sap.m.Select`.

   c) Assign the ID `idDimSelector` and to the event handler `onChange` assign the change event.

*Main.view.xml

```
<mvc:View controllerName="student00.sap.training.diagram.controller.Main" xmlns:html="http://www.w3.org/1999/xhtml"
    xmlns:mvc="sap.ui.core.mvc" displayBlock="true" xmlns="sap.m" xmlns:f="sap.f" xmlns:suite="sap.suite.ui.commons"
    xmlns:viz="sap.viz.ui5.controls">
    <App id="idAppControl">
        <pages>
            <f:DynamicPage id="dynamicPageId" headerExpanded="true" toggleHeaderOnTitleClick="true">
                <f:title></f:title>
                <f:content>
                    <suite:ChartContainer autoAdjustHeight="true" contentChange="onContentChange" id="idChartContainer"
                        showFullScreen="true" showLegend="true" showPersonalization="false" title="{i18n>diagramTitle}">
                        <suite:dimensionSelectors>
                            <Select id="idDimSelector" visible="true" change="onChange"></Select>
                        </suite:dimensionSelectors>
                        <suite:content></suite:content>
                    </suite:ChartContainer>
                </f:content>
            </f:DynamicPage>
        </pages>
    </App>
</mvc:View>
```

Figure 82: Assign ID

2. Create a function `_createSelector`. Use the values in the table below.

   The function fills the previously created selection list with values. Extend the implementation of the function handleSelection to handle the different visualization facettes.

Item-Key	Item-Text
0	per Product
1	per Region
2	per Product and Region

   a) In the view controller create a function `_createSelector`. Use the `byId` function to read the selection list, and assign three objects of type `sap.ui.core.Item`. Use the i18N capabilities of SAPUI5.

   b) These results occur due to the implementation.

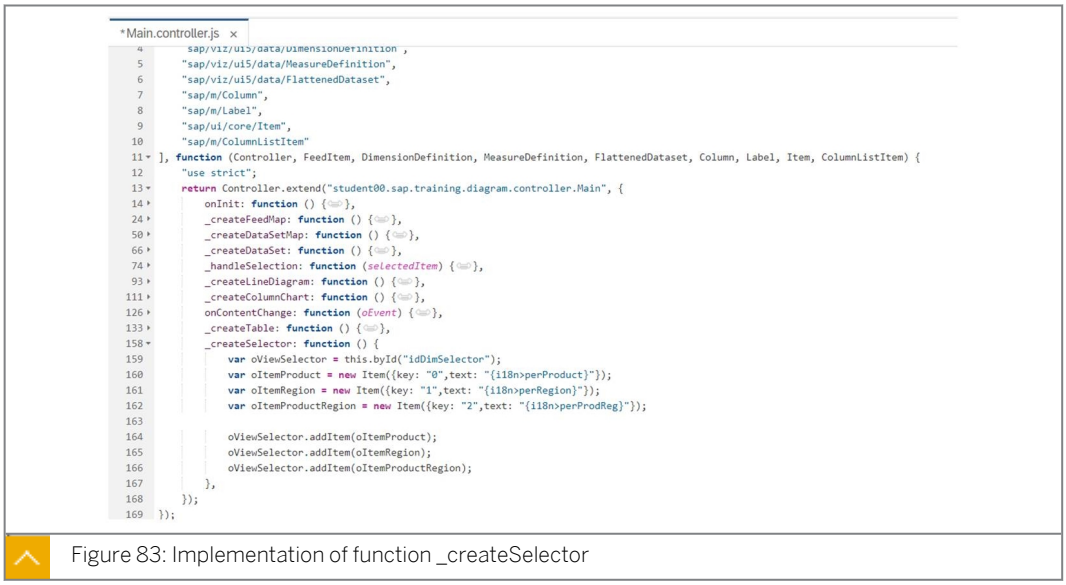


Figure 83: Implementation of function _createSelector

c) Enter the appropriate properties in the I18N file and input the text from the previous sub-step.

d) Open the `Main.controller.js` file and implement the function `onChange`. The function reads out two values: the *selectedItem* parameter from the event object and the key of the selected item in the list using the `getKey` function. Then select the function `_handleSelection` and transfer the value you got from the key parameter of the event.

e) Open the `Main.controller.js` file and implement the function `onChange`. The function reads out two values from the *selectedItem* parameter of the event object.

Figure 84: Function to Read two Values

f) From the onInit function, invoke the function `_createSelector`.

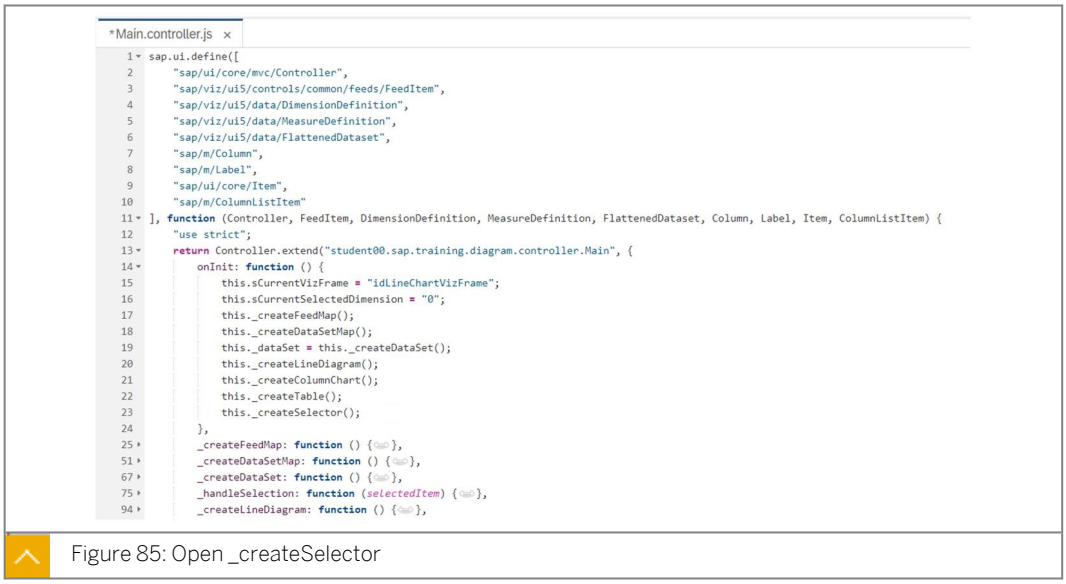


Figure 85: Open _createSelector

g) Expand the `_handleSelection` function, as shown in the following figure:

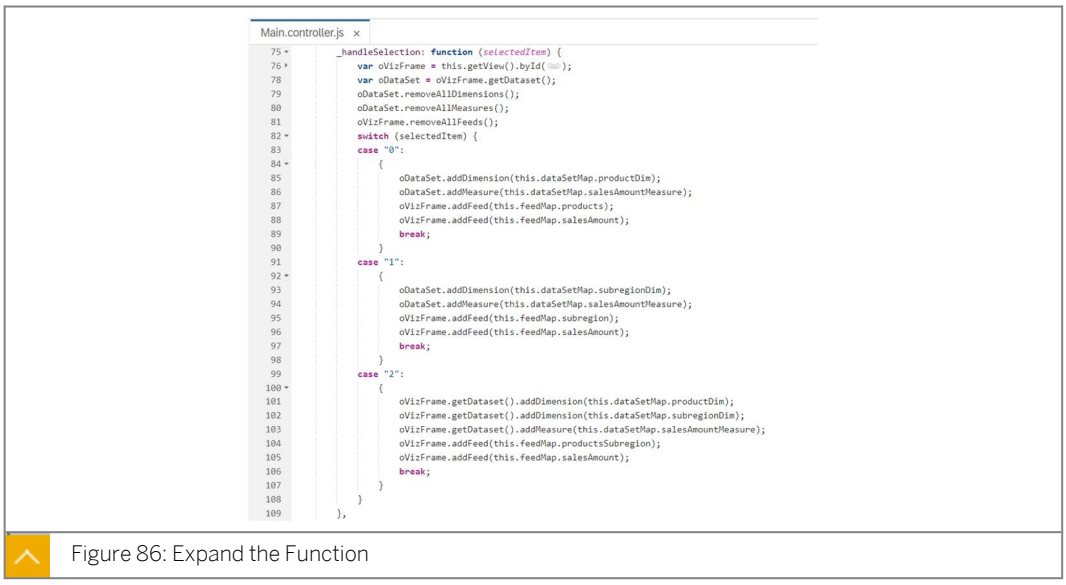


Figure 86: Expand the Function

h) Open your application. A list appears in which you can specify in which relation the sales figures are displayed.

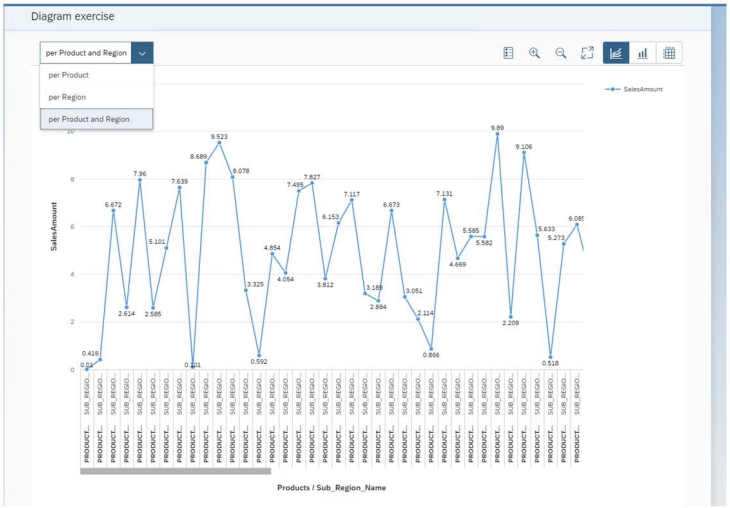


Figure 87: Relationship of Sales Figures

3. To display more meaningful data in the charts rename the file `SalesFigures.json` located in the localService folder to `SalesFigures_Gen.json`. Import the file `SalesFigures.json` located at `s:\courses\UX410_20\templates\diagram\` into the localService folder. Change the Mock Data configuration of your project so that the data from the`SalesFigures.json` file is used.

   a) Select the `SalesFigures.json` file from the context menu.

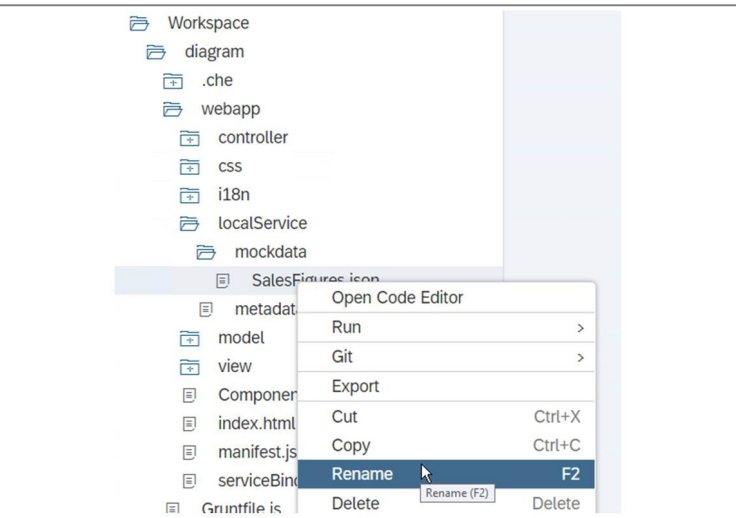


Figure 88: Context Menu

   b) Enter a full name in the dialog and choose *Rename*.

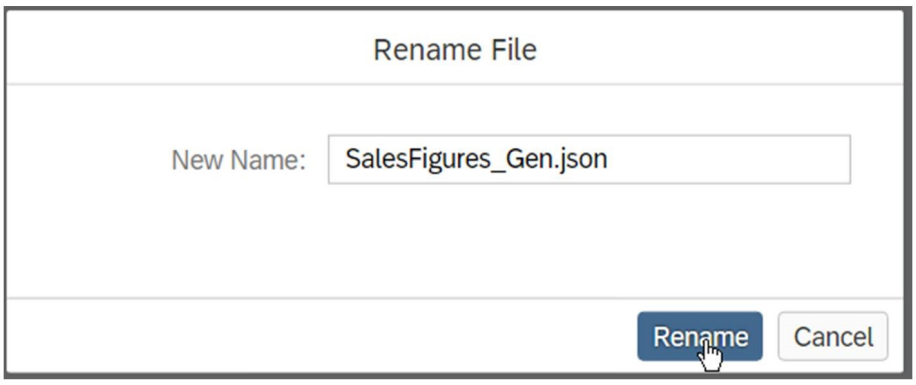


Figure 89: Rename File

c) Import File or Project and navigate to `S:\Courses\UX410_20\templates\diagram`.

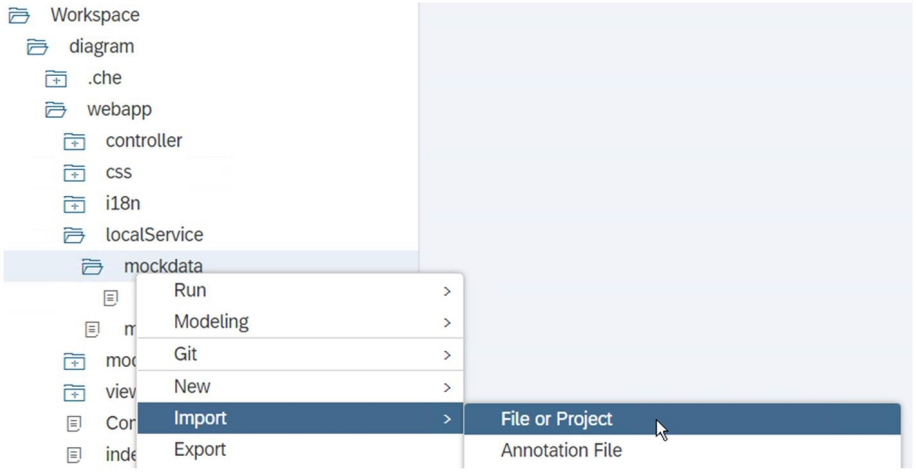


Figure 90: Import File or Project

d) Select the `SalesFigures.json` file and confirm *OK*.

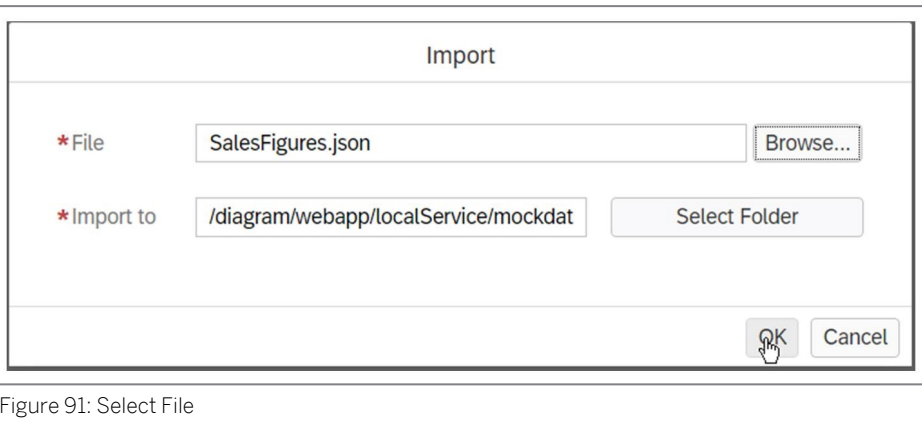


Figure 91: Select File

e) Choose *Project* → *Project Settings*.

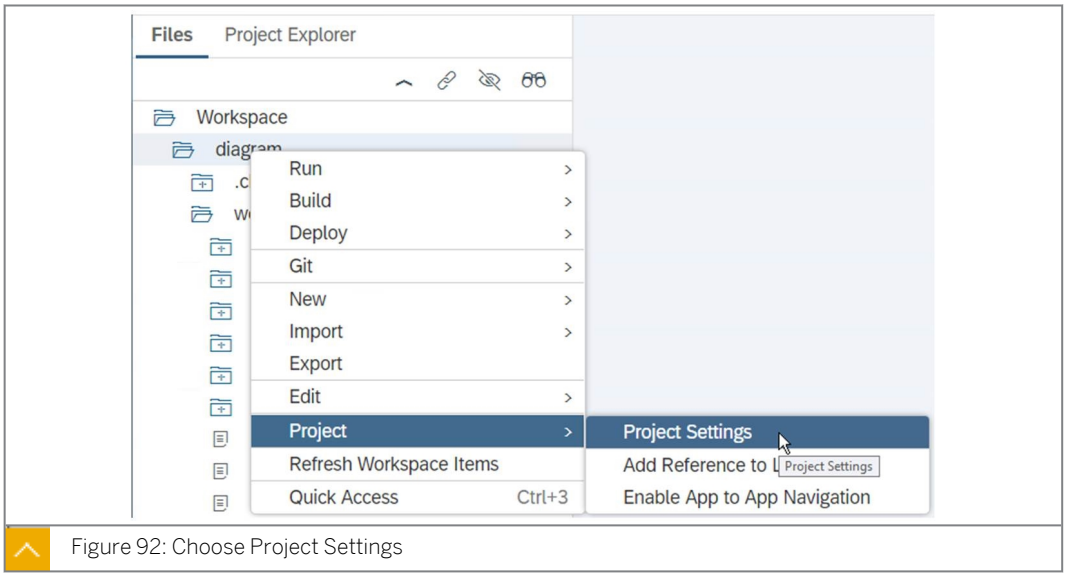


Figure 92: Choose Project Settings

f) Choose *Mock Data* with the option *JSON* files and *Save*.

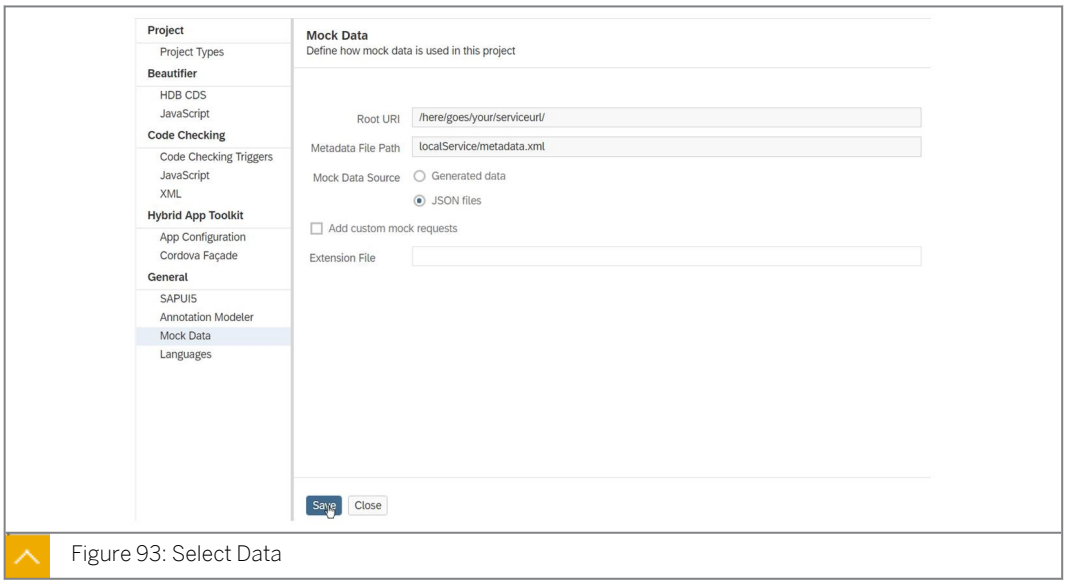


Figure 93: Select Data

g) Restart your application.

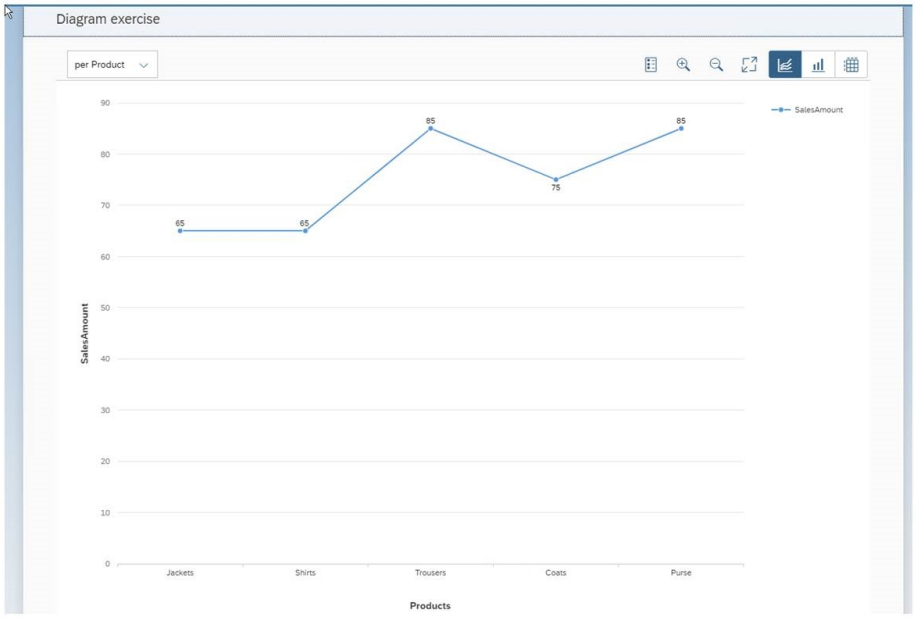


Figure 94: Dimension Selection Implementation

Unit 10
Exercise 8

Navigate in SAP Fiori

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, naming of fields and buttons as well as steps may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

Business Example

You need to implement two small SAPUI5 applications and deploy these applications on the front-end server. Then configure SAP Fiori tiles for both of them and implement a link in the first application that offers to navigate to the second application in the *SAP Fiori launchpad*.

You require an account on the front-end server.

Task 1: Create SAP Fiori startnavigation and endnavigation Projects

Log on to the SAP Web IDE Fullstack and create two SAP Fiori projects named **startnavigation** and **endnavigation**. Add an *XML Main View* to both projects and insert UI controls.

1. Create an SAPUI5 project with the following settings.

   Table 3: Startnavigation Properties

Parameter	Value
Project name	startnavigation
Namespace	student##.sap.training
View Type	XML
View Name	Main
SAPUI5 Version	1.60

2. Implement the basic **startnavigation** project by adding a new *sap.ui.layout.VerticalLayout* control to the page control of the *Main.view.xml* and inserting the following UI controls into the *VerticalLayout*.

   Table 4: Insert sap.m.Link properties

Attribute	Value
id	idNavLink

Attribute	Value
text	Navigate to Fiori-app with link

Table 5: Insert sap.m.Button properties

Attribute	Value
text	Navigate to Fiori-app using event handler
press	onPress

3. Deploy the **startnavigation** project to the front-end server with the properties listed in the table.

Table 6: Deploy startnavigation Properties

Property	Value
Package	ZTRAIN_##
Application name	ZUX410NavStar##
Description	Navigation starter

4. Create a SAPUI5 project using the following settings.

Table 7: endnavigation Properties

Parameter	Value
Project name	endnavigation
Namespace	student##.sap.training
View Type	XML
View Name	Main
SAPUI5 Version	1.60

5. Implement the basic **endnavigation** project by adding a new `sap.ui.layout.VerticalLayoutcontrol` to the *page* control of `Main.view.xml` and insert the following UI controls to the `VerticalLayout`.

Table 8: sap.m.Label Properties

Attribute	Value
id	idinfo
text	Hello world

6. Deploy the **endnavigation** project from the front-end server. Use the table data.

Table 9: Deploy endnavigation Properties

Property	Value
Package	ZTRAIN_##
Application name	ZUX410NavEnd##
Description	Navigation target

Task 2: Configure two tiles in the SAP Fiori Launchpad

1. Create two semantic objects on the front-end server using the following details.

   Table 10: Semantic Object, Name and Description

Semantic Object	Semantic Object Name	Semantic Object Description
UX410NavStart##	UX410NavStart##	Semantic Object for start app##
UX410NavEnd##	UX410NavEnd##	Semantic Object for end app##

2. Create a Catalog on the front-end server with the *SAP Fiori launchpad designer* using the attributes from the table: Catalog ID

   Table 11: Catalog ID

Attribute	Value
Title	ZUX410_BC_##
ID	ZUX410_BC_##

Note:
You can find the *SAP Fiori launchpad designer* in the user menu.

3. Create a tile mapping for `ZUX410NavStar00` using the *NavStart* table properties.

   Table 12: Properties for NavStart Tile Mapping

Property	Value
Semantic Object	UX410NavStart##
Action	Display
Application Type	SAPUI5 Fiori App
Title	UX410 start navigation
URL	/sap/bc/ui5_ui5/sap/zux410navstar##

Property	Value
ID	student##.sap.training.startnavigation

4. Create a tile mapping for `ZUX401NavEnd00` with the *NavEnd* table properties.

Table 13: Properties for NavEnd Tile Mapping

Property	Value
Semantic Object	UX410NavEnd##
Action	display
Application Type	SAPUI5 Fiori App
Title	UX410 endnavigation
URL	/sap/bc/ui5_ui5/sap/zux410navend##
ID	student##.sap.training.endnavigation

5. Create two static tiles with the following attributes.

Table 14: Navigation Start Tile

Attribute	Value
Subtitle	Group ##
Use semantic object navigation:	checked
Semantic Object:	UX410NavStart##

Table 15: Navigation End Tile

Attribute	Value
Subtitle:	Group ##
Use semantic object navigation:	checked
Semantic Object:	UX410NavEnd##

Two tile configurations are displayed.

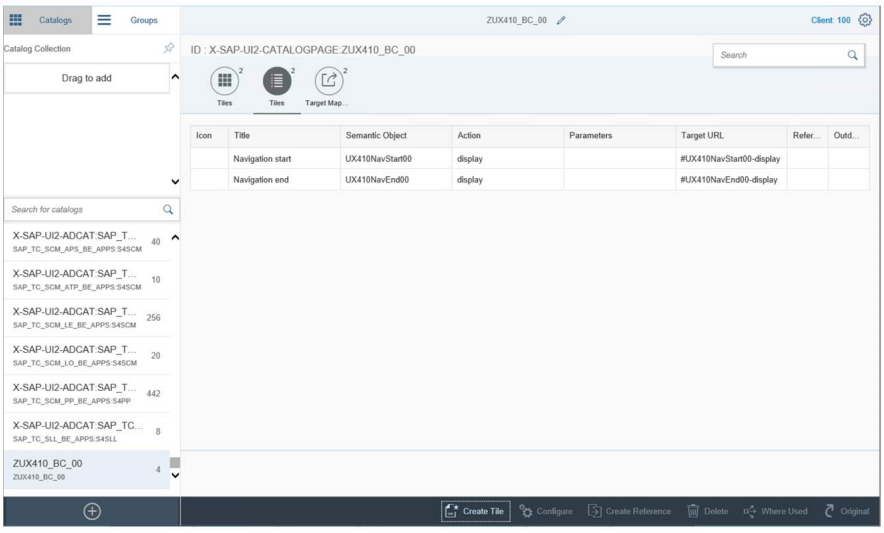


Figure 101: Two Tiles are Displayed

6. Create a new SAP role with the name **`ZUX410_BR_##`** of the type **`Single Role`** on the front-end server and add the new catalog to the role. Assign your user to the role.

7. Log on to the *SAP Fiori launchpad* and add the two new tiles to your homepage.

Task 3: Use the CrossApplicationNavigation Service

Implement the link and button controls to navigate from the *Navigation start* application tile to the *Navigation end* application tile.

1. Start the *SAP Web IDE Full-Stack* to implement the `onInit` function of the starter application. Check the application is running the *SAP Fiori Launchpad*. Obtain a reference to the `CrossApplicationNavigation` service `ShellContainer` and store the reference in a member variable. Create the navigation link to the intent *UX410NavEnd##* display created in the previous task and assign the navigation link to the href attribute of the *sap.m.Link* control that was created in the previous task.

2. Implement the `onPress` event handler in the `Main.controller.js`. Use the reference to the `CrossApplicationNavigation` service and call the function `toExternal`. Pass the **`UX410NavEnd##`** as a semantic object and display as action to the function.

3. Update your deployment of the **startnavigation** project.

4. Log on to the *SAP Fiori launchpad* to test your implementation. Start your application and try the link and button to test the navigation.

Task 4: Pass parameters to the navigation target

Extend the *start navigation application* by passing a parameter to the *end navigation application* during navigation.

1. Extend your implementation of the navigation in the *start navigation application* and pass a parameter with the name *helloText* during the navigation. Assign the string **`Hello UX410`** to the *helloText* parameter.

2. Update your deployment of the **startnavigation** project.

Task 5: Extend the endnavigation application to fetch a parameter

Extend your **endnavigation** application so that when the user navigates from the **startnavigation** application to the **endnavigation** application, the passed parameter is fetched from the component and shown inside the UI.

1. Implement the `onInit` function of the `Main.controller.js` of the **endnavigation** project. Read the *helloText* parameter that is passed by the **startnavigation** application and assign the parameter value to the text attribute of the label that is part of the `Main.view.xml` implementation.
2. Update the deployed application of the endnavigation project.
3. Test your application.

You have now successfully implemented the exercise.

Unit 10
Solution 8

Navigate in SAP Fiori

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, naming of fields and buttons as well as steps may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

Business Example

You need to implement two small SAPUI5 applications and deploy these applications on the front-end server. Then configure SAP Fiori tiles for both of them and implement a link in the first application that offers to navigate to the second application in the *SAP Fiori launchpad*.

You require an account on the front-end server.

Task 1: Create SAP Fiori startnavigation and endnavigation Projects

Log on to the SAP Web IDE Fullstack and create two SAP Fiori projects named **startnavigation** and **endnavigation**. Add an *XML Main View* to both projects and insert UI controls.

1. Create an SAPUI5 project with the following settings.

   Table 3: Startnavigation Properties

Parameter	Value
Project name	startnavigation
Namespace	student##.sap.training
View Type	XML
View Name	Main
SAPUI5 Version	1.60

   a) Select *File* → *New* → *Project from Template* → *SAPUI5 Application Project*, choose the correct SAPUI5 version and *Next*.

   b) Enter **`startnavigation`** as the project name and **`student##.sap.training`** as the *Namespace* field and choose *Next*.

   c) Choose *XML* as a *View Type*, enter **`Main`** as *View Name*, and choose *Finish*.

2. Implement the basic **startnavigation** project by adding a new *sap.ui.layout.VerticalLayout* control to the page control of the *Main.view.xml* and inserting the following UI controls into the *VerticalLayout*.

Table 4: Insert sap.m.Link properties

Attribute	Value
id	idNavLink
text	Navigate to Fiori-app with link

Table 5: Insert sap.m.Button properties

Attribute	Value
text	Navigate to Fiori-app using event handler
press	onPress

a) Open the file `Main.view.xml`.

b) Add an `XML-namespace alias` with the value **`layout`** and assign the value **`sap.ui.layout`**.

c) Add an *sap.ui.layout.VerticalLayout* to the content aggregation of the generated *sap.m.Page* control.

d) Add a control of type *sap.m.Link* to the *VerticalLayout* control and assign the attributes listed in the table, *Insert sap.m.LinkpProperties*.

e) Add a control of type *sap.m.Button* and assign the button attributes listed in the table, *Insert sap.m.Button properties*.

f) Your view implementation should now look like the following figure:

*Main.view.xml ×

```
1  <mvc:View controllerName="student00.sap.training.startnavigation.controller.Main" xmlns:mvc="sap.ui.core.mvc" displayBlock="true"
2      xmlns="sap.m" xmlns:layout="sap.ui.layout">
3      <Shell id="shell">
4          <App id="app">
5              <pages>
6                  <Page id="page" title="{i18n>title}">
7                      <content>
8                          <layout:VerticalLayout>
9                              <Link id="idNavLink" text="Navigate to Fiori-app with link"/>
10                             <Button press="onPress" text="Navigate to Fiori-app using event handler"/>
11                         </layout:VerticalLayout>
12                     </content>
13                 </Page>
14             </pages>
15         </App>
16     </Shell>
17 </mvc:View>
```

Figure 95: View Implementation

g) Save your changes.

3. Deploy the **startnavigation** project to the front-end server with the properties listed in the table.

Table 6: Deploy startnavigation Properties

Property	Value
Package	ZTRAIN_##
Application name	ZUX410NavStar##

Property	Value
Description	Navigation starter

a) Select the **startnavigation** project in the project explorer of the *SAP Web IDE Full-Stack*.

b) Choose *Deploy → Deploy to SAPUI5 ABAP Repository* from the context menu.

c) Select the system alias *FSD_100* and enter your credentials for the front-end server.

d) Select *Deploy a new application* and choose *Next*.

e) Enter data from the table for the name and description fields using your own student and group number.

f) Browse and search for the package `ZTRAIN_##`, where ## is your own group number.

g) Choose *OK* and then *Next.*.

h) Check your dialog looks as shown.

i) Select the transport request that is assigned to your group number from the list and choose *Next*.

j) Start the deployment process by choosing*Finish*.

4. Create a SAPUI5 project using the following settings.

Table 7: endnavigation Properties

Parameter	Value
Project name	endnavigation
Namespace	student##.sap.training
View Type	XML
View Name	Main
SAPUI5 Version	1.60

a) Select *File → New → Project from Template → SAPUI5 Application Project* , choose *SAPUI5-version 1.60* and choose *Next*.

b) Name your project **`endnavigation`** and in the *Namespace* field enter **`student##.sap.training`** and choose *Next*.

c) Select **`XML`** as a *View Type* and **`Main`** in the *View Name* field and choose *Finish*.

5. Implement the basic **endnavigation** project by adding a new `sap.ui.layout.VerticalLayoutcontrol` to the *page* control of `Main.view.xml` and insert the following UI controls to the `VerticalLayout`.

Table 8: sap.m.Label Properties

Attribute	Value
id	idinfo

Attribute	Value
text	Hello world

a) Open the file `Main.view.xml`.

b) Add *XML-namespace* alias with the value **`layout`** and assign the value **`sap.ui.layout`**.

c) Add an `sap.ui.layout.VerticalLayout` to the content aggregation of the generated `sap.m.Page` control.

d) Add a control `sap.m.Label` to the `VerticalLayout` control and assign the attributes listed in the table. Your implementation should now look like the following figure:

Figure 96: Implementation

e) Save your changes.

6. Deploy the **endnavigation** project from the front-end server. Use the table data.

Table 9: Deploy endnavigation Properties

Property	Value
Package	ZTRAIN_##
Application name	ZUX410NavEnd##
Description	Navigation target

a) Select the **endnavigation** project in the *project explorer* of the *SAP Web IDE Full-Stack*.

b) From the context menu, choose *Deploy* → *Deploy to SAPUI5 ABAP Repository*.

c) Select the system alias *FSD_100* and enter your credentials for the front-end server.

d) Select *Deploy a new application* and choose *Next*.

e) Enter the name and description with the provided data using your own student and group number.

f) Browse and search for the package `ZTRAIN_##`, where ## is your own number, choose *OK* and then *Next.*.

g) Select the transport request that is assigned to your group number from the list and choose *Next.*

h) Start the deployment process by selecting *Finish*.

Task 2: Configure two tiles in the SAP Fiori Launchpad

1. Create two semantic objects on the front-end server using the following details.

   Table 10: Semantic Object, Name and Description

Semantic Object	Semantic Object Name	Semantic Object Description
UX410NavStart##	UX410NavStart##	Semantic Object for start app##
UX410NavEnd##	UX410NavEnd##	Semantic Object for end app##

   a) Start *SAP GUI* and log in to the front-end server using your user *Train-##* login details.

   b) From the user menu of the SAP Fiori folder, choose *Define Semantic Object – Customer* or use the transaction code `/UI2/SEMOBJ`.

> Note:
> Add the prefix **`/n`** if you start the transaction using the transaction code.

   c) Choose the edit icon  to switch to edit mode.

   d) Confirm the cross-client message.

> Note:
> If a message informs you that the table is currently locked by another user, this means that another course participant is currently running the transaction for their exercise. Confirm the message, wait until the other participant has finished, and then try again.

   e) Choose *New Entries* on the toolbar.

   f) Add the semantic object configuration with the data from the Table: Semantic Object, Name and Description.

   g) Save your changes and assign them to your transport request.

   h) Confirm the transport assignment dialog and close the transaction.

> Note:
> It is important to close the transaction as soon as you have finished your configuration. If you do not leave, you will lock the table.

2. Create a Catalog on the front-end server with the *SAP Fiori launchpad designer* using the attributes from the table: Catalog ID

Table 11: Catalog ID

Attribute	Value
Title	ZUX410_BC_##
ID	ZUX410_BC_##

Note:
You can find the *SAP Fiori launchpad designer* in the user menu.

a) Start the *SAP Fiori launchpad designer* (client-spec) from the start menu or use transaction **`/UI2/FLPD_CUST`**.

b) When the SAP Fiori launchpad designer has started, assign a customizing request to record your changes by unchecking the checkbox *None* (Local Object), and selecting a customizing request. This is provided by your trainer from the list.

c) Choose  to create a new catalog.

d) Enter the table values in the dialog and choose *Save*.

3. Create a tile mapping for `ZUX410NavStar00` using the *NavStart* table properties.

Table 12: Properties for NavStart Tile Mapping

Property	Value
Semantic Object	UX410NavStart##
Action	Display
Application Type	SAPUI5 Fiori App
Title	UX410 start navigation
URL	/sap/bc/ui5_ui5/sap/zux410navstar##
ID	student##.sap.training.startnavigation

a) Select *Target Mapping* inside your new catalog.

b) Select *Create Target Mapping* in the footer.

c) When prompted for credentials for the back-end system, enter your user name and password and choose *OK*, then *Save*.

After the successful creation of your target mapping, you will see the newly-created mapping in the list of available target mappings.

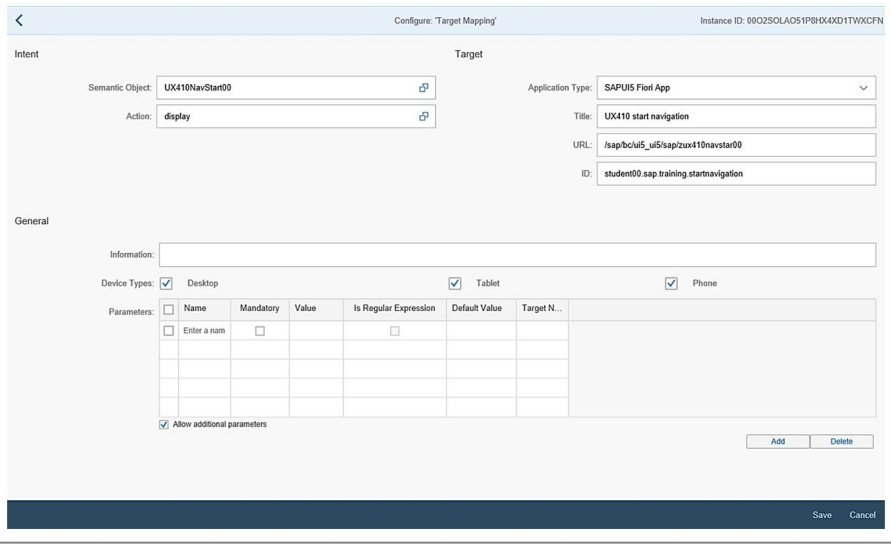


Figure 97: Configure Target mapping

4. Create a tile mapping for `ZUX401NavEnd00` with the *NavEnd* table properties.

Table 13: Properties for NavEnd Tile Mapping

Property	Value
Semantic Object	UX410NavEnd##
Action	display
Application Type	SAPUI5 Fiori App
Title	UX410 endnavigation
URL	/sap/bc/ui5_ui5/sap/zux410navend##
ID	student##.sap.training.endnavigation

a) Select *Target Mapping* inside your new catalog.

b) Select *Create Target Mapping* in the footer.

c) When asked for credentials for the back-end system, enter your user name and password and choose *OK* and *Save*.

After the successful creation of your target mapping, you will see the newly-created mapping in the list of available target mappings.

d) Configure the target mapping.

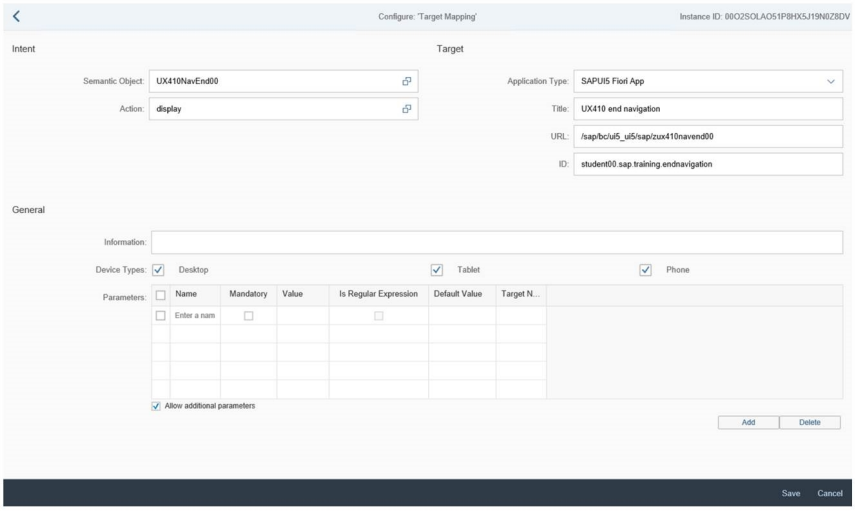


Figure 98: Configuration of the Target Mapping

Your new target mapping is added to the list of available target mappings.

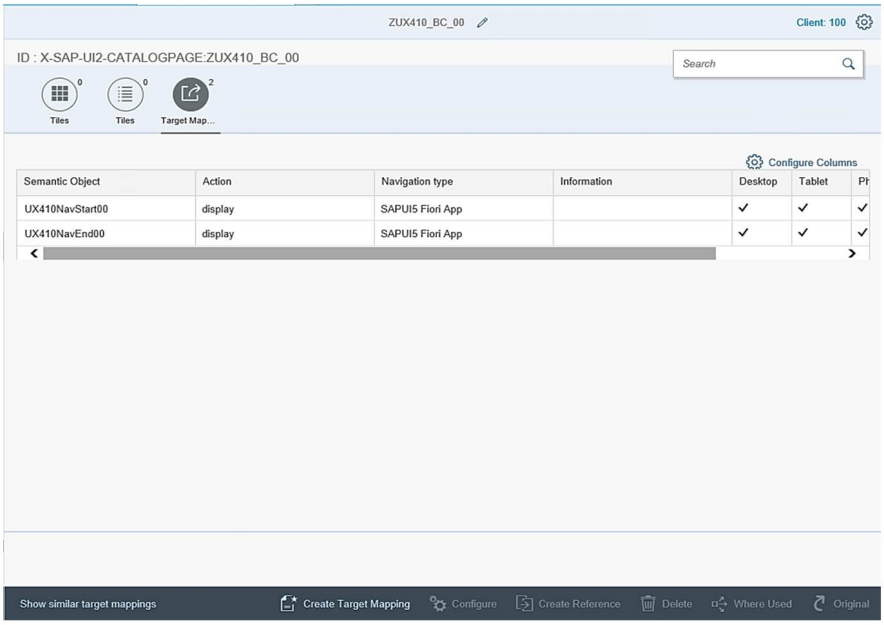


Figure 99: New Target Mapping

5. Create two static tiles with the following attributes.

Table 14: Navigation Start Tile

Attribute	Value
Subtitle	Group ##
Use semantic object navigation:	checked
Semantic Object:	UX410NavStart##

Table 15: Navigation End Tile

Attribute	Value
Subtitle:	Group ##
Use semantic object navigation:	checked
Semantic Object:	UX410NavEnd##

a) Click on *Tiles* and choose *Create Tile* in the footer.

b) Select *App Launcher – Static*.

c) Enter the values as provided and *Save*.

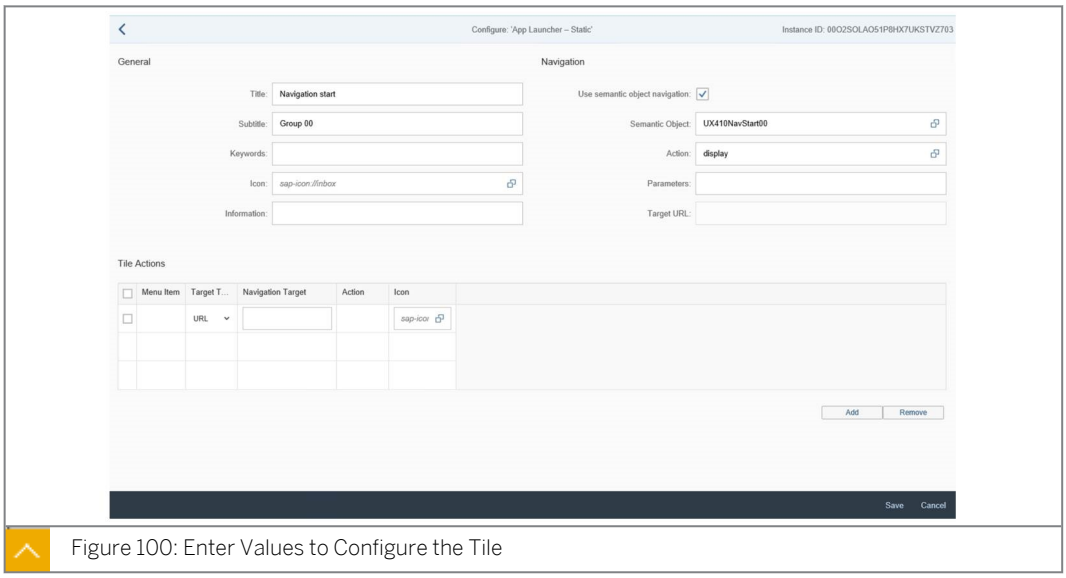


Figure 100: Enter Values to Configure the Tile

d) Repeat to create a tile for the second target.

Two tile configurations are displayed.

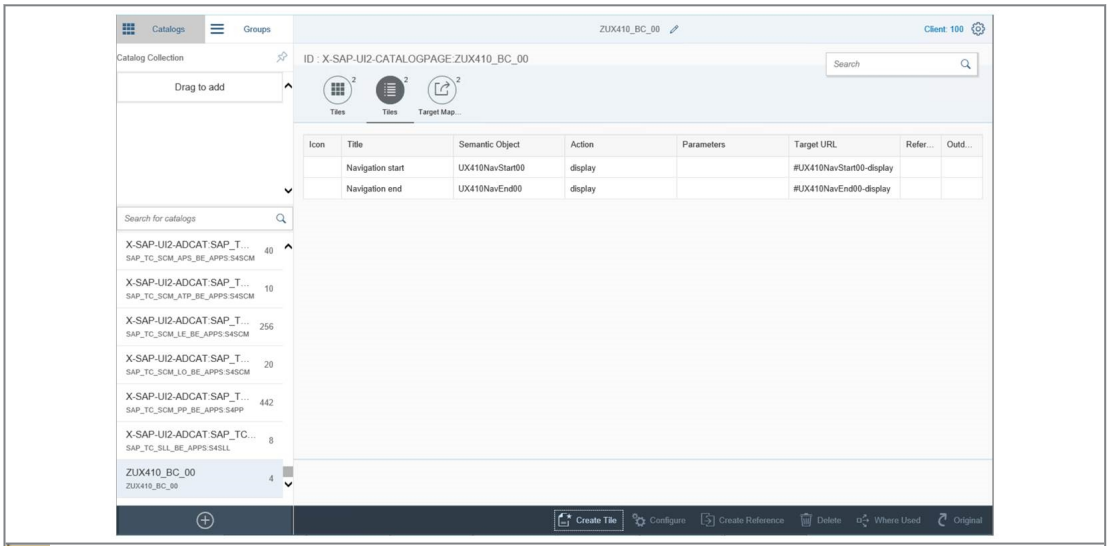


Figure 101: Two Tiles are Displayed

6. Create a new SAP role with the name **ZUX410_BR_##** of the type **Single Role** on the front-end server and add the new catalog to the role. Assign your user to the role.
   a) Log on to the front-end server using *SAP GUI*.
   b) Start transaction `PFCG`.
   c) Enter the role name **ZUX410_BR_##**, select the *Single Role* button and then *Save*.
   d) Choose the *Menu* tab.
   e) On the menu tab, choose the small arrow on the right of the *Transaction* button to open a drop-down list. Choose *SAP Fiori Tile Catalog*.
   f) Choose your *SAP Fiori Catalog* from the *Value Help* and confirm the dialog.
   g) Select the *User* tab and assign your **Train-##** user to the new role.
   h) Save your changes.
7. Log on to the *SAP Fiori launchpad* and add the two new tiles to your homepage.
   a) Open Google Chrome and from the bookmarks choose, *00 Development → fsdhost → FSD Fiori Launchpad (via WDF)*.
   b) Log on to *SAP Fiori launchpad* using **Train-##**, with your group number and password.
   c) When asked for credentials for the back-end system, enter your user name and password and choose *OK* and then *Save*.
   d) Open the *Me* area.
   e) Start the *App Finder*.
   f) Choose your new catalog to see the two tiles display.
   g) Choose the *Pin* icon and then pin to *My Home* and choose *OK*.
   h) Repeat with the other tile and check that both pins are now highlighted.
   i) Leave the *App Finder* and see both tiles in your workspace.

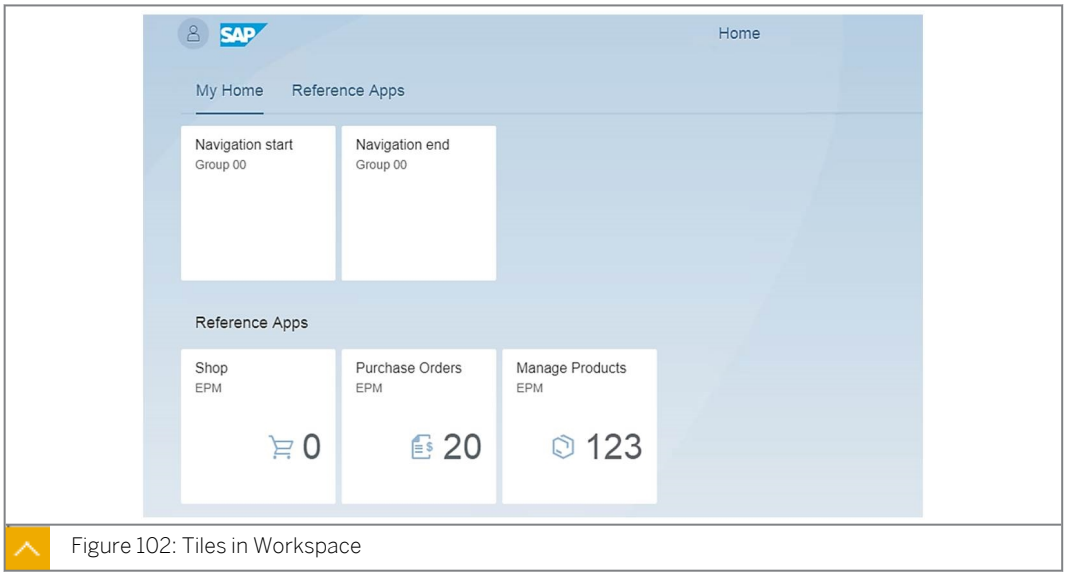


Figure 102: Tiles in Workspace

   j) Start each application tile to test your implementation.

Task 3: Use the CrossApplicationNavigation Service

Implement the link and button controls to navigate from the *Navigation start* application tile to the *Navigation end* application tile.

1. Start the *SAP Web IDE Full-Stack* to implement the `onInit` function of the starter application. Check the application is running the *SAP Fiori Launchpad*. Obtain a reference to the `CrossApplicationNavigation` service `ShellContainer` and store the reference in a member variable. Create the navigation link to the intent *UX410NavEnd##* display created in the previous task and assign the navigation link to the href attribute of the *sap.m.Link* control that was created in the previous task.

   a) Open the `Main.controller.js` file of the **startnavigation** project.

   b) Create an empty implementation of the `onInit` function.

   c) Check whether the application is running the *SAP Fiori launchpad.*

   d) Obtain a reference to the `CrossApplicationNavigation` service and assign the reference to a member variable.

```
Main.controller.js
sap.ui.define([
    "sap/ui/core/mvc/Controller"
], function (Controller) {
    "use strict";

    return Controller.extend("student00.sap.training.startnavigation.controller.Main", {
        onInit :  function() {
            this._fnGetService = sap.ushell && sap.ushell.Container && sap.ushell.Container.getService;
            this._oCrossAppNavigation = this._fnGetService && this._fnGetService("CrossApplicationNavigation");


        }
    });
});
```

Figure 103: Obtain and Assign the Reference

   e) Invoke the function `hrefForExternal` and pass the *UX410NavEnd##* as an `semanticObject` and display as an action.

   f) Assign the return value of the `hrefForExternal` function to a local variable.

```
Main.controller.js
sap.ui.define([
    "sap/ui/core/mvc/Controller"
], function (Controller) {
    "use strict";

    return Controller.extend("student00.sap.training.startnavigation.controller.Main", {
        onInit :  function() {
            this._fnGetService = sap.ushell && sap.ushell.Container && sap.ushell.Container.getService;
            this._oCrossAppNavigation = this._fnGetService && this._fnGetService("CrossApplicationNavigation");

            if(this._oCrossAppNavigation) {
                var oHref = this._oCrossAppNavigation.hrefForExternal({
                target : {
                    semanticObject : "UX410NavEnd##",
                    action : "display"
                }
                }) || "";
            }
        }
    });
});
```

Figure 104: Assign Return Value

g) Obtain a reference to the *sap.m.Link* control from the `Main.view.xml` and assign the reference to a local variable.

h) Call the `setHref` function on the obtained reference and pass the intent reference to the function.

Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller"
], function (Controller) {
    "use strict";

    return Controller.extend("student00.sap.training.startnavigation.controller.Main", {
        onInit :  function() {
            this._fnGetService = sap.ushell && sap.ushell.Container && sap.ushell.Container.getService;
            this._oCrossAppNavigation = this._fnGetService && this._fnGetService("CrossApplicationNavigation");

            if(this._oCrossAppNavigation) {
                var oHref = this._oCrossAppNavigation.hrefForExternal({
                    target : {
                        semanticObject : "UX410NavEnd##",
                        action : "display"
                    }
                }) || "";

                var oLink = this.byId("idNavLink");
                oLink.setHref(oHref);
            }
        }
    });
});
```

Figure 105: Reference

2. Implement the `onPress` event handler in the `Main.controller.js`. Use the reference to the `CrossApplicationNavigation` service and call the function `toExternal`. Pass the **`UX410NavEnd##`** as a semantic object and display as action to the function.

a) Open the `Main.controller.js` file.

Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller"
], function (Controller) {
    "use strict";

    return Controller.extend("student00.sap.training.startnavigation.controller.Main", {
        onInit: function () {
            this._fnGetService = sap.ushell && sap.ushell.Container && sap.ushell.Container.getService;
            this._oCrossAppNavigation = this._fnGetService && this._fnGetService("CrossApplicationNavigation");

            if (this._oCrossAppNavigation) {
                var oHref = this._oCrossAppNavigation.hrefForExternal({
                    target: {
                        semanticObject: "UX410NavEnd00",
                        action: "display"
                    },
                    params : {
                        "helloText" : "Hello UX410"
                    }
                }) || "";

                var oLink = this.byId("idNavLink");
                oLink.setHref(oHref);
            }
        },
```

Figure 106: Main.controller.js

b) Implement a function with the name `onPress` as an event handler.

Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller"
], function (Controller) {
    "use strict";

    return Controller.extend("student00.sap.training.startnavigation.controller.Main", {
        onInit :  function() {
            this._fnGetService = sap.ushell && sap.ushell.Container && sap.ushell.Container.getService;
            this._oCrossAppNavigation = this._fnGetService && this._fnGetService("CrossApplicationNavigation");

            if(this._oCrossAppNavigation) {
                var oHref = this._oCrossAppNavigation.hrefForExternal({
                    target : {
                        semanticObject : "UX410NavEnd##",
                        action : "display"
                    }
                }) || "";

                var oLink = this.byId("idNavLink");
                oLink.setHref(oHref);
            }
        },

        onPress : function(oEvent) {

        }
    });
});
```

Figure 107: onPress Function

c) Check the cross-navigation service is available and call the function `toExternal` on the service reference.

Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller"
], function (Controller) {
    "use strict";

    return Controller.extend("student00.sap.training.startnavigation.controller.Main", {
        onInit :  function() {
            this._fnGetService = sap.ushell && sap.ushell.Container && sap.ushell.Container.getService;
            this._oCrossAppNavigation = this._fnGetService && this._fnGetService("CrossApplicationNavigation");

            if(this._oCrossAppNavigation) {
                var oHref = this._oCrossAppNavigation.hrefForExternal({
                    target : {
                        semanticObject : "UX410NavEnd##",
                        action : "display"
                    }
                }) || "";

                var oLink = this.byId("idNavLink");
                oLink.setHref(oHref);
            }
        },

        onPress : function(oEvent) {
            if(this._oCrossAppNavigation) {
                this._oCrossAppNavigation.toExternal( {
                        target : {
                        semanticObject : "UX410NavEnd##",
                        action : "display"
                    }
                });
            }
        }
    });
});
```

Figure 108: Call the Function toExternal

d) Pass *UX410NavEnd##* as a semantic object and the value **`display`** as an action.

e) Save your changes.

3. Update your deployment of the **startnavigation** project.

a) Select your **startnavigation** project in the project explorer and choose from the context menu *Deploy → Deploy to SAPUI5 ABAP Repository*.

b) Enter your user credentials for the front-end server when required.

c) Check the radio button *Update an existing application* and choose *Next*.

d) Check in the new dialog that the correct application is preselected and choose *Next*.

e) Choose the *Finish* button to deploy the application.

f) Confirm the deployment by choosing *OK*.

g) Wait until the deployment is finished.

4. Log on to the *SAP Fiori launchpad* to test your implementation. Start your application and try the link and button to test the navigation.

a) Select the bookmark in Google Chrome to start the *SAP Fiori launchpad*.

b) Enter your log on credentials.

c) Start the *Starter Application*.

d) Select the *Link* to display the target.

e) Navigate back to the Navigation start by using the back button of the *SAP Fiori launchpad*.

f) Select the *Button* control to display the target.

Task 4: Pass parameters to the navigation target

Extend the *start navigation application* by passing a parameter to the *end navigation application* during navigation.

1. Extend your implementation of the navigation in the *start navigation application* and pass a parameter with the name *helloText* during the navigation. Assign the string **`Hello UX410`** to the *helloText* parameter.

a) Start the *SAP Web IDE Full-Stack*, if not already started.

b) Open the `Main.controller.js` file of the starter application.

c) Enhance the `onInit` function of your controller by passing the configuration `object params` to the `hrefForExternal` function and assign the parameter *helloText* with value **`Hello UX410`** to the parameter.

Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller"
], function (Controller) {
    "use strict";

    return Controller.extend("student00.sap.training.startnavigation.controller.Main", {
        onInit: function () {
            this._fnGetService = sap.ushell && sap.ushell.Container && sap.ushell.Container.getService;
            this._oCrossAppNavigation = this._fnGetService && this._fnGetService("CrossApplicationNavigation");

            if (this._oCrossAppNavigation) {
                var oHref = this._oCrossAppNavigation.hrefForExternal({
                    target: {
                        semanticObject: "UX410NavEnd00",
                        action: "display"
                    },
                    params : {
                        "helloText" : "Hello UX410"
                    }
                }) || "";

                var oLink = this.byId("idNavLink");
                oLink.setHref(oHref);
            }
        },

        onPress: function (oEvent) {
            if (this._oCrossAppNavigation) {
                this._oCrossAppNavigation.toExternal({
                    target: {
                        semanticObject: "UX410NavEnd00",
                        action: "display"
                    }
                });
            }
        }
    });
});
```

Figure 109: Enhance the onInit Function

d) Enhance the `onPress` function of your controller by passing the configuration `object params` to the `hrefForExternal` function, and assigning the parameter *helloText* with the value **`Hello UX410`** to the parameter.

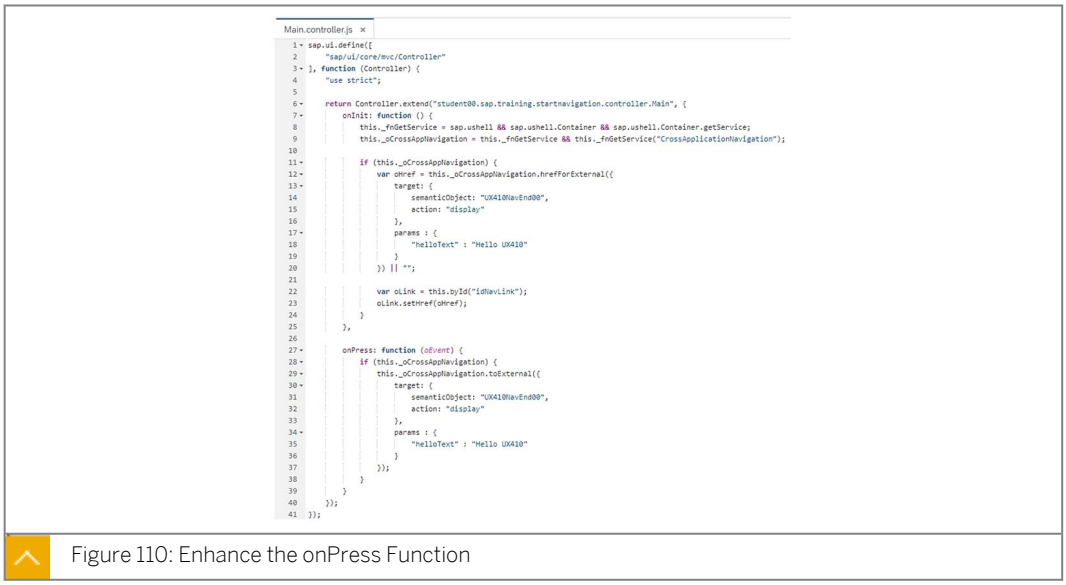


Figure 110: Enhance the onPress Function

2. Update your deployment of the **startnavigation** project.
   a) In the project explorer select your **startnavigation** project and select *Deploy → Deploy to SAPUI5 ABAP Repository* from the context menu.
   b) Enter your user credentials for the front-end server when required.
   c) Check the radio button *Update an existing application* and choose *Next*.
   d) Check the correct application is preselected in the dialog and choose *Next*.
   e) Choose to deploy the application with the *Finish* button.

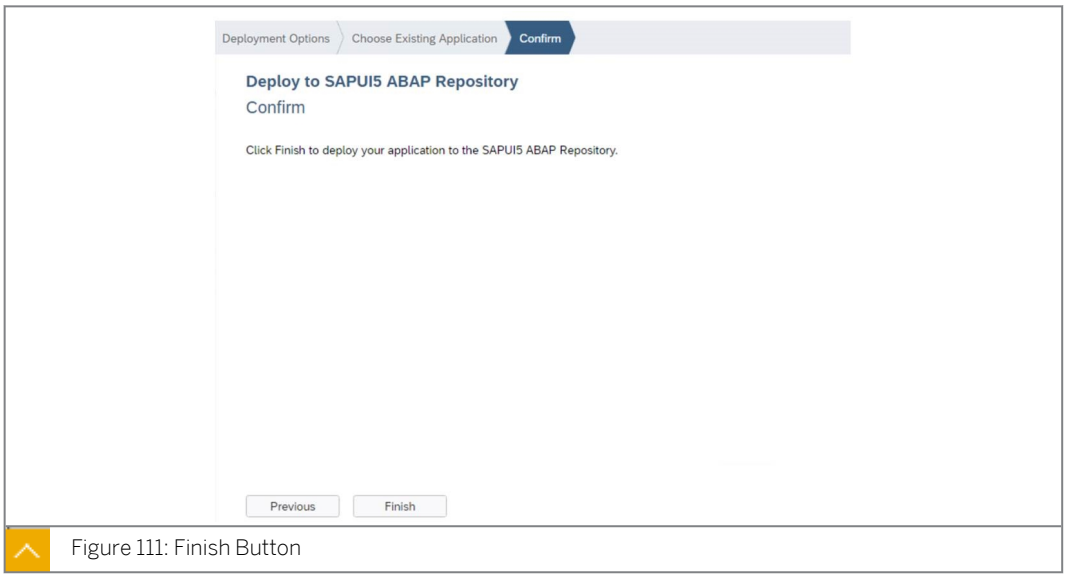


Figure 111: Finish Button

   f) Confirm the deployment by choosing *OK*.
   g) Wait for the deployment to finish.

Task 5: Extend the endnavigation application to fetch a parameter

Extend your **endnavigation** application so that when the user navigates from the **startnavigation** application to the **endnavigation** application, the passed parameter is fetched from the component and shown inside the UI.

1. Implement the `onInit` function of the `Main.controller.js` of the **endnavigation** project. Read the *helloText* parameter that is passed by the **startnavigation** application and assign the parameter value to the text attribute of the label that is part of the `Main.view.xml` implementation.

   a) Open the file `Main.controller.js` of the **endnavigation** project.

   b) Implement an empty `onInit` function inside the controller implementation.

   c) Obtain the component data and assign the component data to a local variable with the name *oComponentData*.

Figure 112: Component Data

   d) Check if the `startupParameters` of the component contains the parameter *helloText*.

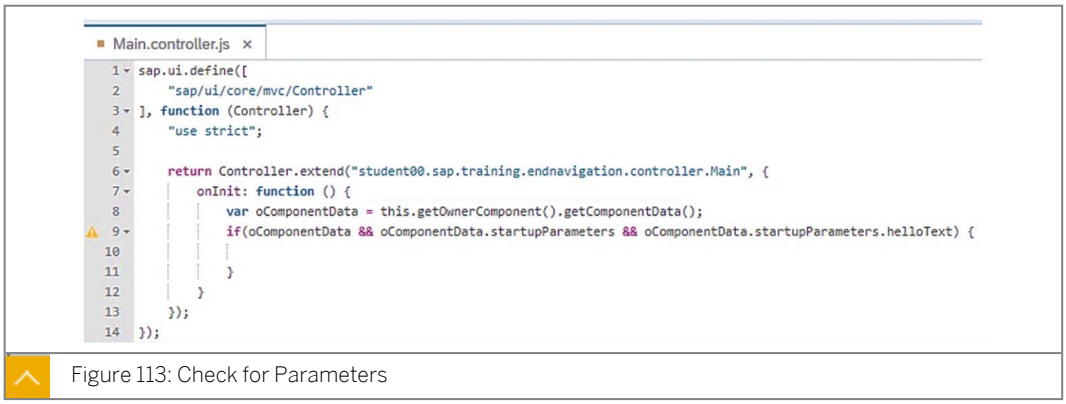


Figure 113: Check for Parameters

   e) Obtain the value of the parameter *helloText* and assign the value to a local variable.

Figure 114: Obtain and Assign Value

f) Obtain a reference to a UI control of type *sap.m.Label* with ID **`idInfo`** and assign the reference to a local variable.

g) Call the `setText` function on the object and pass the `startup` parameter value to the text attribute of the UI control.

Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller"
], function (Controller) {
    "use strict";

    return Controller.extend("student00.sap.training.endnavigation.controller.Main", {
        onInit: function () {
            var oComponentData = this.getOwnerComponent().getComponentData();
            if(oComponentData && oComponentData.startupParameters && oComponentData.startupParameters.helloText) {
                var sHelloText = oComponentData.startupParameters.helloText[0];
                var oLabel = this.byId("idInfo");
                oLabel.setText(sHelloText);
            }
        }
    });
});
```

Figure 115: Call the setText Function

h) Save your changes.

2. Update the deployed application of the endnavigation project.
   a) In the project explorer select your endnavigation project and from the context menu, choose *Deploy → Deploy to SAPUI5 ABAP Repository*.
   b) Enter your user credentials for the front-end server when required.
   c) Select the radio button *Update an existing application* and choose *Next*.
   d) Check in the next dialog that the correct application is preselected and choose *Next*.
   e) Choose *Finish* to deploy your application.
   f) Confirm the deployment by selecting *OK*.
   g) Wait for the deployment to finish.
3. Test your application.
   a) Log on to the *SAP Fiori launchpad* or refresh and reload the current session.
   b) Select the link on the *Navigation Start* tile.
   c) Look for the **`passed startup`** parameter.
   d) Navigate back to the *Navigation start* and select your new button to check if the button implementation works correctly.

You have now successfully implemented the exercise.

Unit 11
Exercise 9

Create a Dynamic Page App

Create a Dynamic Page application using SAP Cloud Platform Developer Edition.

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, naming of fields and buttons as well as steps may differ from the exercise solution. The following figures show student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

Credentials to log on to the SAP Cloud Platform Developer Edition.

1. Create an SAPUI5 project with the following settings:

   Table 16: Project Properties;

Parameter	Value
Project name	dynamicpage
Namespace	student##.sap.training
View Type	XML
View Name	App
SAPUI5 Version	1.60

2. Add the *UX_TRAVEL_SRV OData* service to your project. The service should be referenced by the default model of your application.
3. Remove the generated *sap.m.Page* control in the generated view of `App.view.xml` and remove the pages-aggregation of the generated App-control.
4. Add views of type *XML* to the project. Locate the *views* in the view folder and the *view controller* in the controller folder of your project

   Table 17: View Name Properties

Namespace	View name
student##.sap.training.dynamicpage	Overview
student##.sap.training.dynamicpage	Carrier
student##.sap.training.dynamicpage	NotFound

5. Add the following translations as key value pairs to the default `i18n.properties` file of your project. Add the mentioned documentation note before each section.

Table 18: App Descriptor Properties

Key	Value
appTitle	Exercise:dynamic page app
appDescription	A simple dynamic page app

Table 19: Overview Views Properties

Key	
overviewPageTitle	Carriers
columnId	Id
columnName	Name

Table 20: Carrier View Properties

Key	Value
idLabelText	Id
nameLabelText	Name
currLabelText	Currency
urlLabelText	Url

Table 21: NotFound View Properties

Key	Value
notFoundMessageTitle	Not Found
notFoundMessageText	Sorry, but the requested resource is not available
notFoundMessageDescription	Please check the URL and try again.

6. Implement the `Overview.view.xml` file. The Overview page lists all available carriers from the entity set with the name *UX_C_Carrier_TP*. The Overview implements *sap.f.DynamicPage* control. The DynamicPage control displays a *sap.m.Table* control in the content aggregation. Use *f XML namespace alias* for the *SAPUI5-namspace sap.f*

Table 22: Attribute and Value pairs for the sap.f.DynamicPage control:

Attribute	Value
id	idDynamicPage
headerExpanded	true
toggleHeaderOnTitleClick	true

Table 23: Attribute and Value pairs for sap.m.Table control:

Attribute	Value
items	{UX_C_Carrier_TP}

Table 24: Attribute and Value pairs for sap.m.Title control:

Attribute	Value
text	{i18n>overviewPageTitle}
Level	H2

Table 25: Bind the text attribute to the values listed:

Attribute	Value
text	{i18n>columnId}
text	{i18n>columnName}

Table 26: Attribute and Value pairs for sap.m.ColumnListItem:

Attribute	Value
Press	onPress
Type	Navigation

Table 27: Attribute and Value pairs for a UI control sap.m.ObjectIdentifier:

Attribute	Value
title	{Carrid}

Table 28: Attribute and Value pairs for sap.m.Text:

Attribute	Value
text	{Carrname}

7. Implement the *Carrier view* using an *sap.f.DynamicPage* control. Use the *XML-namespace alias f* for *SAPUI5-namespace sap.f*. Add the listed attributes and values to the *sap.f.DynamicPage* control.

Table 29: Carrier View Properties:

Attribute	Value
id	idCarrierPage
headerExpanded	true

Attribute	Value
toggleHeaderOnTitleClick	true

Table 30: sap.m.Title Attribute and Value:

Attribute	Value
text	{Carrname}

Table 31: sap.m.text Control Attribute and Value:

Attribute	Value
text	{Carrid}

Table 32: Page Header Control Attribute and Value:

Attribute	Value
pinnable	true

Table 33: Flexbox Control Attribute and Value:

Attribute	Value
alignItems	Start
justifyContent	SpaceBetween

Table 34: HorizontalLayout Control Attribute and Value:

Attribute	Value
allowWrapping	true

Table 35: VerticalLayout Control Attribute and Value:

Attribute	Value
class	sapUiMediumMarginEnd

Table 36: ObjectAttribute 1 Control Attribute and Value:

Attribute	Value
title	{i18n>currLabelText}
title	{Currcode}

Table 37: ObjectAttribute 2 Control Attribute and Value:

Attribute	Value
title	{i18n>urlLabel}
title	{Url}

8. Implement `NotFound.view.xml`. Add an *sap.m.MessagePage* control with the following attributes and assign the listed values:

Table 38: Message Page Control Attribute/Value

Attribute	Value
title	{i18n>notFoundMessageTitle}
text	{i18n>notFoundMessageText}
description	{i18n>notFoundMessageDescription}
showNavButton	true
navButtonPress	onNavBack

9. Implement the controller of the carrier view including these functions:

Carrier.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller"
], function (Controller) {
    "use strict";

    return Controller.extend("student00.sap.training.dynamicpage.controller.Carrier", {
        onInit: function () {},

        getRouter: function () {},

        _onObjectMatched: function (oEvent) {},

        _onBindingChange: function () {},

        onNavBack: function () {}

    });

});
```

Figure 120: Implement Controller

10. Implement the *NotFound* controller.

NotFound.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller"
], function (Controller) {
    "use strict";

    return Controller.extend("student00.sap.training.dynamicpage.controller.NotFound", {
        getRouter: function () {
            return sap.ui.core.UIComponent.getRouterFor(this);
        },

        onNavBack: function () {
            var oHistory, sPreviousHash, oRouter;
            oHistory = sap.ui.core.routing.History.getInstance();
            sPreviousHash = oHistory.getPreviousHash();

            if (sPreviousHash !== undefined) {
                window.history.go(-1);
            } else {
                oRouter = this.getRouter();
                oRouter.navTo("Overview", true /*no history*/ );
            }
        }
    });
});
```

Figure 125: Implement the NotFound Controller

11. Implement the *controller* of the *Overview* view.

Overview.controller.js

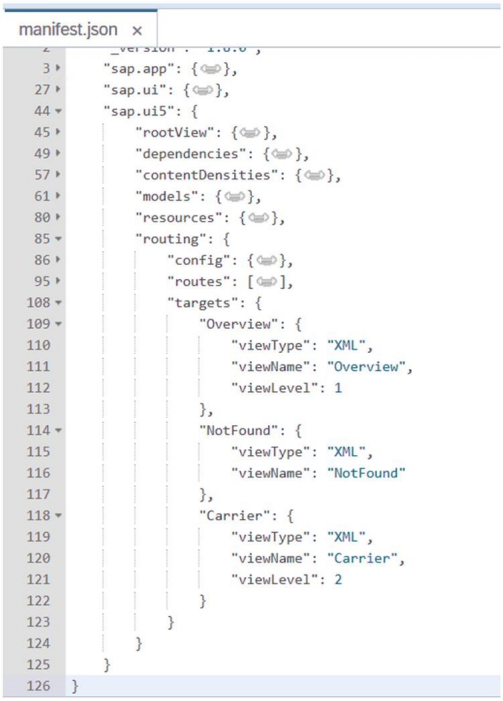
```
sap.ui.define([
    "sap/ui/core/mvc/Controller"
], function (Controller) {
    "use strict";

    return Controller.extend("student00.sap.training.dynamicpage.controller.Overview", {
        getRouter: function () {
            return sap.ui.core.UIComponent.getRouterFor(this);
        },

        onPress: function (oEvent) {
            var oItem, oCtx, sCarrid, oRouter;
            oItem = oEvent.getSource();
            oCtx = oItem.getBindingContext();
            sCarrid = oCtx.getProperty("Carrid");

            oRouter = this.getRouter();
            oRouter.navTo("Carrier", {
                carrierId: sCarrid
            }, false /*with history*/ );
        }
    });
});
```

Figure 126: Implement the Controller

12. Implement the routing definition inside the `manifest.json`. Follow the routing and target configuration.

```
*manifest.json ×
87      "routing": {
88          "config": {
89              "routerClass": "sap.m.routing.Router",
90              "viewType": "XML",
91              "async": true,
92              "viewPath": "student00.sap.training.dynamicpage.view",
93              "controlAggregation": "pages",
94              "controlId": "app",
95              "clearControlAggregation": false
96          },
```

Figure 127: Implement the Routing Definition



```
manifest.json ×
3   "sap.app": {},
27  "sap.ui": {},
44  "sap.ui5": {
45      "rootView": {},
49      "dependencies": {},
57      "contentDensities": {},
61      "models": {},
80      "resources": {},
85      "routing": {
86          "config": {},
95          "routes": [],
108         "targets": {
109             "Overview": {
110                 "viewType": "XML",
111                 "viewName": "Overview",
112                 "viewLevel": 1
113             },
114             "NotFound": {
115                 "viewType": "XML",
116                 "viewName": "NotFound"
117             },
118             "Carrier": {
119                 "viewType": "XML",
120                 "viewName": "Carrier",
121                 "viewLevel": 2
122             }
123         }
124     }
125 }
126 }
```

Figure 128: Implement Definition

13. Test your application.

Unit 11
Solution 9

Create a Dynamic Page App

Create a Dynamic Page application using SAP Cloud Platform Developer Edition.

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, naming of fields and buttons as well as steps may differ from the exercise solution. The following figures show student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

Credentials to log on to the SAP Cloud Platform Developer Edition.

1. Create an SAPUI5 project with the following settings:

   Table 16: Project Properties;

Parameter	Value
Project name	dynamicpage
Namespace	student##.sap.training
View Type	XML
View Name	App
SAPUI5 Version	1.60

   a) Log on to the SAP Web IDE Full-Stack.

   b) Choose *File → New → Project from Template* and select the entry *SAPUI5 Application Project* and *Next.*

   c) Name your project *dynamicpage* and give the namespace as *student##.sap.training*. Choose *Next*.

   d) Choose *XML* as a *View Type* and give the value **App** in the field *View Name* and choose *Finish*.

   e) When your project is created, you can find it in the project explorer of the *SAP Web IDE Full-Stack*.

2. Add the *UX_TRAVEL_SRV OData* service to your project. The service should be referenced by the default model of your application.

   a) Select your project and from the context menu, choose *New-OData service*.

   b) From the list of available systems, choose the *FSD_100 – FSD system*.

   c) Enter your user name and password for the front-end server if required.

d) Search for the *UX_TRAVEL_SRV service* and choose the service from the list of results and choose *Next*.

e) Select the *Radio* button labelled *Use default model* and choose *Next*.

f) Complete the data source assignment by choosing *Finish*.

3. Remove the generated *sap.m.Page* control in the generated view of `App.view.xml` and remove the pages-aggregation of the generated App-control.

a) Open the file `App.view.xml` in the view folder of your project.

b) Remove the *pages* aggregation and the *sap.m.Page* control with the content aggregation.

c) Shorten the app tag implementation.

d) Save your changes.

4. Add views of type *XML* to the project. Locate the *views* in the view folder and the *view controller* in the controller folder of your project

Table 17: View Name Properties

Namespace	View name
student##.sap.training.dynamicpage	Overview
student##.sap.training.dynamicpage	Carrier
student##.sap.training.dynamicpage	NotFound

a) Select the webapp folder of your project and from the context menu, choose *New SAPUI5 View*.

b) Select *XML* as the *view type*. Enter the name of your project in **`namespace`** and *overview* in the *View Name* field. Choose *Next* and *Finish* to exit the view creation dialog.

c) Repeat steps a and b to create the listed views.

d) Your controller and view folders display with your views created as shown.

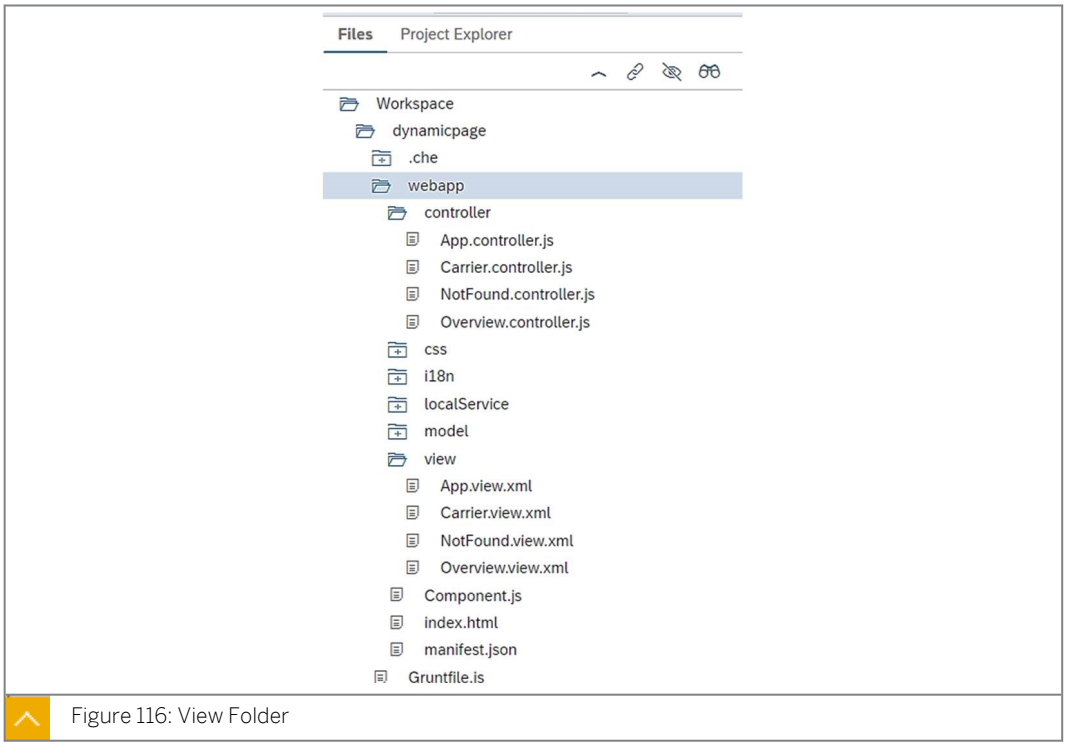


Figure 116: View Folder

5. Add the following translations as key value pairs to the default `i18n.properties` file of your project. Add the mentioned documentation note before each section.

Table 18: App Descriptor Properties

Key	Value
appTitle	Exercise:dynamic page app
appDescription	A simple dynamic page app

Table 19: Overview Views Properties

Key	
overviewPageTitle	Carriers
columnId	Id
columnName	Name

Table 20: Carrier View Properties

Key	Value
idLabelText	Id
nameLabelText	Name
currLabelText	Currency
urlLabelText	Url

Table 21: NotFound View Properties

Key	Value
notFoundMessageTitle	Not Found
notFoundMessageText	Sorry, but the requested resource is not available
notFoundMessageDescription	Please check the URL and try again.

a) Open the file `i18n.properties` in the *i18n* folder of your project.

b) Add the listed key value pairs from the tables to the files.

c) The files appear as shown with the key-value pairs and comments added.

i18n.properties

```
# App Descriptor
appTitle=Exercise: dynamic page app
appDescription=A simple dynamic page app

# Overview view
overviewPageTitle=Carriers
columnId=Id
columnName=Name

# Carrier view
idLabelText=Id
nameLabelText=Name
currLabelText=Currency
urlLabelText=Url

# NotFound View
notFoundMessageTitle=Not Found
notFoundMessageText=Sorry, but the requested resource is not available.
notFoundMessageDescription=Please check the URL and try again.
```

Figure 117: i18n File

d) Save your changes.

6. Implement the `Overview.view.xml` file. The Overview page lists all available carriers from the entity set with the name *UX_C_Carrier_TP*. The Overview implements *sap.f.DynamicPage* control. The DynamicPage control displays a *sap.m.Table* control in the content aggregation. Use *f XML namespace alias* for the *SAPUI5-namspace sap.f*

Table 22: Attribute and Value pairs for the sap.f.DynamicPage control:

Attribute	Value
id	idDynamicPage
headerExpanded	true
toggleHeaderOnTitleClick	true

Table 23: Attribute and Value pairs for sap.m.Table control:

Attribute	Value
items	{UX_C_Carrier_TP}

Table 24: Attribute and Value pairs for sap.m.Title control:

Attribute	Value
text	{i18n>overviewPageTitle}
Level	H2

Table 25: Bind the text attribute to the values listed:

Attribute	Value
text	{i18n>columnId}
text	{i18n>columnName}

Table 26: Attribute and Value pairs for sap.m.ColumnListItem:

Attribute	Value
Press	onPress
Type	Navigation

Table 27: Attribute and Value pairs for a UI control sap.m.ObjectIdentifier:

Attribute	Value
title	{Carrid}

Table 28: Attribute and Value pairs for sap.m.Text:

Attribute	Value
text	{Carrname}

a) Open the file `Overview.view.xml` in the view folder of your project and remove the default implementation of the view.

b) Add a *XML-namespace alias* for the *SAPUI5-namespace sap.f* to the view.

c) Add a UI control of type *sap.f.DynamicPage* with the listed attribute and value pairs.

d) Add the *content* aggregation to the *sap.f.DynamicPage* control.

e) Add an *sap.m.Table* control with the listed attributes to the content aggregation.

f) Add the *headerToolbar* aggregation to the table control.

g) Add an *sap.m.Toolbar* control to the *headerToolbar* aggregation and insert the *content* aggregation in the *sap.m.Toolbar*.

h) Add an *sap.m.Title* control to the *content* aggregation of the *sap.m.Toolbar* with the listed attributes.

i) Add the columns aggregation to the *sap.m.Table* control.

j) Add two *sap.m.Column* controls to the columns aggregation and add to each of the *column* controls a control of type *sap.m.Text* with the listed attribute and value pairs.

k) Add the *items* aggregation to the *sap.m.Table* control.

l) Add a control of type *sap.m. ColumnListItem* as a template control and add the listed attribute and values to the control.

m) Add the *cells* aggregation to *sap.m.ColumnListItemcontrol*.

n) Add the *sap.m.ObjectIdentifier* and the *sap.m.Text* control to the cells aggregation and add the listed attribute and value pairs.

o) Your implementation should like the following figure:

*Overview.view.xml ×

```
<mvc:View xmlns:core="sap.ui.core" xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m" xmlns:f="sap.f"
    controllerName="student00.sap.training.dynamicpage.controller.Overview"
    xmlns:html="http://www.w3.org/1999/xhtml">
    <f:DynamicPage id="idDynamicPage" headerExpanded="true" toggleHeaderOnTitleClick="true">
        <f:content>
            <Table items="{/UX_C_Carrier_TP}">
                <headerToolbar>
                    <Toolbar>
                        <content>
                            <Title text="{i18n>overviewPageTitle}" level="H2"/>
                        </content>
                    </Toolbar>
                </headerToolbar>
                <columns>
                    <Column>
                        <Text text="{i18n>columnId}"/>
                    </Column>
                    <Column>
                        <Text text="{i18n>columnName}"/>
                    </Column>
                </columns>
                <items>
                    <ColumnListItem press="onPress" type="Navigation">
                        <cells>
                            <ObjectIdentifier title="{Carrid}"/>
                            <Text text="{Carrname}"/>
                        </cells>
                    </ColumnListItem>
                </items>
            </Table>
        </f:content>
    </f:DynamicPage>
</mvc:View>
```

Figure 118: Add Controls with Attributes

p) Save your changes.

7. Implement the *Carrier view* using an *sap.f.DynamicPage* control. Use the *XML-namespace alias f* for *SAPUI5-namespace sap.f*. Add the listed attributes and values to the *sap.f.DynamicPage* control.

Table 29: Carrier View Properties:

Attribute	Value
id	idCarrierPage
headerExpanded	true

Attribute	Value
toggleHeaderOnTitleClick	true

Table 30: sap.m.Title Attribute and Value:

Attribute	Value
text	{Carrname}

Table 31: sap.m.text Control Attribute and Value:

Attribute	Value
text	{Carrid}

Table 32: Page Header Control Attribute and Value:

Attribute	Value
pinnable	true

Table 33: Flexbox Control Attribute and Value:

Attribute	Value
alignItems	Start
justifyContent	SpaceBetween

Table 34: HorizontalLayout Control Attribute and Value:

Attribute	Value
allowWrapping	true

Table 35: VerticalLayout Control Attribute and Value:

Attribute	Value
class	sapUiMediumMarginEnd

Table 36: ObjectAttribute 1 Control Attribute and Value:

Attribute	Value
title	{i18n>currLabelText}
title	{Currcode}

Table 37: ObjectAttribute 2 Control Attribute and Value:

Attribute	Value
title	{i18n>urlLabel}
title	{Url}

a) Open the file `Carrier.view.xml` in the view folder of your project and remove the default implementation of the view.

b) Add an *XML-namespace alias f* to the view control and assign *SAPUI5-namespace sap.f*.

c) Add the UI control *sap.f.DynamicPage* to the *view* control and assign the listed attribute and values to the newly added control.

d) Add the *title* aggregation to the newly added *sap.f.DynamicPage* control.

e) Add a UI control of type *sap.f.DynamicPageTitle* to the title aggregation.

f) Add the *heading* aggregation to the *sap.f. DynamicPageTitle* control.

g) Add an *sap.m.Title* control to the *heading* aggregation and add the *text* attribute to the control. Assign the binding to the property *Carrname*.

h) Add the *expandedContent* aggregation to the *sap.f.DynamicPageTitle* control.

i) Add an *sap.m.Title* control to the expanded aggregation and add the *text* attribute to the control. Assign the binding to the property *Carrid*.

j) Add the *snappedContent* aggregation to the *sap.f.DynamicPageTitle* control.

k) Add an *sap.m.Title* control to the *heading* aggregation and add the *text* attribute to the control. Assign the binding to the property *Carrid*.

l) Add the *header* aggregation to the *sap.f.DynamicPage* control.

m) Add a control of type *sap.f.DynamicPageHeader* to the `header.aggregation`. Add the attribute with the name *pinnable* and assign the value **`true`**.

n) Add the *content* aggregation to the *sap.f.DynamicPageHeader* control.

o) Add an *sap.m.FlexBox* control to the content aggregation of the *sap.f.DynamicPageHeader* control. Add the attribute *alignItems* to the *sap.m.FlexBox* control and assign the value **`Start`**. Add the attribute *justifyContent* and assign the value **`SpaceBetween`**.

p) Add an *XML-namespace alias* with name layout and assign the value **`sap.ui.layout`**.

q) Add the *sap.ui.layout.HorizontalLayout* to the *items* aggregation. Add the attribute *allowWrapping* to the *sap.ui.layout.HorizontalLayout* control and assign the value **`true`**.

r) Add an *sap.ui.layout.VerticalLayout* control to the *HorizontalLayout* control. Add the attribute *class* to the *sap.ui.layout.VerticalLayout* control and assign the value **`sapUiMediumMarginEnd`**.

s) Add an *sap.m.ObjectAttribute* control to the *sap.ui.layout.VerticalLayout* control. Add the attribute *title* and assign the binding to key *currLabelText* of the `ResourceModel i18n`. Add the control *text* and bind the attribute to the property *Currcode* of the entity.

t) Add an *sap.m.ObjectAttribute* control to the *sap.ui.layout.VerticalLayout* control. Add the attribute *title* and assign the binding to key **`urlLabel`** of the `ResourceModel i18n`. Add the control *text* and bind the attribute to the property *Url* of the entity.

u) Your implementation should look like the following figure:

Carrier.view.xml

```
<mvc:View xmlns:core="sap.ui.core" xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m"
    xmlns:f="sap.f" xmlns:layout="sap.ui.layout"
    controllerName="student00.sap.training.dynamicpage.controller.Carrier"
    xmlns:html="http://www.w3.org/1999/xhtml">
    <f:DynamicPage id="dynamicPageId" headerExpanded="true" toggleHeaderOnTitleClick="true">
        <f:title>
            <f:DynamicPageTitle>
                <f:heading>
                    <Title text="{Carrname}"/>
                </f:heading>
                <f:expandedContent>
                    <Text text="{Carrid}"/>
                </f:expandedContent>
                <f:snappedContent>
                    <Text text="{travel>Carrid}"/>
                </f:snappedContent>
            </f:DynamicPageTitle>
        </f:title>
        <f:header>
            <f:DynamicPageHeader pinnable="true">
                <f:content>
                    <FlexBox alignItems="Start" justifyContent="SpaceBetween">
                        <items>
                            <layout:HorizontalLayout allowWrapping="true">
                                <layout:VerticalLayout class="sapUiMediumMarginEnd">
                                    <ObjectAttribute title="{i18n>currLabelText}" text="{Currcode}"/>
                                    <ObjectAttribute title="{i18n>urlLabelText}" text="{Url}"/>
                                </layout:VerticalLayout>
                            </layout:HorizontalLayout>
                        </items>
                    </FlexBox>
                </f:content>
            </f:DynamicPageHeader>
        </f:header>
    </f:DynamicPage>
</mvc:View>
```

Figure 119: Added Controls and Attributes

v) Save your work.

8. Implement `NotFound.view.xml`. Add an *sap.m.MessagePage* control with the following attributes and assign the listed values:

Table 38: Message Page Control Attribute/Value

Attribute	Value
title	{i18n>notFoundMessageTitle}
text	{i18n>notFoundMessageText}
description	{i18n>notFoundMessageDescription}
showNavButton	true
navButtonPress	onNavBack

a) Open the file `Notfound.view.xml` and remove the existing *sap.m.App* control.

b) Add an *sap.m.MessagePage* control to the view implementation and use the listed attributes.

c) Save your changes.

9. Implement the controller of the carrier view including these functions:

Carrier.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller"
], function (Controller) {
    "use strict";

    return Controller.extend("student00.sap.training.dynamicpage.controller.Carrier", {
        onInit: function () {},

        getRouter: function () {},

        _onObjectMatched: function (oEvent) {},

        _onBindingChange: function () {},

        onNavBack: function () {}

    });

});
```

Figure 120: Implement Controller

a) Implement the *onInit* function.

Carrier.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller"
], function (Controller) {
    "use strict";

    return Controller.extend("student00.sap.training.dynamicpage.controller.Carrier", {
        onInit: function () {
            var oRouter = this.getRouter();
            oRouter.getRoute("Carrier").attachMatched(this._onObjectMatched, this);
        },
```

Figure 121: Implement the onInit Function

b) Implement the *getRouter* function.

Carrier.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller"
], function (Controller) {
    "use strict";

    return Controller.extend("student00.sap.training.dynamicpage.controller.Carrier", {
        onInit: function () {},

        getRouter: function () {
            return sap.ui.core.UIComponent.getRouterFor(this);
        },

        _onObjectMatched: function (oEvent) {},

        _onBindingChange: function () {},

        onNavBack: function () {}

    });

});
```

Figure 122: Implement the getRouter Function

c) Implement the function *_onObjectMatched*.

Carrier.controller.js

```
    _onObjectMatched: function (oEvent) {
        var oArgs, oView;
        oArgs = oEvent.getParameter("arguments");
        this._sCarrierId = oArgs.carrierId;
        oView = this.getView();

        oView.bindElement({
            path: "/UX_C_Carrier_TP('" + this._sCarrierId + "')",
            events: {
                change: this._onBindingChange.bind(this),
                dataRequested: function () {
                    oView.setBusy(true);
                },
                dataReceived: function () {
                    oView.setBusy(false);
                }
            }
        });
    },
```

Figure 123: Implement _onObjectMatched Function

d) Implement the *onNavBack* function.

Carrier.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller"
], function (Controller) {
    "use strict";

    return Controller.extend("student00.sap.training.dynamicpage.controller.Carrier", {
        onInit: function () {},

        getRouter: function () {},

        _onObjectMatched: function (oEvent) {},

        _onBindingChange: function () {},

        onNavBack: function () {
            var oHistory, sPreviousHash, oRouter;

            oHistory = sap.ui.core.routing.History.getInstance();
            sPreviousHash = oHistory.getPreviousHash();

            if (sPreviousHash !== undefined) {
                window.history.go(-1);
            } else {
                oRouter = this.getRouter();
                oRouter.navTo("Overview", true /*no history*/ );
            }
        }

    });

});
```

Figure 124: Implement the onNavBack Function

10. Implement the *NotFound* controller.

NotFound.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller"
], function (Controller) {
    "use strict";

    return Controller.extend("student00.sap.training.dynamicpage.controller.NotFound", {
        getRouter: function () {
            return sap.ui.core.UIComponent.getRouterFor(this);
        },

        onNavBack: function () {
            var oHistory, sPreviousHash, oRouter;
            oHistory = sap.ui.core.routing.History.getInstance();
            sPreviousHash = oHistory.getPreviousHash();

            if (sPreviousHash !== undefined) {
                window.history.go(-1);
            } else {
                oRouter = this.getRouter();
                oRouter.navTo("Overview", true /*no history*/ );
            }
        }
    });
});
```

Figure 125: Implement the NotFound Controller

11. Implement the *controller* of the *Overview* view.

Overview.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller"
], function (Controller) {
    "use strict";

    return Controller.extend("student00.sap.training.dynamicpage.controller.Overview", {
        getRouter: function () {
            return sap.ui.core.UIComponent.getRouterFor(this);
        },

        onPress: function (oEvent) {
            var oItem, oCtx, sCarrid, oRouter;
            oItem = oEvent.getSource();
            oCtx = oItem.getBindingContext();
            sCarrid = oCtx.getProperty("Carrid");

            oRouter = this.getRouter();
            oRouter.navTo("Carrier", {
                carrierId: sCarrid
            }, false /*with history*/ );
        }
    });
});
```

Figure 126: Implement the Controller

12. Implement the routing definition inside the `manifest.json`. Follow the routing and target configuration.

*manifest.json

```
    "routing": {
        "config": {
            "routerClass": "sap.m.routing.Router",
            "viewType": "XML",
            "async": true,
            "viewPath": "student00.sap.training.dynamicpage.view",
            "controlAggregation": "pages",
            "controlId": "app",
            "clearControlAggregation": false
        },
```

Figure 127: Implement the Routing Definition

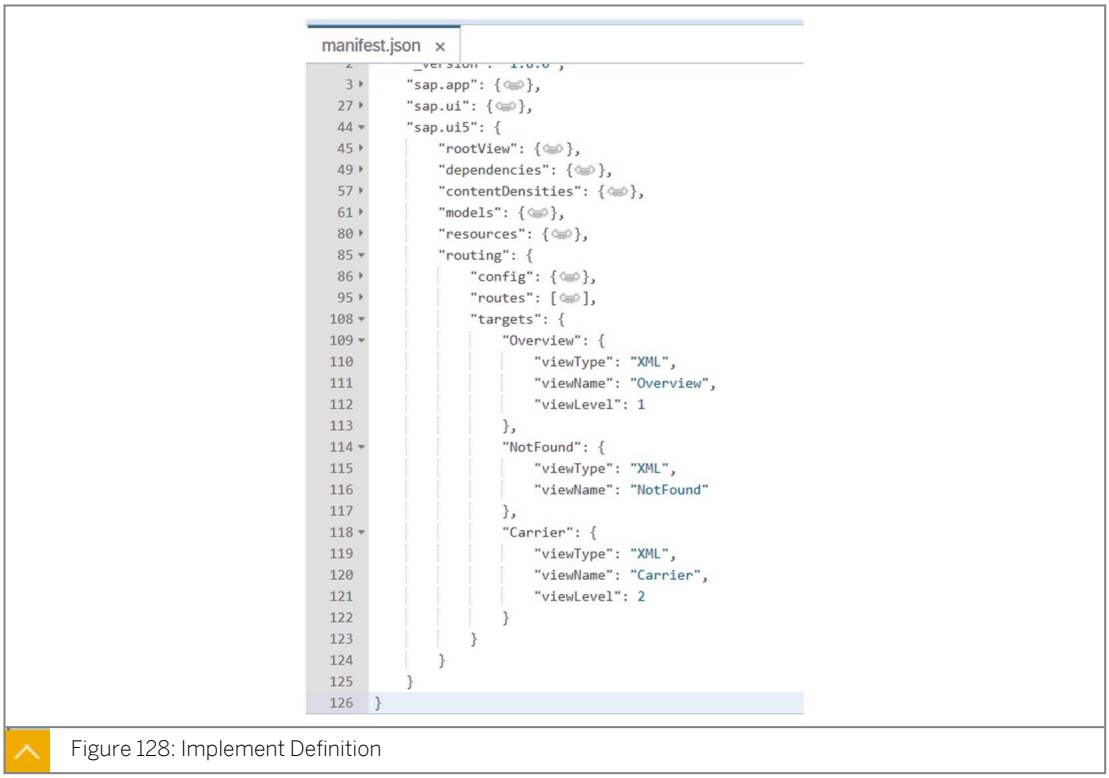


Figure 128: Implement Definition

a) Open the default route to `manifest.json` with Descriptor Editor and choose *Delete*.

b) Go to the target definitions and remove the target with the name *TargetApp*.

c) Select the item *Overview* from the list of available route names and press the plus button to create a default route for the target *Overview* and choose the option with the name *Overview* from the list of available targets.

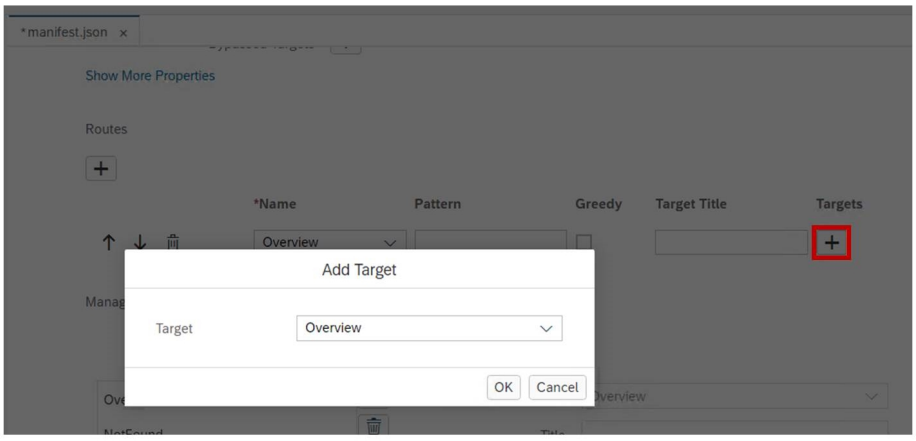


Figure 129: Target Overview

d) To create a new route for the pattern carriers {carrierId}, select the plus icon on the level of the new route and add carrier as the target reference and choose *OK*.

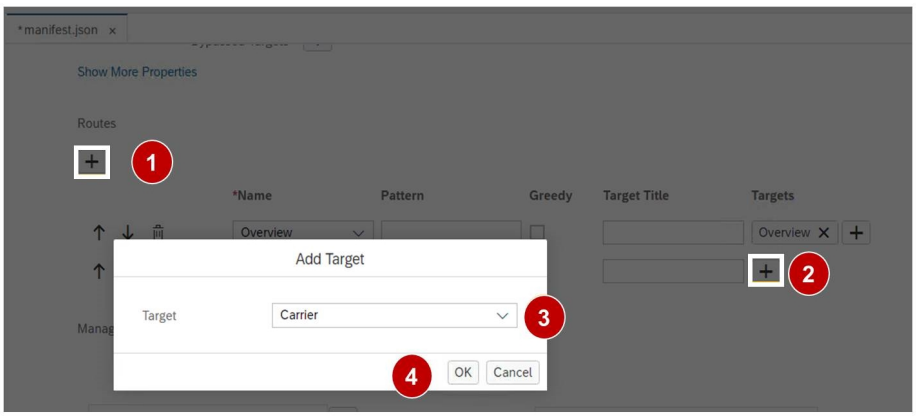


Figure 130: Add Target

e) Add the pattern for carriers or carrierId to the *pattern* field.

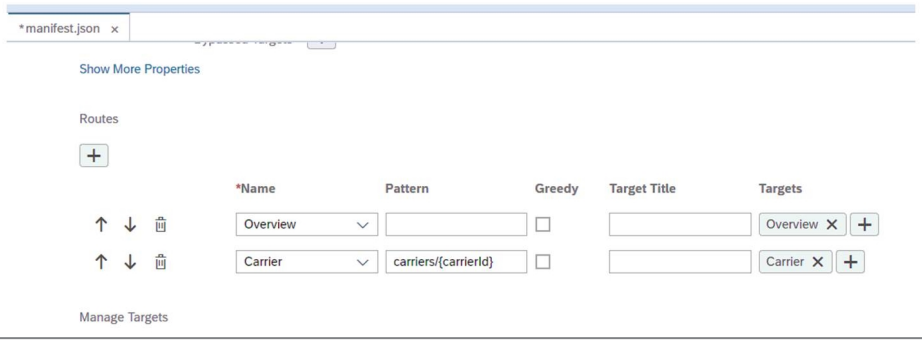


Figure 131: Add Pattern

f) Add the pattern carriers or carrierId.

g) Set the property *viewLevel* of target *Overview* to **`1`**.

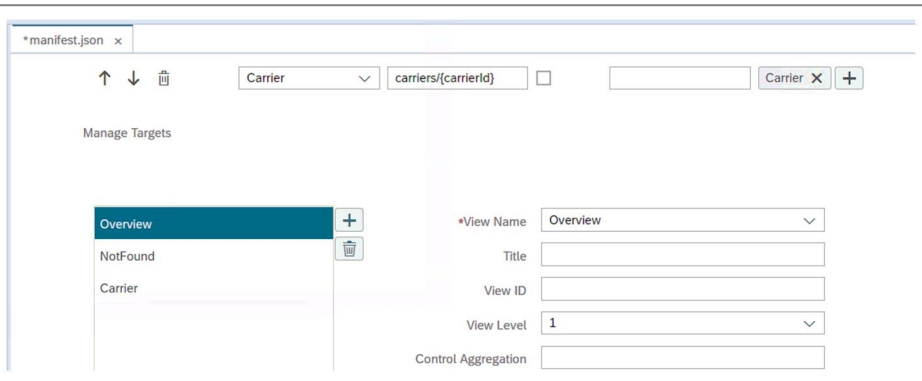


Figure 132: Set Property of Overview 1

h) Set the property *viewLevel* of target *Carrier* to **`2`**.

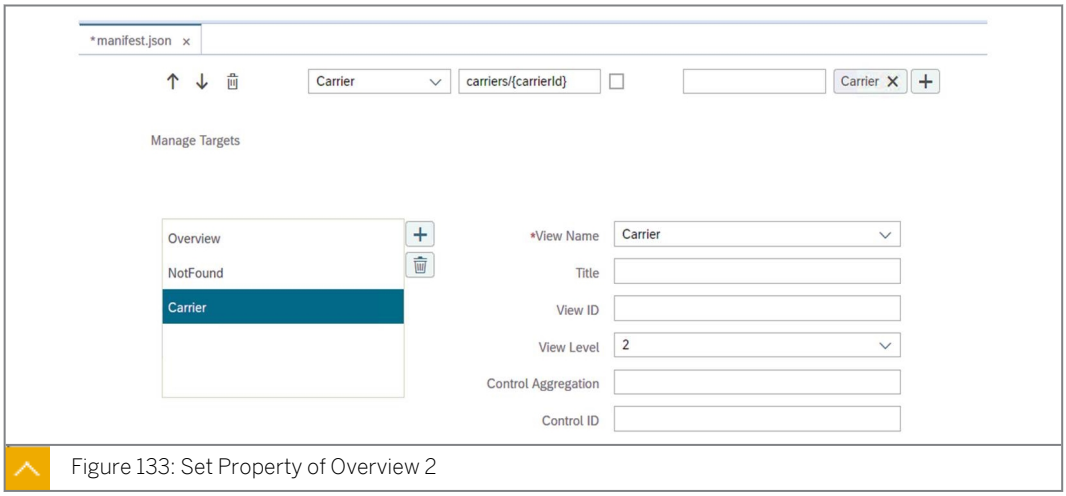


Figure 133: Set Property of Overview 2

i) Save your changes.

13. Test your application.

a) Select the file `index.html` in the webapp folder of your project and choose from the context menu *Run → Run as Web Application*.

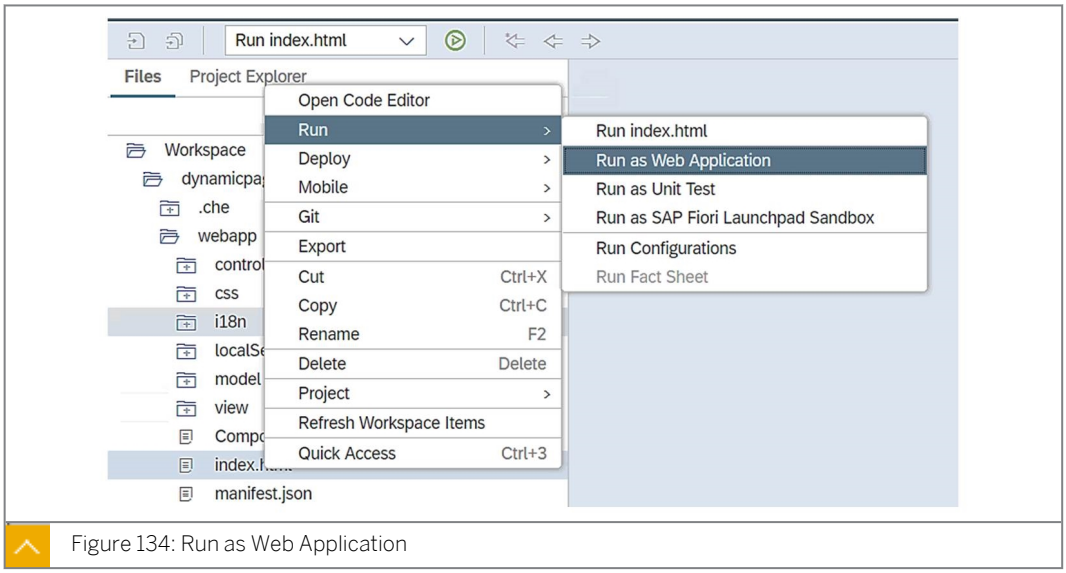


Figure 134: Run as Web Application

b) Close the information dialog if one displays.

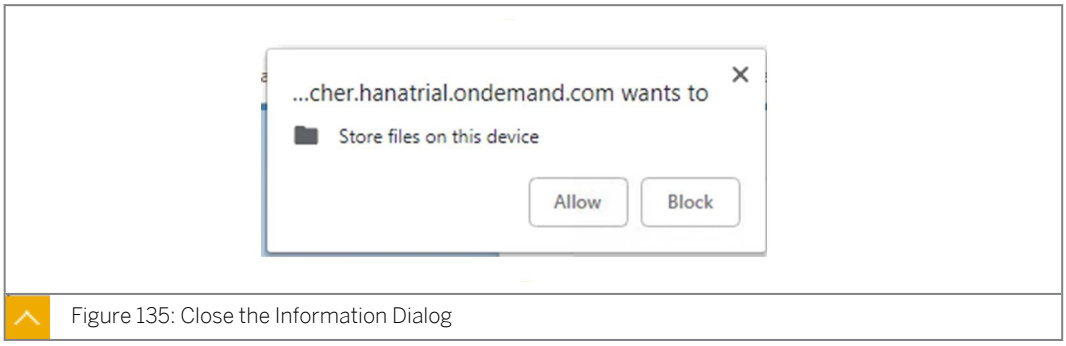


Figure 135: Close the Information Dialog

c) Sign in with your credentials for the front-end server if required.

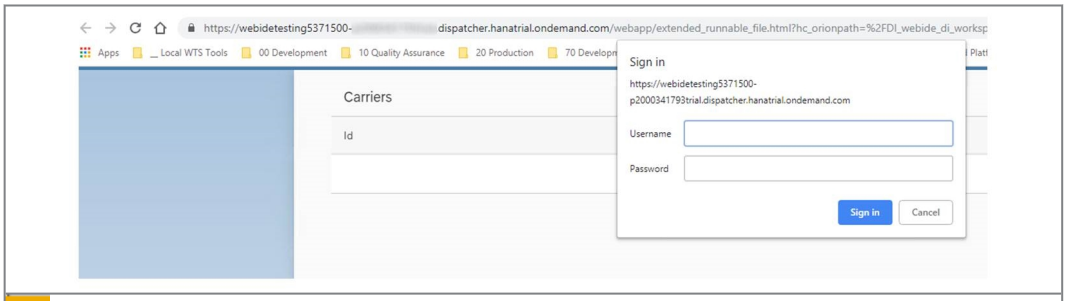


Figure 136: Sign In to Front-End Server

d) Choose a carrier from the list of carriers.

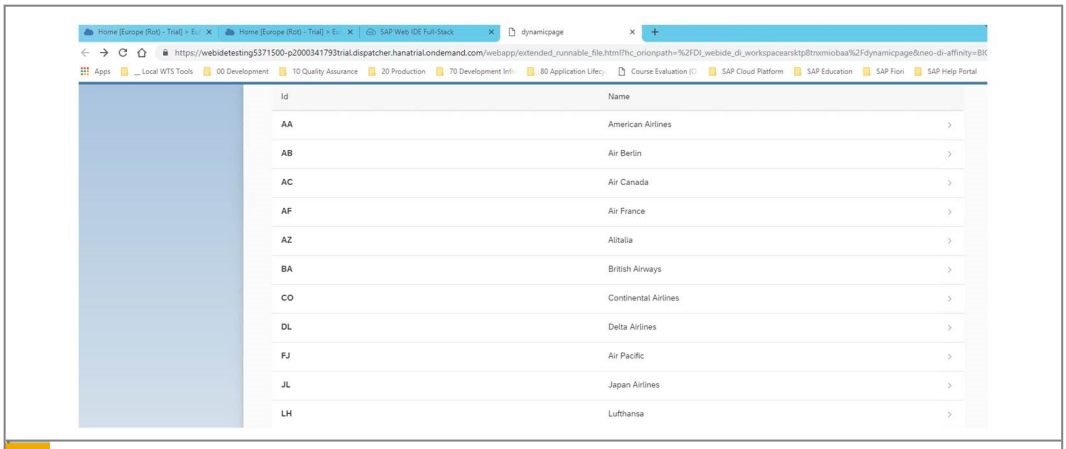


Figure 137: Choose a Carrier

e) The details of the selected carrier are displayed.

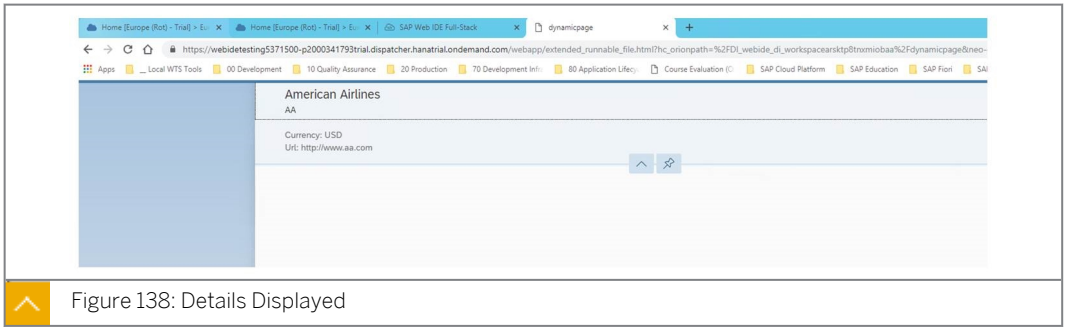


Figure 138: Details Displayed

Use the back button on your browser to navigate to the carrier-overview page. In a real world scenario the application is embedded in the SAP Fiori lauchpad. In the SAP Fiori lauchpad the user is able to use the *Back* button in the launchpad.

Unit 11
Exercise 10

Create Master-Detail using the Flexible Column Layout

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, step procedures and names such as *buttons* and *fields* may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

Business example

In this exercise you learn how to implement a master-detail application using the flexible column layout.

1. Log on to the SAP Web IDE Full-Stack and export the project *dynamicpage* from the previous exercise and import the project with the name *flexiblecolumn* into the workspace of your *SAP Web IDE Full-Stack*.
2. Import the content of the file `dynamicpage.zip` as a project with the name *flexiblecolumnlayout*.
3. Add a control of the type *sap.f.FlexibleColumnLayout* to the *App* control located in the `App.view.xml` file. Add an *XML Namespace with alias f* to your *View* control to reference the SAPUI5 namespace and assign the value *sap.f*. Add the listed attributes to the control.

   Table 39: sap.f.FlexibleColumnLayout Attributes/Values

Attribute	Value
id	idFlexLayout
layout	{appView>/layout}
backgroundDesign	Translucent

4. Add the following implementation of the `onInit` function to the controller of the app view.

App.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/ui/model/json/JSONModel"
], function (Controller,JSONModel) {
    "use strict";

    return Controller.extend("student00.sap.training.dynamicpage.controller.App", {
        onInit : function() {
            var oViewModel = new JSONModel({
                busy : true,
                delay : 0,
                layout : "OneColumn",
                previousLayout : "",
                actionButtonsInfo : {
                    midColumn : {
                        fullScreen : false
                    }
                }
            });
            this.getOwnerComponent().setModel(oViewModel, "appView");
        }
    });
});
```

Figure 140: Add onInit

5. Update your routing configuration inside the `manifest.json` file. Make sure the attribute *clearControlAggregation* is set to *true*. Change the following attributes at the config-object.

Table 40: Routing Configuration

Attribute	Value
controlAggregation	beginColumnPages
controlId	idFlexLayout
clearControlAggregation	true

6. Add the *Overview target* to the carrier route and add the attribute *controlAggregation* to the *NotFound* target. Assign the value **`midColumnPages`** to the controlAggregation attribute. Add the attribute *controlAggregation* to the carrier target and assign the value **`midColumnPages`**.

7. Add and implement a file with the name ListSelector.js to the controller folder.

Note:
Ensure you are using the correct namespace related to your implementation and group.

ListSelector.js

```
sap.ui.define([
    "sap/ui/base/Object"
], function (Object) {
    "use strict";
    return Object.extend("student00.sap.training.dynamicpage.controller.ListSelector", {

        constructor : function () {
            this._oWhenListHasBeenSet = new Promise(function (fnResolveListHasBeenSet) {
                this._fnResolveListHasBeenSet = fnResolveListHasBeenSet;
            }.bind(this));
            this.oWhenListLoadingIsDone = new Promise(function (fnResolve, fnReject) {}.bind(this));
        },
```

Figure 147: Implement ListSelectore.js

```
        setBoundMasterList : function (oList) {
            this._oList = oList;
            this._fnResolveListHasBeenSet(oList);
        },
```

Figure 148: Implement ListSelectore.js

```
        selectAListItem : function (sBindingPath) {
            this.oWhenListLoadingIsDone.then(
                function () {
                    var oList = this._oList,oSelectedItem;
                    if (oList.getMode() === "None") {
                        return;
                    }
                    oSelectedItem = oList.getSelectedItem();
                    if (oSelectedItem && oSelectedItem.getBindingContext().getPath() === sBindingPath) {
                        return;
                    }
                    oList.getItems().some(function (oItem) {
                        if (oItem.getBindingContext() && oItem.getBindingContext().getPath() === sBindingPath) {
                            oList.setSelectedItem(oItem);
                            return true;
                        }
                    });
                }.bind(this),
                function () {
                    jQuery.sap.log.warning("Could not select the list item with the path" + sBindingPath +
                        " because the list encountered an error or had no items");
                }
            );},
```

Figure 149: Implement ListSelectore.js

```
        clearMasterListSelection : function () {
            this._oWhenListHasBeenSet.then(function () {
                this._oList.removeSelections(true);
            }.bind(this));
        }
    });
});
```

Figure 150: Implement ListSelectore.js

8. Add a reference to the ListSelector.js implementation in your `Component.js` file and assign an instance of type *ListSelector* to a member variable with the name *oListSelector*.

9. Use the *sap.f.semantic.SemanticPage* in the `Carrier.view.xml` instead of the *sap.f.DynamicPage* control. Show the same data as before and leave the overall layout the same. Replace *SAPUI5-namespace sap.f.semantic* with *XML-namespace alias semantic*. Use the *titlHeading* aggregation to display the title *sap.m.Title* control and the *headerContent* aggregation to display the *FlexBox* implementation.

10. Add an *sap.f.semantic.CloseAction* to the *SemanticPage* implementation of `Carrier.view.xml`. Use the following attribute and values.

Table 41: sap.f.semantic.CloseAction Attributes and Values

Attributes	Value
id	idCloseAction
press	onCloseDetailPress

11. Add an sap.f.semantic.FullScreenAction to the SemanticPage implementation of `Carrier.view.xml`. Use the following attributes and values:

Table 42: sap.f.semantic.FullScreenAction Attributes and Values

Attribute	Value
id	idEnterFullScreen
visible	{= !${device>/system/phone} && !${appView>/actionButtonsInfo/midColumn/fullScreen}}
press	onToggleFullScreen

12. Add `sap.f.semantic.ExitFullScreenAction` to the *SemanticPage* implementation of `Carrier.view.xml`.

Table 43: sap.f.semantic.ExitFullScreenAction Attributes and Values

Attribute	Value
id	idExitFullScreen
visible	{= !${device>/system/phone} && ${appView>/actionButtonsInfo/midColumn/fullScreen}}
press	onToggleFullScreen

13. Add the listed attributes to the *sap.m.Table* control in the file `Overview.view.xml`.

Table 44: sap.m.Table Action Attributes and Values

Attribute	Value
mode	{= ${device>/system/phone} ? 'None' : 'SingleSelectMaster'}
selectionChange	onSelectionChange

14. Change the implementation of the Overview controller.
15. Implement the function `_showCarrierDetails`.

```
    _showCarrierDetails : function(sCarrid) {
        var oRouter = this.getRouter();
        oRouter.navTo("Carrier", {
            carrierId: sCarrid
        }, false /*with history*/ );
    }
```

Figure 166: Implement showCarrierDetails

16. Change the implementation of the Carrier controller. Add the following code to the Carrier controller implementation.

Carrier.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller"
], function (Controller) {
    "use strict";

    return Controller.extend("student00.sap.training.dynamicpage.controller.Carrier", {
        onInit: function () {
            var oRouter = this.getRouter();
            oRouter.getRoute("Carrier").attachMatched(this._onObjectMatched, this);
        },

        getRouter: function () {
            return sap.ui.core.UIComponent.getRouterFor(this);
        },
```

Figure 167: Implementation

```
        _onObjectMatched: function (oEvent) {
            var oArgs, oView;
            oArgs = oEvent.getParameter("arguments");
            this._sCarrierId = oArgs.carrierId;
            oView = this.getView();

            oView.bindElement({
                path: "/ZBC_C_Carrier_TP('" + this._sCarrierId + "')",
                events: {
                    change: this._onBindingChange.bind(this),
                    dataRequested: function () {
                        oView.setBusy(true);
                    },
                    dataReceived: function () {
                        oView.setBusy(false);
                    }
                }
            });
            this.getOwnerComponent().oListSelector.selectAListItem("/ZBC_C_Carrier_TP('" + this._sCarrierId + "')");
            var oViewModel = this.getOwnerComponent().getModel("appView");
            oViewModel.setProperty("/layout","TwoColumnsMidExpanded");
            this.getOwnerComponent().setModel(oViewModel,"appView");
        },
```

Figure 168: Implementation

```
        _onBindingChange: function () {
            var oView = this.getView();
            var oElementBinding = oView.getElementBinding();
            // No data for the binding
            if (oElementBinding && !oElementBinding.getBoundContext()) {
                this.getOwnerComponent().oListSelector.clearMasterListSelection();
                this.getRouter().getTargets().display("NotFound");
                return;
            }
            var sPath = oElementBinding.getPath();
            this.getOwnerComponent().oListSelector.selectAListItem(sPath);
        },
```

Figure 169: Implementation

```
53      onCloseDetailPress: function () {
54          this.getOwnerComponent().getModel("appView").setProperty(
55                                  "/actionButtonsInfo/midColumn/fullScreen", false);
56          this.getOwnerComponent().oListSelector.clearMasterListSelection();
57          this.getRouter().navTo("Overview");
58      },
59
60      onToggleFullScreen: function () {
61          var bFullScreen = this.getOwnerComponent().getModel("appView").getProperty(
62              "/actionButtonsInfo/midColumn/fullScreen");
63          this.getOwnerComponent().getModel("appView").setProperty(
64              "/actionButtonsInfo/midColumn/fullScreen", !bFullScreen);
65          if (!bFullScreen) {
66              // store current layout and go full screen
67              this.getOwnerComponent().getModel("appView").setProperty("/previousLayout",
68                      this.getOwnerComponent().getModel("appView").getProperty("/layout"));
69              this.getOwnerComponent().getModel("appView").setProperty("/layout", "MidColumnFullScreen");
70          } else {
71              // reset to previous layout
72              this.getOwnerComponent().getModel("appView").setProperty("/layout",
73                      this.getOwnerComponent().getModel("appView").getProperty("/previousLayout"));
74          }
75      },
```

Figure 170: Implementation

```
77      onNavBack: function () {
78          var oHistory, sPreviousHash, oRouter;
79
80          oHistory = sap.ui.core.routing.History.getInstance();
81          sPreviousHash = oHistory.getPreviousHash();
82
83          if (sPreviousHash !== undefined) {
84              window.history.go(-1);
85          } else {
86              oRouter = this.getRouter();
87              oRouter.navTo("Overview", true /*no history*/ );
88          }
89      }
90
91  });
```

Figure 171: Implementation

17. Test your application.

Unit 11
Solution 10

Create Master-Detail using the Flexible Column Layout

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, step procedures and names such as *buttons* and *fields* may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

Business example

In this exercise you learn how to implement a master-detail application using the flexible column layout.

1. Log on to the SAP Web IDE Full-Stack and export the project *dynamicpage* from the previous exercise and import the project with the name *flexiblecolumn* into the workspace of your *SAP Web IDE Full-Stack*.
   a) Choose the project *dynamicpage* from the workspace and select *Export* from the context menu.
   b) Download and open the `dynamicpage.zip` file.
2. Import the content of the file `dynamicpage.zip` as a project with the name *flexiblecolumnlayout*.
   a) Select the workspace folder and from the context menu, choose *Import → File or Project*.
   b) Browse for the exported file and change the file name in the Import field to **`/flexiblecolumnlayout`** and confirm with *OK*.
   c) On successful import you see the newly imported project with the given name.
3. Add a control of the type *sap.f.FlexibleColumnLayout* to the *App* control located in the `App.view.xml` file. Add an *XML Namespace with alias f* to your *View* control to reference the SAPUI5 namespace and assign the value *sap.f*. Add the listed attributes to the control.

   Table 39: sap.f.FlexibleColumnLayout Attributes/Values

Attribute	Value
id	idFlexLayout
layout	{appView>/layout}

Attribute	Value
backgroundDesign	Translucent

a) Open the file `App.view.xml` located in the view folder of your project.

b) Add the *XML namespace alias f* to the view control and assign *SAPUI5 namespace sap.f*.

c) Add a control of type *sap.f.FlexbileColumnLayout* to the *App* control and add the attributes and values listed in the table to the control.

*App.view.xml ×

```
<mvc:View controllerName="student00.sap.training.dynamicpage.controller.App" xmlns:mvc="sap.ui.core.mvc"
    displayBlock="true" xmlns="sap.m" xmlns:f="sap.f">
    <Shell id="shell">
        <App id="app">
            <f:FlexibleColumnLayout id="idFlexLayout" layout="{appView>/layout}" backgroundDesign="Translucent"/>
        </App>
    </Shell>
</mvc:View>
```

Figure 139: Add a Control

d) Save your changes.

4. Add the following implementation of the `onInit` function to the controller of the app view.

App.controller.js ×

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/ui/model/json/JSONModel"
], function (Controller,JSONModel) {
    "use strict";

    return Controller.extend("student00.sap.training.dynamicpage.controller.App", {
        onInit : function() {
            var oViewModel = new JSONModel({
                busy : true,
                delay : 0,
                layout : "OneColumn",
                previousLayout : "",
                actionButtonsInfo : {
                    midColumn : {
                        fullScreen : false
                    }
                }
            });
            this.getOwnerComponent().setModel(oViewModel, "appView");
        }
    });
});
```

Figure 140: Add onInit

a) Open the file `App.controller.js`. The file is in the controller folder of your project.

b) Add a reference to the `sap.ui.model.json.JSONModel` implementation.

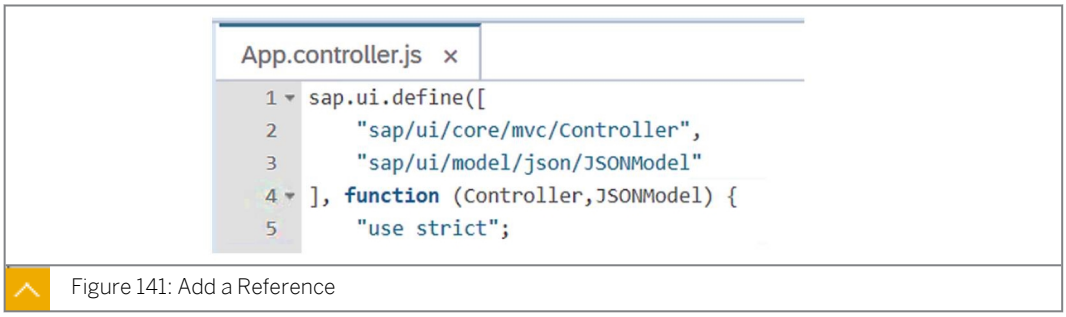


Figure 141: Add a Reference

c) Implement the `onInit` function.

5. Update your routing configuration inside the `manifest.json` file. Make sure the attribute *clearControlAggregation* is set to *true*. Change the following attributes at the config-object.

Table 40: Routing Configuration

Attribute	Value
controlAggregation	beginColumnPages
controlId	idFlexLayout
clearControlAggregation	true

a) Open the `manifest.json` file located directly under the webapp folder.

b) Switch to the Routing tab of the Descriptor Editor.

c) Select the link *Show More Properties*.

d) Change the attributes to the new attributes.

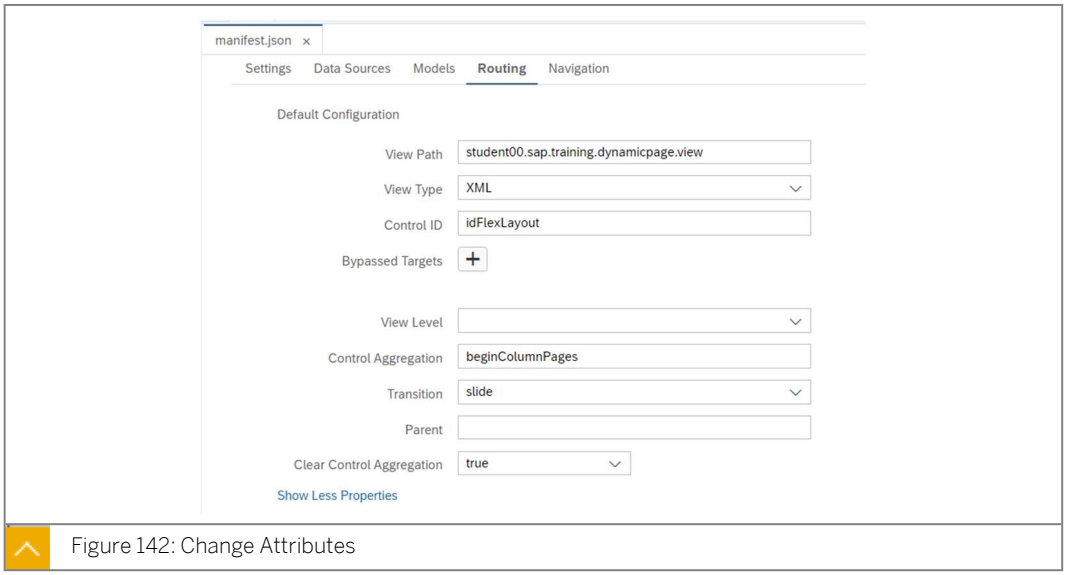


Figure 142: Change Attributes

6. Add the *Overview target* to the carrier route and add the attribute *controlAggregation* to the *NotFound* target. Assign the value **`midColumnPages`** to the controlAggregation attribute. Add the attribute *controlAggregation* to the carrier target and assign the value **`midColumnPages`**.

a) Select the plus symbol at the level of the carrier route to add the overview target.

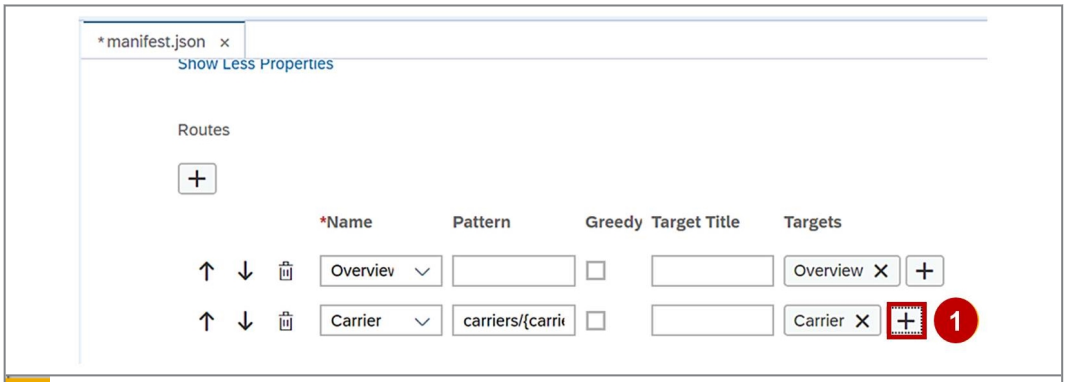


Figure 143: Select + symbol

b) Choose the entry *Overview* from the list of available targets and confirm with *OK*.

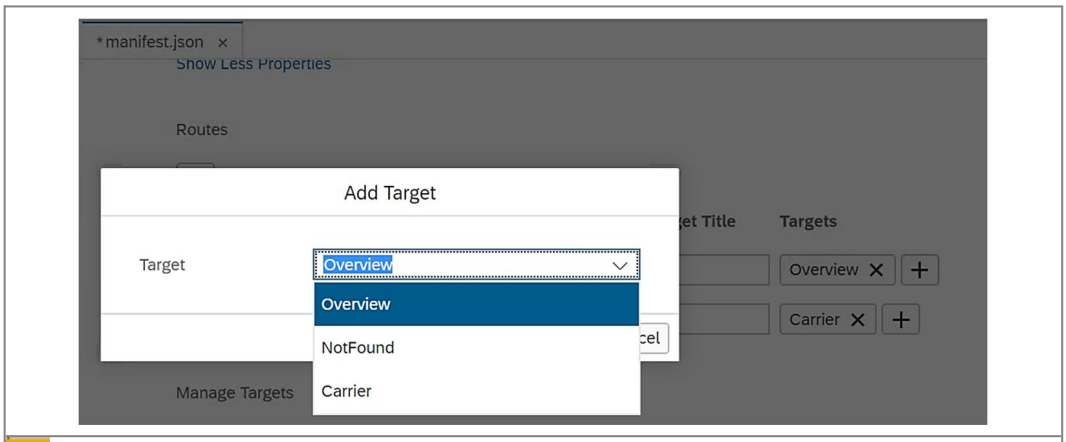


Figure 144: Choose Overview

c) Select the *NotFound* target and add the value **`midColumnPages`** to the *ControlAggregation* attribute.

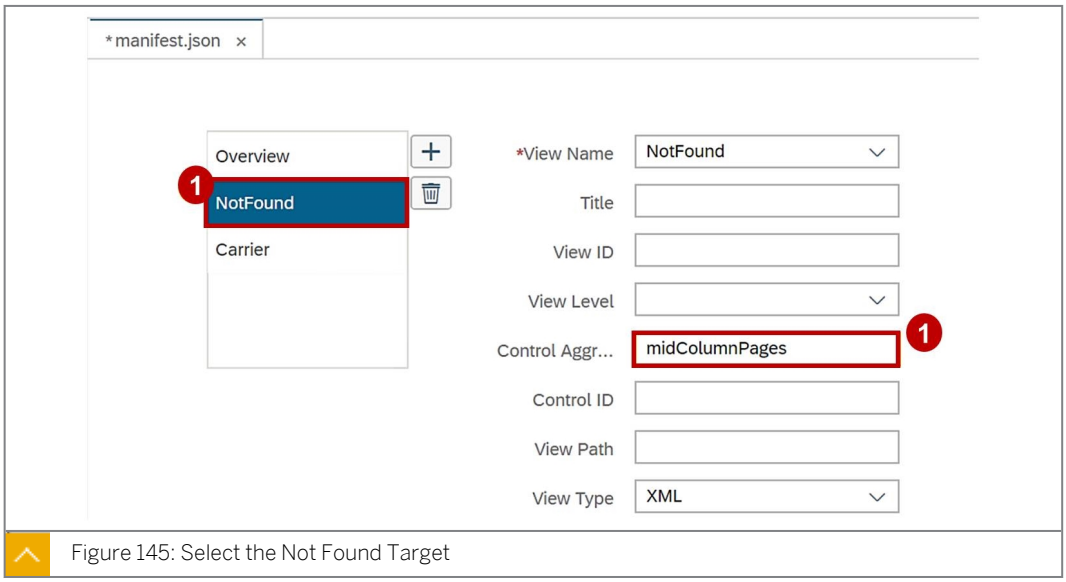


Figure 145: Select the Not Found Target

d) Select the carrier target and add the value *midColumnPages* to the *ControlAggregation* attribute.

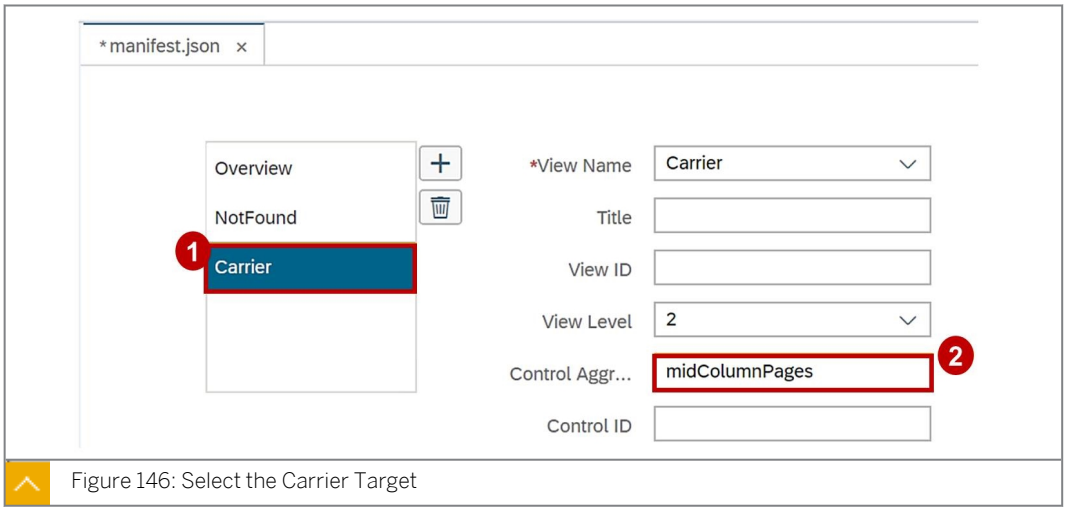


Figure 146: Select the Carrier Target

e) Save your changes.

7. Add and implement a file with the name ListSelector.js to the controller folder.

Note:
Ensure you are using the correct namespace related to your implementation and group.

Figure 147: Implement ListSelectore.js

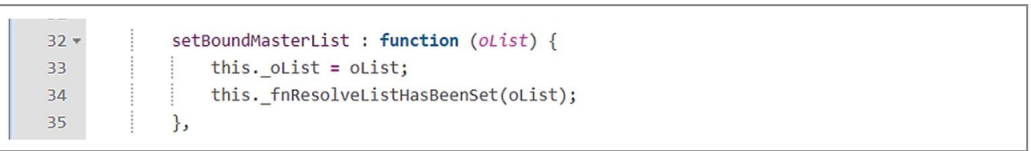


Figure 148: Implement ListSelectore.js

```
37      selectAListItem : function (sBindingPath) {
38          this.oWhenListLoadingIsDone.then(
39              function () {
40                  var oList = this._oList,oSelectedItem;
41                  if (oList.getMode() === "None") {
42                      return;
43                  }
44                  oSelectedItem = oList.getSelectedItem();
45                  if (oSelectedItem && oSelectedItem.getBindingContext().getPath() === sBindingPath) {
46                      return;
47                  }
48                  oList.getItems().some(function (oItem) {
49                      if (oItem.getBindingContext() && oItem.getBindingContext().getPath() === sBindingPath) {
50                          oList.setSelectedItem(oItem);
51                          return true;
52                      }
53                  });
54              }.bind(this),
55              function () {
56                  jQuery.sap.log.warning("Could not select the list item with the path" + sBindingPath +
57                      " because the list encountered an error or had no items");
58              }
59          );},
```

Figure 149: Implement ListSelectore.js

```
61      clearMasterListSelection : function () {
62          this._oWhenListHasBeenSet.then(function () {
63              this._oList.removeSelections(true);
64          }.bind(this));
65      }
66  });
67  });
```

Figure 150: Implement ListSelectore.js

a) From the context menu, choose *New* → *File* and then select the controller folder of your project.

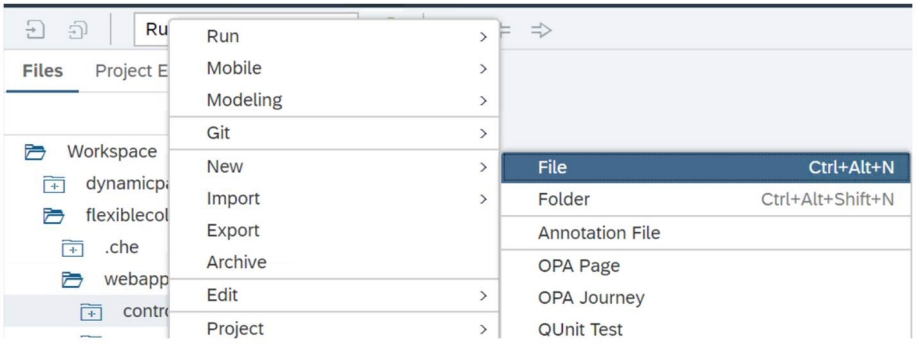


Figure 151: Select Controller Folder

b) Add the value **`ListSelector.js`** in the file name field and confirm *OK*.

Figure 152: Enter Values

c) Implement the file and save your changes.

8. Add a reference to the ListSelector.js implementation in your `Component.js` file and assign an instance of type *ListSelector* to a member variable with the name *oListSelector*.
   a) Open the file `Component.js` located in the webapp folder of your project.
   b) Add a reference to the *ListSelector* implementation.

Component.js ×

```
1  sap.ui.define([
2      "sap/ui/core/UIComponent",
3      "sap/ui/Device",
4      "student00/sap/training/dynamicpage/model/models",
5      "student00/sap/training/dynamicpage/controller/ListSelector"
6  ], function (UIComponent, Device, models, ListSelector) {
7      "use strict";
```

Figure 153: Add a Reference

   c) Declare a member variable with the name *oListSelector* and assign a new instance of *ListSelector*.

*Component.js ×

```
1  sap.ui.define([
2      "sap/ui/core/UIComponent",
3      "sap/ui/Device",
4      "student00/sap/training/dynamicpage/model/models",
5      "student00/sap/training/dynamicpage/controller/ListSelector"
6  ], function (UIComponent, Device, models, ListSelector) {
7      "use strict";
8
9      return UIComponent.extend("student00.sap.training.dynamicpage.Component", {
10
11         metadata: {},
14
15         /**/
20         init: function () {
21             this.oListSelector = new ListSelector();
22             // call the base component's init function
23             UIComponent.prototype.init.apply(this, arguments);
24             // enable routing
```

Figure 154: Declare a Variable

   d) Save your changes.

9. Use the *sap.f.semantic.SemanticPage* in the `Carrier.view.xml` instead of the *sap.f.DynamicPage* control. Show the same data as before and leave the overall layout the same. Replace *SAPUI5-namespace sap.f.semantic* with *XML-namespace alias semantic*. Use the *titlHeading* aggregation to display the title *sap.m.Title* control and the *headerContent* aggregation to display the *FlexBox* implementation.
   a) Open the file `Carrier.view.xml` located in the view folder of your project.
   b) Replace *SAPUI5-namespace sap.f.semantic* with *XML-namespace alias semantic*.

Carrier.view.xml

```
<mvc:View xmlns:core="sap.ui.core" xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m"
    xmlns:f="sap.f" xmlns:layout="sap.ui.layout" xmlns:semantic="sap.f.semantic"
    controllerName="student00.sap.training.dynamicpage.controller.Carrier"
    xmlns:html="http://www.w3.org/1999/xhtml">
    <f:DynamicPage id="dynamicPageId" headerExpanded="true" toggleHeaderOnTitleClick="true">
        <f:title>
            <f:DynamicPageTitle>
```

Figure 155: Replace Namespace

c) Implement the changes.

Carrier.view.xml

```
<mvc:View xmlns:core="sap.ui.core" xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m"
    xmlns:f="sap.f" xmlns:layout="sap.ui.layout" xmlns:semantic="sap.f.semantic"
    controllerName="student00.sap.training.dynamicpage.controller.Carrier"
    xmlns:html="http://www.w3.org/1999/xhtml">
    <semantic:SemanticPage id="dynamicPageId" headerExpanded="true" toggleHeaderOnTitleClick="true">
        <semantic:titleHeading>
            <Title text="{Carrname}"/>
        </semantic:titleHeading>
        <semantic:headerContent>
            <FlexBox alignItems="Start" justifyContent="SpaceBetween">
                <items>
                    <layout:HorizontalLayout allowWrapping="true">
                        <layout:VerticalLayout class="sapUiMediumMarginEnd">
                            <ObjectAttribute title="{i18n>currLabelText}" text="{Currcode}"/>
                            <ObjectAttribute title="{i18n>urlLabelText}" text="{Url}"/>
                        </layout:VerticalLayout>
                    </layout:HorizontalLayout>
                </items>
            </FlexBox>
        </semantic:headerContent>
    </semantic:SemanticPage>
</mvc:View>
```

Figure 156: Implement the Changes

d) Save the changes.

10. Add an *sap.f.semantic.CloseAction* to the *SemanticPage* implementation of `Carrier.view.xml`. Use the following attribute and values.

Table 41: sap.f.semantic.CloseAction Attributes and Values

Attributes	Value
id	idCloseAction
press	onCloseDetailPress

a) Open the file `Carrier.view.xml`.

b) Add the *closeAction* aggregation to the *sap.f.semantic.SemanticPage* control.

c) Add an *sap.f.semantic.CloseAction* control with the listed attribute and values.

Figure 157: Add a Control

11. Add an sap.f.semantic.FullScreenAction to the SemanticPage implementation of `Carrier.view.xml`. Use the following attributes and values:

Table 42: sap.f.semantic.FullScreenAction Attributes and Values

Attribute	Value
id	idEnterFullScreen
visible	{= !${device>/system/phone} && !${appView>/actionButton-sInfo/midColumn/fullScreen}}
press	onToggleFullScreen

a) Add the `fullScreenAction` aggregation to the *SemanticPage* control.

b) Add a control of type *sap.f.semantic.FullScreenAction* and add the listed attribute and values.

Figure 158: Add a Control

12. Add `sap.f.semantic.ExitFullScreenAction` to the *SemanticPage* implementation of `Carrier.view.xml`.

Table 43: sap.f.semantic.ExitFullScreenAction Attributes and Values

Attribute	Value
id	idExitFullScreen

Attribute	Value
visible	{= !${device>/system/phone} && ${appView>/actionButton-sInfo/midColumn/fullScreen}}
press	onToggleFullScreen

a) Add the `exitFullScreenAction` aggregation to the *SemanticPage* control.

b) Add a control of type *sap.f.semantic.ExitFullScreenAction* to the aggregation and add the listed attribute and values to the control.

Carrier.view.xml

```
<mvc:View xmlns:core="sap.ui.core" xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m"
    xmlns:f="sap.f" xmlns:layout="sap.ui.layout" xmlns:semantic="sap.f.semantic"
    controllerName="student00.sap.training.dynamicpage.controller.Carrier"
    xmlns:html="http://www.w3.org/1999/xhtml">
    <semantic:SemanticPage id="dynamicPageId" headerExpanded="true" toggleHeaderOnTitleClick="true">
        <semantic:titleHeading>
            <Title text="{Carrname}"/>
        </semantic:titleHeading>
        <semantic:closeAction>
            <semantic:CloseAction
                id="idCloseAction"
                press="onCloseDetailPress"/>
        </semantic:closeAction>
        <semantic:fullScreenAction>
            <semantic:FullScreenAction
                id="idEnterFullScreen"
                visible="{= !${device>/system/phone} &amp;amp;&amp;amp; !${appView>/actionButtonsInfo/midColumn/fullScreen}}"
                press="onToggleFullScreen"/>
        </semantic:fullScreenAction>
        <semantic:exitFullScreenAction>
            <semantic:ExitFullScreenAction
                id="idExitFullScreen"
                visible="{= !${device>/system/phone} &amp;amp;&amp;amp; ${appView>/actionButtonsInfo/midColumn/fullScreen}}"
                press="onToggleFullScreen"/>
        </semantic:exitFullScreenAction>
```

Figure 159: Add a Control

c) Save your changes.

13. Add the listed attributes to the *sap.m.Table* control in the file `Overview.view.xml`.

Table 44: sap.m.Table Action Attributes and Values

Attribute	Value
mode	{= ${device>/system/phone} ? 'None' : 'SingleSelectMaster'}
selectionChange	onSelectionChange

a) Open the file `Overview.view.xml` located in the view folder of your project.

b) Add the listed attribute and value pairs to the Table control.

Overview.view.xml

```
<mvc:View xmlns:core="sap.ui.core" xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m" xmlns:f="sap.f"
    controllerName="student00.sap.training.dynamicpage.controller.Overview"
    xmlns:html="http://www.w3.org/1999/xhtml">
    <f:DynamicPage id="idDynamicPage" headerExpanded="true" toggleHeaderOnTitleClick="true">
        <f:content>
            <Table items="{/ZBC_C_Carrier_TP}"
                mode="{= ${device>/system/phone} ? 'None' : 'SingleSelectMaster'}"
                selectionChange="onSelectionChange">
                <headerToolbar>
```

Figure 160: Add Listed Attributes

c) Save your changes.

14. Change the implementation of the Overview controller.

a) Adapt the implementation of the `onInit` function.

Overview.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller"
], function (Controller) {
    "use strict";

    return Controller.extend("student00.sap.training.dynamicpage.controller.Overview", {
        onInit : function() {
            this._oList = this.byId("idCarrierList");
            this.getView().addEventDelegate({
                onBeforeFirstShow: function () {
                    this.getOwnerComponent().oListSelector.setBoundMasterList(this._oList);
                }.bind(this)
            });
            this.getView().addEventDelegate({
                onBeforeFirstShow: function () {
                    this.getOwnerComponent().oListSelector.setBoundMasterList(this._oList);
                }.bind(this)
            });

            this.getRouter().getRoute("Overview").attachPatternMatched(this._onMasterMatched, this);
            this.getRouter().attachBypassed(this.onBypassed, this);
        },
```

Figure 161: Adapt Implementation

b) Add a function with the name **`onBypassed`** and implement the function.

```
onBypassed : function () {
    this._oList.removeSelections(true);
},
```

Figure 162: Function onBypassed

c) Add a function with the name **`_onMasterMatched`** and implement the function.

```
_onMasterMatched :  function() {
    this.getOwnerComponent().getModel("appView").setProperty("/layout", "OneColumn");
},
```

Figure 163: Function onMasterMatched

d) Add a function with the name **`onSelectionChange`** and implement the function.

```
onSelectionChange : function(oEvent)  {
    var oItem, oCtx, sCarrid;
    oItem = oEvent.getParameter("listItem");
    oCtx = oItem.getBindingContext();
    sCarrid = oCtx.getProperty("Carrid");
    this._showCarrierDetails(sCarrid);
},
```

Figure 164: Function on SelectionChange

e) Adapt the implementation of the function `onPress`.

```
onPress: function (oEvent) {
    var oItem, oCtx, sCarrid;
    oItem = oEvent.getSource();
    oCtx = oItem.getBindingContext();
    sCarrid = oCtx.getProperty("Carrid");

    this._showCarrierDetails(sCarrid);
},
```

Figure 165: Function onPress

15. Implement the function `_showCarrierDetails`.

```
_showCarrierDetails : function(sCarrid) {
    var oRouter = this.getRouter();
    oRouter.navTo("Carrier", {
        carrierId: sCarrid
    }, false /*with history*/ );
}
```

Figure 166: Implement showCarrierDetails

a) Open the file `Overview.controller.js` located in the controller folder of your project and change the implementation as instructed.

b) Save your changes.

16. Change the implementation of the Carrier controller. Add the following code to the Carrier controller implementation.

Carrier.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller"
], function (Controller) {
    "use strict";

    return Controller.extend("student00.sap.training.dynamicpage.controller.Carrier", {
        onInit: function () {
            var oRouter = this.getRouter();
            oRouter.getRoute("Carrier").attachMatched(this._onObjectMatched, this);
        },

        getRouter: function () {
            return sap.ui.core.UIComponent.getRouterFor(this);
        },
```

Figure 167: Implementation

```
16  _onObjectMatched: function (oEvent) {
17      var oArgs, oView;
18      oArgs = oEvent.getParameter("arguments");
19      this._sCarrierId = oArgs.carrierId;
20      oView = this.getView();
21
22      oView.bindElement({
23          path: "/ZBC_C_Carrier_TP('" + this._sCarrierId + "')",
24          events: {
25              change: this._onBindingChange.bind(this),
26              dataRequested: function () {
27                  oView.setBusy(true);
28              },
29              dataReceived: function () {
30                  oView.setBusy(false);
31              }
32          }
33      });
34      this.getOwnerComponent().oListSelector.selectAListItem("/ZBC_C_Carrier_TP('" + this._sCarrierId + "')");
35      var oViewModel = this.getOwnerComponent().getModel("appView");
36      oViewModel.setProperty("/layout","TwoColumnsMidExpanded");
37      this.getOwnerComponent().setModel(oViewModel,"appView");
38  },
```

Figure 168: Implementation

```
40  _onBindingChange: function () {
41      var oView = this.getView();
42      var oElementBinding = oView.getElementBinding();
43      // No data for the binding
44      if (oElementBinding && !oElementBinding.getBoundContext()) {
45          this.getOwnerComponent().oListSelector.clearMasterListSelection();
46          this.getRouter().getTargets().display("NotFound");
47          return;
48      }
49      var sPath = oElementBinding.getPath();
50      this.getOwnerComponent().oListSelector.selectAListItem(sPath);
51  },
```

Figure 169: Implementation

```
53  onCloseDetailPress: function () {
54          this.getOwnerComponent().getModel("appView").setProperty(
55                  "/actionButtonsInfo/midColumn/fullScreen", false);
56          this.getOwnerComponent().oListSelector.clearMasterListSelection();
57          this.getRouter().navTo("Overview");
58  },
59
60  onToggleFullScreen: function () {
61          var bFullScreen = this.getOwnerComponent().getModel("appView").getProperty(
62              "/actionButtonsInfo/midColumn/fullScreen");
63          this.getOwnerComponent().getModel("appView").setProperty(
64              "/actionButtonsInfo/midColumn/fullScreen", !bFullScreen);
65          if (!bFullScreen) {
66              // store current layout and go full screen
67              this.getOwnerComponent().getModel("appView").setProperty("/previousLayout",
68                      this.getOwnerComponent().getModel("appView").getProperty("/layout"));
69              this.getOwnerComponent().getModel("appView").setProperty("/layout", "MidColumnFullScreen");
70          } else {
71              // reset to previous layout
72              this.getOwnerComponent().getModel("appView").setProperty("/layout",
73                      this.getOwnerComponent().getModel("appView").getProperty("/previousLayout"));
74          }
75      },
```

Figure 170: Implementation

```
77      onNavBack: function () {
78          var oHistory, sPreviousHash, oRouter;
79
80          oHistory = sap.ui.core.routing.History.getInstance();
81          sPreviousHash = oHistory.getPreviousHash();
82
83          if (sPreviousHash !== undefined) {
84              window.history.go(-1);
85          } else {
86              oRouter = this.getRouter();
87              oRouter.navTo("Overview", true /*no history*/ );
88          }
89      }
90
91  });
```

Figure 171: Implementation

a) Open the file `Carrier.controller.js` located in the controller folder of your project.

b) Implement your changes.

17. Test your application.

a) Select the `index.html` file located in the webapp folder of your project and choose from the context menu *Run → Run as Web Application*.

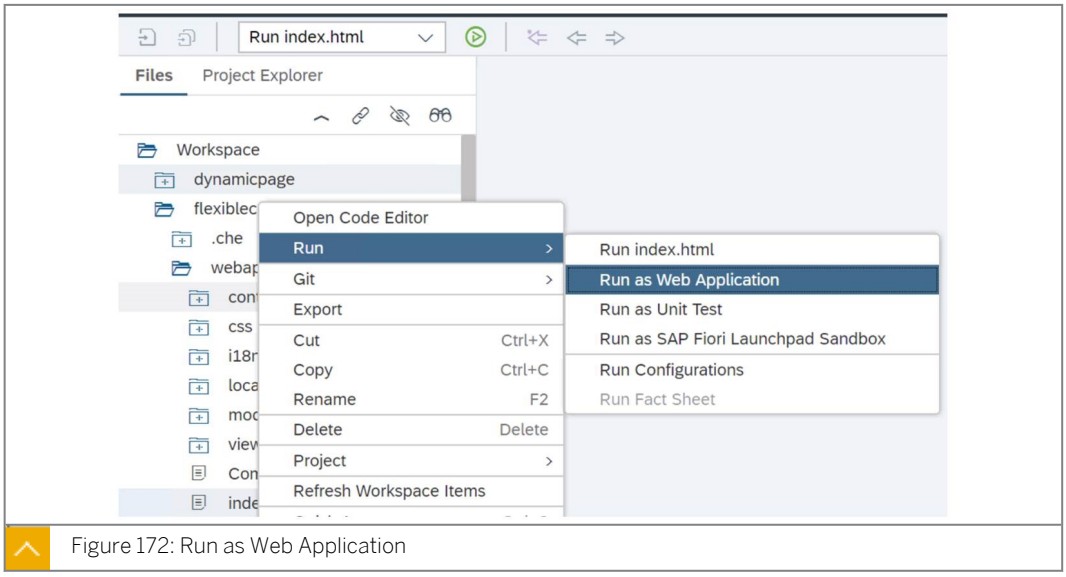


Figure 172: Run as Web Application

b) Log on with your credentials when required.

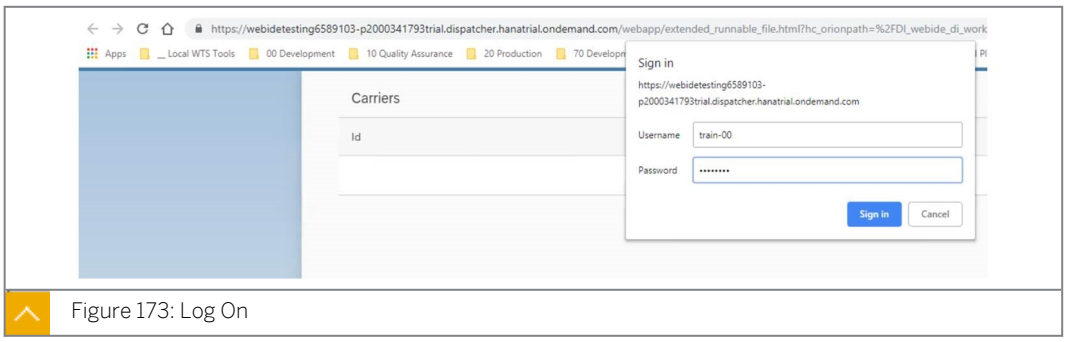


Figure 173: Log On

c) Test your application. Choose a carrier and display the details.

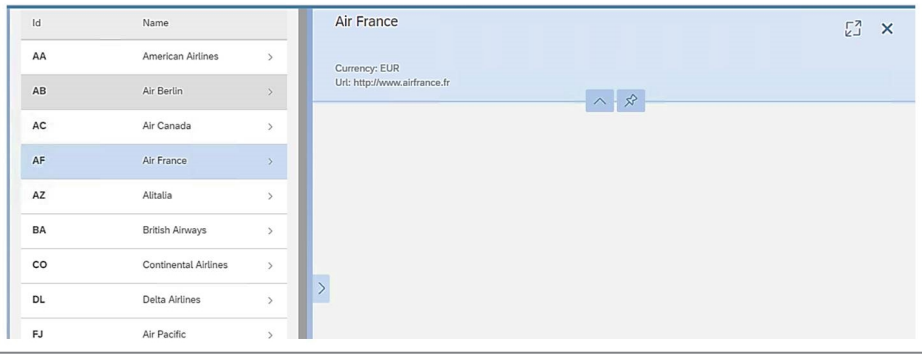


Figure 174: Display the Details

Unit 12
Exercise 11

Implement Value Helps

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, naming of fields and buttons as well as steps may differ from the exercise solution. The exercises show the application of group 00. Use your own student and group number for the exercises as assigned by your instructor.

Business Example

In the following exercise you will implement Value Help.

Task 1: Log on to the SAP Web IDE Full-Stack and create a SAPUI5 project

1. Log on to the *SAP Web IDE Full-Stack* and create an SAPUI5 project with the following settings:

   Table 45: Project Properties.

Parameter	Value
Project name	valuehelp
Namespace	student##.sap.training
View Type	XML
View Name	Main
SAPUI5 Version	1.60

Task 2: Implement value help

Implement the value help floorplan as described in the SAP Fiori Guidelines. This provides the user with a selection dialog to select a carrier. The application makes use of the OData service with the name UX_TRAVEL_SRV.

1. Add the UX_TRAVEL_SRV OData service to your project. The service is referenced by the default model of your application.
2. Replace the generated *sap.m.Page* control in the `Main.view.xml` with a *sap.f.DynamicPage* control. Assign the following attributes and values to the *sap.f.DynamicPage* control.

   Table 46: sap.f.DynamicPage Attributes and Values.

Attribute	Value
id	idDynPage
headerExpanded	true

Attribute	Value
toggleHeaderOnTitleClick	true
preserveHeaderStateOnScroll	true

3. Add an *sap.f.DynamicPageTitle* control to the title aggregation of the *sap.f.DynamicPage* control. Implement a *sap.m.Input* control with the following attributes and values inside the heading aggregation of the *sap.f.DynamicPageTitle* control. Add the aggregation `suggestionItems` to the newly added *sap.m.Input* control and add an *sap.ui.core.Item* control as template. Assign the binding for attribute `Carrid` to the text attribute of the *sap.ui.core.Item* control

Table 47: sap.m.Input control Attributes and Values

Attribute	Value
id	idCarInput
placeholder	Enter carrier id
showSuggestion	true
showValueHelp	true
suggestionItems	{/UX_C_Carrier_TP}
type	Text
valueHelpRequest	onValueHelpRequest
submit	onSubmit

4. Create an XML fragment with the name **`CarrierSelectionDialog`** inside the view folder of your project. Implement selection *Dialog*, using the *sap.m.SelectDialog* control with the listed attributes and values. Add a template of type *sap.m.StandardListItem* to the new *sap.m.SelectDialog* and use the listed attributes and values.

Table 48: sap.m.SelectDialog Attributes and Values

Attribute	Value
title	Carriers
class	sapUiPopupWithPadding
item	{/UX_C_Carrier_TP}
search	onValueHelpSearch
confirm	onValueHelpClose
cancel	onValueHelpClose

Table 49: sap.m.StandardListItem Attributes and Values

Attribute	Value
iconDensityAware	false

Attribute	Value
iconInsert	false
title	{Carrid}
description	{Carrname}

Task 3: Task Controller Implementation
Implement the event handler *onValueHelpRequest* inside the `Main.controller.js` file. The implementation obtains the value from the source object of the event and assigns the value to a local variable named *sInputValue*. The ID of the source object is stored inside a member variable with name *_sInputId*.

1. Create an instance of `CarrierSelectionDialog` and store the instance inside a member variable with the name `_oValueHelpDialog`. Get the binding of the items aggregation from the dialog and apply a filter for *Carrid* with *FilterOperator.Contains* and remember the AMD (Asynchronous module definition). Pass the *sInputValue* variable to the filter, then open the dialog.
2. Test the status of your implementation.
3. Implement the function `onValueHelpSearch` to handle events of `SelectDialog`. The `onValueHelpSearch` function is called when the user enters a value in the search field of the dialog and starts the search. The function needs to read the event parameter value and use this value for filtering. Make it possible for the user to search for the carrier name (property carrname, FilterOperator.StartsWith) and carrier ID (property Carrid, FilterOperator.Contains).
4. Implement the function `onValueHelpClose` to handle the event of the `SelectDialog`. The `onValueHelpClose` function is called when the dialog is closed. The event object of the event handler will get a parameter *selectedItem*. The parameter contains the item of the list selected by the user. Read the title property from the selected item and display the title in the input field of *Main.view.xml*. Recall that you stored the id of the input field in the member variable *_sInputId*. Reset the filters from the select dialog to complete the implementation. Now test your application.

Task 4: Show the details and the available flights of the selected carrier.
Add a fragment to the project to show the flights for a selected carrier.

1. Implement a new fragment with the name **`Flights.fragment.xml`** in the view folder of your project and add XML namespace aliases for `sap.m` as default and `sap.ui.core` with alias core to the `sap.ui.core.FragmentDefinition`.
2. Add an *sap.m.Table* control with the listed attributes and values to the newly-created fragment.

   Table 50: sap.m.Table control Attributes and Values.

Attribute	Value
id	idFlights
items	{ path: 'to_Flight', sorter: { path: 'Carrid' }}
mode	SingleSelectMaster

Attribute	Value
growingThreshold	true
growing	true
visible	true

Add an sap.m.Table control to your fragment.

Figure 195: Table Control

3. Add an *sap.m.Title* to the `headerToolbar` aggregation of the *sap.m.Table* with the listed attributes and values.

Table 51: sap.m.Title control Attributes and Values.

Attribute	Value
text	Flights
level	H2

Add the `headerToolbar` aggregation to the *sap.m.Table* control and add an *sap.m.Title* control with the listed attributes and values.

Figure 196: Table and Title Controls

4. Add columns to the table with the listed attributes and values.
5. Add one *sap.m.Text* control to each column using the listed attributes.

Table 57: sap.m.Text control Attributes and Values

Attribute	Value
text	Carrier
text	Flight no.
text	Date
text	Max seats

Attribute	Value
text	Occ. seats

Flights.fragment.xml

```
<core:FragmentDefinition xmlns="sap.m" xmlns:core="sap.ui.core">
    <Table id="idFlights" items="{ path: 'to_Flight', sorter: { path: 'Carrid' }}" mode="SingleSelectMaster" growing="true"
        growingThreshold="10" visible="true">
        <headerToolbar>
            <Toolbar>
                <Title text="Flights" level="H2"/>
            </Toolbar>
        </headerToolbar>
        <columns>
            <Column width="12em">
                <Text text="Carrier"/>
            </Column>
            <Column minScreenWidth="Tablet" demandPopin="true">
                <Text text="Flight no."/>
            </Column>
            <Column minScreenWidth="Tablet" demandPopin="true" hAlign="Right">
                <Text text="Date"/>
            </Column>
            <Column minScreenWidth="Tablet" demandPopin="true" hAlign="Center">
                <Text text="max. seats"/>
            </Column>
            <Column hAlign="Right">
                <Text text="occ. seats"/>
            </Column>
        </columns>
    </Table>
</core:FragmentDefinition>
```

Figure 198: Text Control

Add an *sap.m.Text* control to each of the columns and assign the attributes and values.

6. Add an `sap.m.ColumnListItem` to the items aggregation of the table and add UI controls to the cells aggregation of the `sap.m.ColumnListItem`.

Task 5: Extend the implementation of the Main.view.xml so it displays the details of the selected carrier

1. Add an *sap.f.DynamicPageHeader* to the header aggregation of the `DynamicPage` control with the listed attributes and values.

   Table 58: sap.f.DynamicPageHeader Attributes and Values.

Attribute	Value
pinnable	false

2. Add an *sap.m.FlexBox* control to the content aggregation with the listed attributes and values.

   Table 59: sap.m.FlexBox control Attributes and Values.

Attribute	Value
alignItems	Start
justifyContent	SpaceBetween
id	idHeaderLayout
visible	false

3. Add an *sap.m.Panel* control to the items aggregation of the *sap.m.FlexBox* control and add the listed attributes.

   Table 60: sap.m.Panel control Attributes and Values

Attribute	
backgroundDesign	Transparent
class	sapUiNoContentPadding sapUiSmallMar-ginTop

4. Add a *sap.ui.layout. HorizontalLayout* with the following attribute and value to the content aggregation of the `sap.m.Panel` control. Use layout as an XML namespace alias.

   Table 61: sap.ui.layout. HorizontalLayout Attribute and Value

Attribute	Value
allowWrapping	true

5. Add an *sap.ui.layout.VerticalLayout* with the listed attributes and values to the content aggregation of the *sap.ui.layout.HorizontalLayout* and add the following UI controls. Add an *sap.m.Text* control with attribute text and assign the value **`Airline`** to the control. Add an *sap.m.ObjectAttribute* control with the attributes title and text. Assign the value **`Carrier`** to the title attribute and bind the text attribute to the property *Carrname* of the entity. Add an *sap.m.ObjectAttribute* control with the attributes title and text. Assign the value **`Carrier id`** to the title attribute and bind the text attribute to the property *Carrid* of the entity.

6. Add an *sap.ui.layout.VerticalLayout* with the following attributes and values to the content aggregation of the *sap.ui.layout.HorizontalLayout* and add the following UI controls. Add the attribute class to the `VerticalLayout` control and assign the value **`sapUiMediumMarginBegin`** to the attribute. Add an *sap.m.Text* control with attribute text and assign the value **`Airline`** details to the control. Add an *sap.m.ObjectAttribute* control with the attributes title and text. Assign the value **`Currency`** to the title attribute and bind the text attribute to the property *Currcode* of the entity. Add an *sap.m.ObjectAttribute* control with the attributes title and text. Assign the value **`URL`** to the title attribute and bind the text attribute to the property *URL* of the entity.

7. Add an *sap.m.MessagePage* control to the content aggregation of the *sap.f.DynamicPage* control with the listed attributes and values.

   Table 62: sap.m.MessagePage control Attributes and Values.

	Value
showHeader	false
icon	sap-icon://search

Task 6: Implement controller logic to display the carrier details when the user selects a carrier from the list of carriers

1. Open the file `Main.controller.js` to declare a member variable *_oFormFragments* and add to the implementation by assigning a literal JavaScript object to the variable. This

variable acts as a map for fragments. Fragment instances can be stored by name in the map.

Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/ui/model/Filter",
    "sap/ui/model/FilterOperator"
], function (Controller, Filter, FilterOperator) {
    "use strict";

    return Controller.extend("student00.sap.training.valuehelp.controller.Main", {
        _oFormFragments: {},
        onValueHelpRequest: function (oEvent) {},

        onValueHelpSearch: function (oEvent) {},

        onValueHelpClose: function (oEvent) {}
    });
});
```

Figure 202: Member Variable

2. Add and implement the `onInit` function of the controller from the *Main* view. Obtain the UI control *idHeaderLayout* reference and assign the reference to a member variable of the controller. Name the variable *_oDynamicPage* and set the visibility of the layout to *false*.

Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/ui/model/Filter",
    "sap/ui/model/FilterOperator"
], function (Controller, Filter, FilterOperator) {
    "use strict";

    return Controller.extend("student00.sap.training.valuehelp.controller.Main", {
        _oFormFragments: {},

        onInit: function () {
            this.oDynamicPage = this.byId("idHeaderLayout");
            this.oDynamicPage.setVisible(false);
        },

        onValueHelpRequest: function (oEvent) {},

        onValueHelpSearch: function (oEvent) {},

        onValueHelpClose: function (oEvent) {}
    });
});
```

Figure 203: Function

3. Implement the function named `_createFlightsTable`. The function creates an instance of the flights fragment. The function checks if an instance of the fragment was already created. When the instance of the fragment map exists it should be returned. If not the instance must be created. Assign the instance to the map and it should return to the caller. After instantiation of the fragment, obtain a reference to the UI control with ID *idDynPage*. Call on the obtained reference `destroyContent` function and add the new fragment object by calling the `setContent` function on the dynamic page object.

Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/ui/model/Filter",
    "sap/ui/model/FilterOperator"
], function (Controller, Filter, FilterOperator) {
    "use strict";

    return Controller.extend("student00.sap.training.valuehelp.controller.Main", {
        _oFormFragments: {},

        onInit: function () {},

        onValueHelpRequest: function (oEvent) {},

        onValueHelpSearch: function (oEvent) {},

        onValueHelpClose: function (oEvent) {},

        _createFlightsTable : function() {
            var oFragment = this._oFormFragments["Flights"];
            if (oFragment) {
                return oFragment;
            }
            oFragment = sap.ui.xmlfragment(this.getView().getId(), "student00.sap.training.valuehelp.view.Flights");
            var oDynPage = this.byId("idDynPage");
            oDynPage.destroyContent();
            oDynPage.setContent(oFragment);
            this._oFormFragments["Flights"] = oFragment;
            return oFragment;
        }
    });
});
```

Figure 204: Function

4. Extend the implementation of the event handler `onValueHelpClose` by setting the visibility of the `DynamicPage` object stored in the member variable *_oDynamicPage* to *true*. Call the function `_createFlightsTable` and assign the return value to a local variable. Bind the fragment to the binding path of the selected carrier. Also bind the UI control with id *idHeaderLayout* to that binding path. Implement the function.

Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/ui/model/Filter",
    "sap/ui/model/FilterOperator"
], function (Controller, Filter, FilterOperator) {
    "use strict";

    return Controller.extend("student00.sap.training.valuehelp.controller.Main", {
        _oFormFragments: {},

        onInit: function () {},

        onValueHelpRequest: function (oEvent) {},

        onValueHelpSearch: function (oEvent) {},

        onValueHelpClose: function (oEvent) {
            var oSelectedItem = oEvent.getParameter("selectedItem");
            if (oSelectedItem) {
                var oCarridInput = this.getView().byId(this._sInputId);
                oCarridInput.setValue(oSelectedItem.getTitle());
            }
            oEvent.getSource().getBinding("items").filter([]);
            this.oDynamicPage.setVisible(true);
            var oFlightsFragment = this._createFlightsTable();
            var sPath = oSelectedItem.getBindingContext().getPath();
            oFlightsFragment.bindElement(sPath);
            this.byId("idHeaderLayout").bindElement(sPath);
        },

        _createFlightsTable : function() {}
    });
});
```

Figure 205: Implement Function

5. Test your implementation.

The details of the selected carrier are displayed.

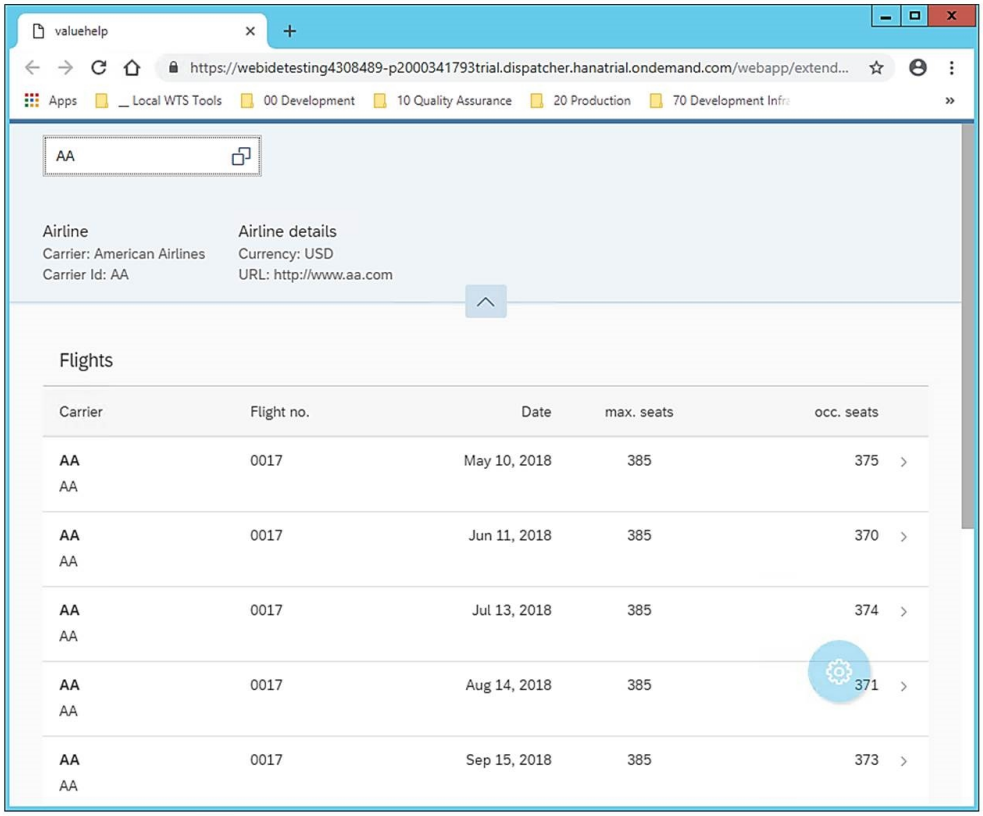


Figure 207: Carrier

Unit 12
Solution 11

Implement Value Helps

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, naming of fields and buttons as well as steps may differ from the exercise solution. The exercises show the application of group 00. Use your own student and group number for the exercises as assigned by your instructor.

Business Example

In the following exercise you will implement Value Help.

Task 1: Log on to the SAP Web IDE Full-Stack and create a SAPUI5 project

1. Log on to the *SAP Web IDE Full-Stack* and create an SAPUI5 project with the following settings:

   Table 45: Project Properties.

Parameter	Value
Project name	valuehelp
Namespace	student##.sap.training
View Type	XML
View Name	Main
SAPUI5 Version	1.60

   a) Choose *File* → *New* → *Project from Template* and choose the option *SAPUI5 Application Project* and *Next*.

   b) Name your project **`valuehelp`** and in the *Namespace* field enter the value **`student##.sap.training`**, select UI5 version 1.60, and choose *Next*.

   c) Choose XML as a View Type and enter the value **`Main`** in the *View Name* field and choose *Finish*.

   d) Your created project will appear in the project explorer of the *SAP Web IDE Full-Stack*.

Task 2: Implement value help

Implement the value help floorplan as described in the SAP Fiori Guidelines. This provides the user with a selection dialog to select a carrier. The application makes use of the OData service with the name UX_TRAVEL_SRV.

1. Add the UX_TRAVEL_SRV OData service to your project. The service is referenced by the default model of your application.

a) Select your project and from the context menu, choose *New → OData service*.

b) Choose *FSD_100 – FSD system* from the list of available systems.

c) Enter your username and password for the front-end server.

d) Search for the **`UX_TRAVEL_SRV service`**, then select the service from the list of results and choose *Next*.

e) Ensure the radio button labelled *Use default model* is selected and choose *Next*.

f) Complete the data source assignment by choosing *Finish*.

2. Replace the generated *sap.m.Page* control in the `Main.view.xml` with a *sap.f.DynamicPage* control. Assign the following attributes and values to the *sap.f.DynamicPage* control.

Table 46: sap.f.DynamicPage Attributes and Values.

Attribute	Value
id	idDynPage
headerExpanded	true
toggleHeaderOnTitleClick	true
preserveHeaderStateOnScroll	true

a) Open the file `Main.view.xml` and remove the generated *sap.m.Page* control.

b) Add an XML namespace with alias f to your implementation and assign the *sap.f-namespace*.

c) Add a control of type *sap.f.DynamicPage* with the listed attribute and value pairs to the pages-aggregation of the *sap.m.App-control* control.

d) Save your changes.

3. Add an *sap.f.DynamicPageTitle* control to the title aggregation of the *sap.f.DynamicPage* control. Implement a *sap.m.Input* control with the following attributes and values inside the heading aggregation of the *sap.f.DynamicPageTitle* control. Add the aggregation `suggestionItems` to the newly added *sap.m.Input* control and add an *sap.ui.core.Item* control as template. Assign the binding for attribute `Carrid` to the text attribute of the *sap.ui.core.Item* control

Table 47: sap.m.Input control Attributes and Values

Attribute	Value
id	idCarInput
placeholder	Enter carrier id
showSuggestion	true
showValueHelp	true
suggestionItems	{/UX_C_Carrier_TP}
type	Text

Attribute	Value
valueHelpRequest	onValueHelpRequest
submit	onSubmit

a) Open the file `Main.view.xml`.

b) Add the title aggregation to the *sap.f. DynamicPage* control.

c) Add a control of type *sap.f. DynamicPageTitle* inside the title aggregation.

d) Add the heading aggregation of the *sap.f.DynamicPageTitle* control.

e) Add an *sap.m.Input* control to the heading aggregation using the listed attributes. Add the `suggestionItems` aggregation to the control and insert an *sap.ui.core.Item* control with the described configuration. Remember to add an XML namespace alias for the sap.ui.core namespace. At the end your implementation should look like the following:

```
<pages>
    <f:DynamicPage id="idDynPage" headerExpanded="true" toggleHeaderOnTitleClick="true" preserveHeaderStateOnScroll="true">
    <f:title>
        <f:DynamicPageTitle>
            <f:heading>
                <Input id="idCarInput" placeholder="Enter carrier id ..." showSuggestion="true" showValueHelp="true" suggestionItems="{/UX_C_Carrier_TP}"
                    type="Text" valueHelpRequest="onValueHelpRequest" submit="onSubmit">
                    <suggestionItems>
                        <core:Item text="{Carrid}"/>
                    </suggestionItems>
                </Input>
            </f:heading>
        </f:DynamicPageTitle>
    </f:title>
```

Figure 175: Heading Aggregation

4. Create an XML fragment with the name **`CarrierSelectionDialog`** inside the view folder of your project. Implement selection *Dialog*, using the *sap.m.SelectDialog* control with the listed attributes and values. Add a template of type *sap.m.StandardListItem* to the new *sap.m.SelectDialog* and use the listed attributes and values.

Table 48: sap.m.SelectDialog Attributes and Values

Attribute	Value
title	Carriers
class	sapUiPopupWithPadding
item	{/UX_C_Carrier_TP}
search	onValueHelpSearch
confirm	onValueHelpClose
cancel	onValueHelpClose

Table 49: sap.m.StandardListItem Attributes and Values

Attribute	Value
iconDensityAware	false
iconInsert	false
title	{Carrid}

Attribute	Value
description	{Carrname}

a) Create a new file with the name **`CarrierSelectionDialog.fragment.xml`**. Select the view folder of your project and choose from the context menu *New* → *File*.

b) Enter **`CarrierSelectionDialog.fragment.xml`** in the new dialog and confirm *OK*

c) Add a UI control of type *sap.ui.core.FragmentDefinition* to the new file. Create two XML namespaces for the SAPUI5 namespace *sap.m* and *sap.ui.core*

d) Add a *sap.m.SelectDialog* to the fragment and assign the listed attribute and value pairs

e) Add a *sap.m.StandardListItem* control to the added *SelectDialog* control and add the listed attribute and value pairs.

f) At the end your implementation should look the following figure:

CarrierSelectionDialog.fragmen... ×

```
1  <core:FragmentDefinition
2      xmlns="sap.m"
3      xmlns:core="sap.ui.core">
4      <SelectDialog
5          title="Carriers"
6          class="sapUiPopupWithPadding"
7          items="{/UX_C_Carrier_TP}"
8          search="onValueHelpSearch"
9          confirm="onValueHelpClose"
10         cancel="onValueHelpClose">
11         <StandardListItem
12             iconDensityAware="false"
13             iconInset="false"
14             title="{Carrid}"
15             description="{Carrname}" />
16     </SelectDialog>
17 </core:FragmentDefinition>
```

Figure 176: Implementation

g) Save your changes and close the file.

Task 3: Task Controller Implementation

Implement the event handler *onValueHelpRequest* inside the `Main.controller.js` file. The implementation obtains the value from the source object of the event and assigns the value to a local variable named *sInputValue*. The ID of the source object is stored inside a member variable with name *_sInputId*.

1. Create an instance of `CarrierSelectionDialog` and store the instance inside a member variable with the name `_oValueHelpDialog`. Get the binding of the items aggregation from the dialog and apply a filter for *Carrid* with *FilterOperator.Contains* and remember the AMD (Asynchronous module definition). Pass the *sInputValue* variable to the filter, then open the dialog.

a) Open the file `Main.controller.js` and add a function with the name **`onValueHelpRequest`**.

*Main.controller.js ×

```
1  sap.ui.define([
2      "sap/ui/core/mvc/Controller"
3  ], function (Controller) {
4      "use strict";
5
6      return Controller.extend("student00.sap.training.valuehelp.controller.Main", {
7          onValueHelpRequest: function (oEvent) {
8
9          }
10     });
11 });
```

Figure 177: Main.controller.js

b) Create the two required variables and assign the appropriate values.

Main.controller.js ×

```
1  sap.ui.define([
2      "sap/ui/core/mvc/Controller"
3  ], function (Controller) {
4      "use strict";
5
6      return Controller.extend("student00.sap.training.valuehelp.controller.Main", {
7          onValueHelpRequest: function (oEvent) {
8              var sInputValue = oEvent.getSource().getValue();
9              this._sInputId = oEvent.getSource().getId();
10
11         }
12     });
13 });
```

Figure 178: Variables

c) Create an instance of *CarrierSelectionDialog* and assign the instance to the member variable with the name *_oValueHelpDialog*. Set a reference.

Main.controller.js ×

```
1  sap.ui.define([
2      "sap/ui/core/mvc/Controller"
3  ], function (Controller) {
4      "use strict";
5
6      return Controller.extend("student00.sap.training.valuehelp.controller.Main", {
7          onValueHelpRequest: function (oEvent) {
8              var sInputValue = oEvent.getSource().getValue();
9              this._sInputId = oEvent.getSource().getId();
10             if (!this._oValueHelpDialog) {
11                 this._oValueHelpDialog = sap.ui.xmlfragment(
12                     "student00.sap.training.valuehelp..view.CarrierSelectionDialog",
13                     this
14                 );
15                 this.getView().addDependent(this._oValueHelpDialog);
16             }
17
18         }
19     });
20 });
```

Figure 179: Dialog

d) Add a reference to *sap.ui.model.Filter* and *sap.ui.model.FilterOperator* to your controller.

*Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/ui/model/Filter",
    "sap/ui/model/FilterOperator"
], function (Controller, Filter, FilterOperator) {
    "use strict";

    return Controller.extend("student00.sap.training.valuehelp.controller.Main", {
        onValueHelpRequest: function (oEvent) {
            var sInputValue = oEvent.getSource().getValue();
            this._sInputId = oEvent.getSource().getId();
            if (!this._oValueHelpDialog) {
                this._oValueHelpDialog = sap.ui.xmlfragment(
                    "student00.sap.training.valuehelp..view.CarrierSelectionDialog",
                    this
                );
                this.getView().addDependent(this._oValueHelpDialog);
            }

        }
    });
});
```

Figure 180: Reference

e) Implement the filter functionality.

Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/ui/model/Filter",
    "sap/ui/model/FilterOperator"
], function (Controller, Filter, FilterOperator) {
    "use strict";

    return Controller.extend("student00.sap.training.valuehelp.controller.Main", {
        onValueHelpRequest: function (oEvent) {
            var sInputValue = oEvent.getSource().getValue();
            this._sInputId = oEvent.getSource().getId();
            if (!this._oValueHelpDialog) {
                this._oValueHelpDialog = sap.ui.xmlfragment(
                    "student00.sap.training.valuehelp..view.CarrierSelectionDialog",
                    this
                );
                this.getView().addDependent(this._oValueHelpDialog);
            }
            this._oValueHelpDialog.getBinding("items").filter([new Filter(
                "Carrid",
                FilterOperator.Contains, sInputValue
            )]);


        }
    });
});
```

Figure 181: Filter

f) Call the open-function on the dialog instance to show the requested value help dialog.

Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/ui/model/Filter",
    "sap/ui/model/FilterOperator"
], function (Controller, Filter, FilterOperator) {
    "use strict";

    return Controller.extend("student00.sap.training.valuehelp.controller.Main", {
        onValueHelpRequest: function (oEvent) {
            var sInputValue = oEvent.getSource().getValue();
            this._sInputId = oEvent.getSource().getId();
            if (!this._oValueHelpDialog) {
                this._oValueHelpDialog = sap.ui.xmlfragment(
                    "student00.sap.training.valuehelp..view.CarrierSelectionDialog",
                    this
                );
                this.getView().addDependent(this._oValueHelpDialog);
            }
            this._oValueHelpDialog.getBinding("items").filter([new Filter(
                "Carrid",
                FilterOperator.Contains, sInputValue
            )]);

            this._oValueHelpDialog.open(sInputValue);
        }
    });
});
```

Figure 182: Dialog

2. Test the status of your implementation.

a) Select the file `index.html` and from the context menu, choose *Run → Run as Web Application*.

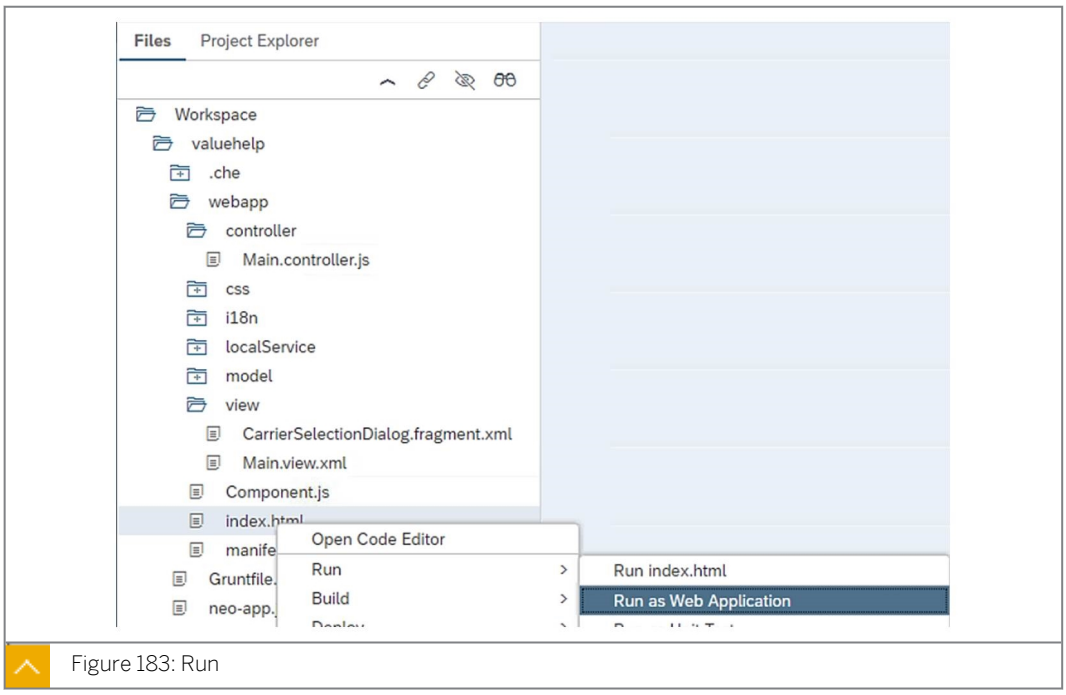


Figure 183: Run

b) Enter your front-end server credentials when required.

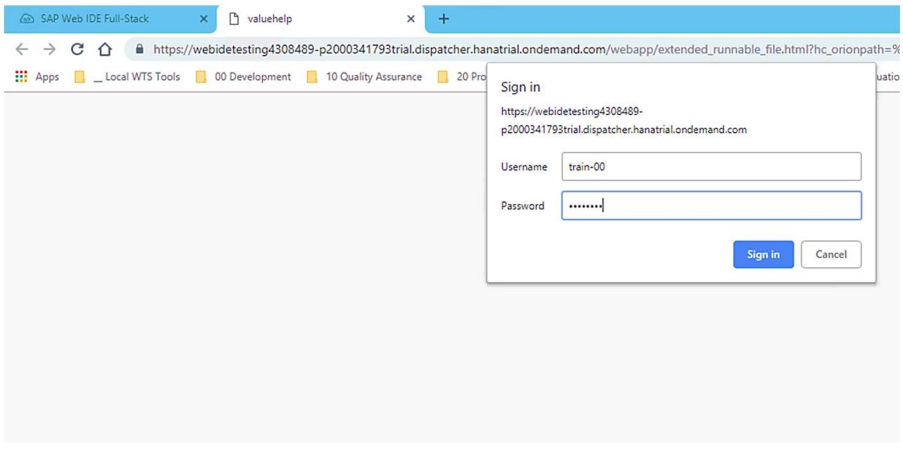


Figure 184: Sign In

c) Enter a carrier ID in the *value help* field.

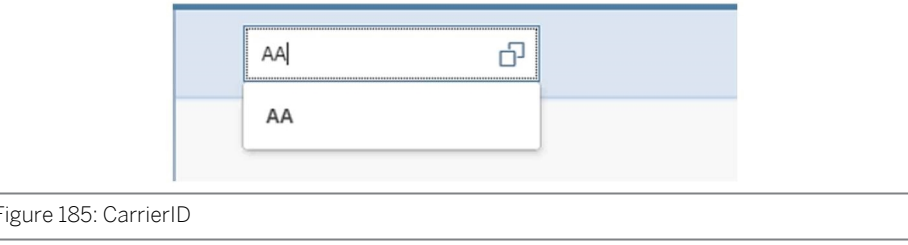


Figure 185: CarrierID

d) Press *F4*.

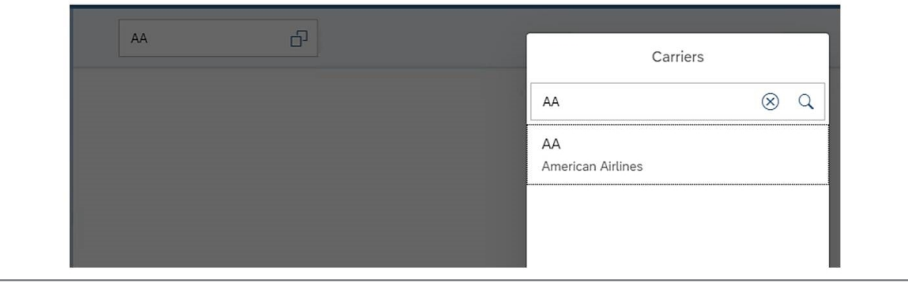


Figure 186: F4

e) Close the dialog. Currently, there is no functionality implemented when closing or selecting a carrier from the list.

3. Implement the function `onValueHelpSearch` to handle events of `SelectDialog`. The `onValueHelpSearch` function is called when the user enters a value in the search field of the dialog and starts the search. The function needs to read the event parameter value and use this value for filtering. Make it possible for the user to search for the carrier name (property carrname, FilterOperator.StartsWith) and carrier ID (property Carrid, FilterOperator.Contains).

   a) Open the file `Main.controller.js`.

   b) Add an event handler function `onValueHelpSearch` to the *Main* controller.

Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/ui/model/Filter",
    "sap/ui/model/FilterOperator"
], function (Controller, Filter, FilterOperator) {
    "use strict";

    return Controller.extend("student00.sap.training.valuehelp.controller.Main", {
        onValueHelpRequest: function (oEvent) {},

        onValueHelpSearch: function (oEvent) {

        }
    });
});
```

Figure 187: EventHandler

c) Obtain the value parameter from the event object and assign it to a local variable.

*Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/ui/model/Filter",
    "sap/ui/model/FilterOperator"
], function (Controller, Filter, FilterOperator) {
    "use strict";

    return Controller.extend("student00.sap.training.valuehelp.controller.Main", {
        onValueHelpRequest: function (oEvent) {},

        onValueHelpSearch: function (oEvent) {
            var sValue = oEvent.getParameter("value");
        }
    });
});
```

Figure 188: Value Parameter

d) Create the necessary filter objects and start the filter process.

Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/ui/model/Filter",
    "sap/ui/model/FilterOperator"
], function (Controller, Filter, FilterOperator) {
    "use strict";

    return Controller.extend("student00.sap.training.valuehelp.controller.Main", {
        onValueHelpRequest: function (oEvent) {},

        onValueHelpSearch: function (oEvent) {
            var sValue = oEvent.getParameter("value");
            var oFilter = new Filter(
                "Carrid",
                FilterOperator.Contains, sValue
            );

            var oFilter2 = new Filter(
                "Carrname",
                FilterOperator.StartsWith, sValue
            );

            var oFilter3 = new Filter({
                filters: [oFilter, oFilter2],
                and: false
            });

            oEvent.getSource().getBinding("items").filter([oFilter3]);
        }
    });
});
```

Figure 189: Filter

4. Implement the function `onValueHelpClose` to handle the event of the `SelectDialog`. The `onValueHelpClose` function is called when the dialog is closed. The event object of the event handler will get a parameter *selectedItem*. The parameter contains the item of the list selected by the user. Read the title property from the selected item and display the title in the input field of *Main.view.xml*. Recall that you stored the id of the input field in the member variable *_sInputId*. Reset the filters from the select dialog to complete the implementation. Now test your application.

   a) Add a function `onValueHelpClose` to the implementation of `Main.controller.js`.

```
Main.controller.js
1  sap.ui.define([
2      "sap/ui/core/mvc/Controller",
3      "sap/ui/model/Filter",
4      "sap/ui/model/FilterOperator"
5  ], function (Controller, Filter, FilterOperator) {
6      "use strict";
7
8      return Controller.extend("student00.sap.training.valuehelp.controller.Main", {
9          onValueHelpRequest: function (oEvent) {},
26
27         onValueHelpSearch: function (oEvent) {},
46
47         onValueHelpClose: function (oEvent) {
48
49         }
50     });
51 });
```

Figure 190: Function

   b) Implement the function.

```
Main.controller.js
1  sap.ui.define([
2      "sap/ui/core/mvc/Controller",
3      "sap/ui/model/Filter",
4      "sap/ui/model/FilterOperator"
5  ], function (Controller, Filter, FilterOperator) {
6      "use strict";
7
8      return Controller.extend("student00.sap.training.valuehelp.controller.Main", {
9          onValueHelpRequest: function (oEvent) {},
26
27         onValueHelpSearch: function (oEvent) {},
46
47         onValueHelpClose: function (oEvent) {
48             var oSelectedItem = oEvent.getParameter("selectedItem");
49             if (oSelectedItem) {
50                 var oCarridInput = this.getView().byId(this._sInputId);
51                 oCarridInput.setValue(oSelectedItem.getTitle());
52             }
53             oEvent.getSource().getBinding("items").filter([]);
54         }
55     });
56 });
```

Figure 191: Implement Function

   c) Restart your application and open the value help dialog.

   d) Enter a carrier ID or carrier name in the search field of the value help dialog and start the search.

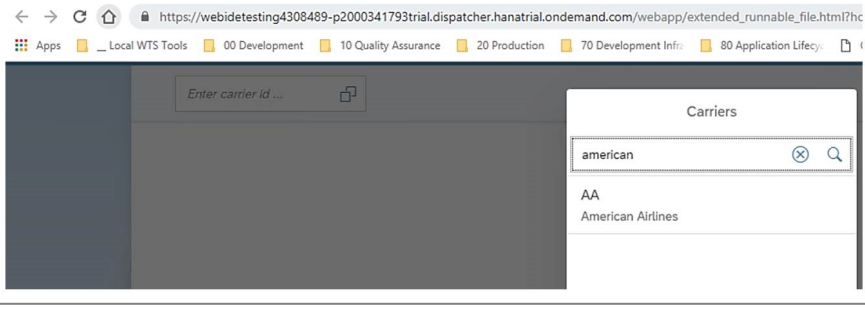


Figure 192: Value Help

e) Select one carrier from the list. The dialog closes and the carrier ID is shown in the input field.

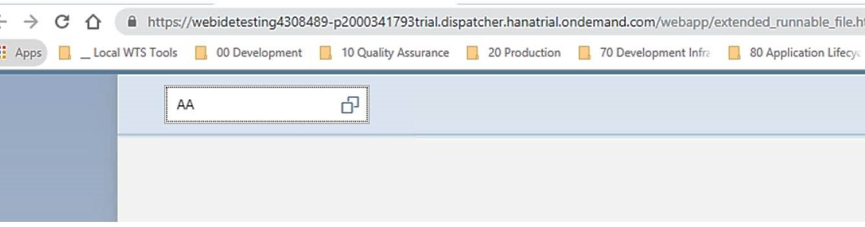


Figure 193: Carrier

Task 4: Show the details and the available flights of the selected carrier.
Add a fragment to the project to show the flights for a selected carrier.

1. Implement a new fragment with the name **`Flights.fragment.xml`** in the view folder of your project and add XML namespace aliases for `sap.m` as default and `sap.ui.core` with alias core to the `sap.ui.core.FragmentDefinition`.

   a) Select the view folder and choose from the context menu.

   b) Enter the name for the file you are creating in the input field and confirm *OK*.

   c) Implement the FragmentDefinition and add the required namespaces.

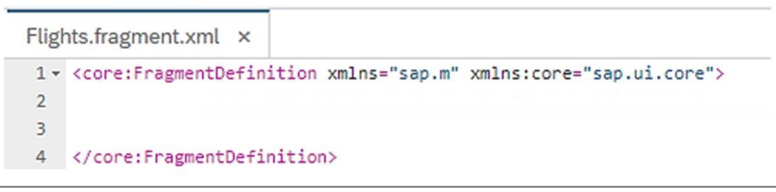


Figure 194: Namespaces

2. Add an *sap.m.Table* control with the listed attributes and values to the newly-created fragment.

Table 50: sap.m.Table control Attributes and Values.

Attribute	Value
id	idFlights
items	{ path: 'to_Flight', sorter: { path: 'Carrid' }}
mode	SingleSelectMaster
growingThreshold	true

Attribute	Value
growing	true
visible	true

Add an sap.m.Table control to your fragment.

Figure 195: Table Control

3. Add an *sap.m.Title* to the `headerToolbar` aggregation of the *sap.m.Table* with the listed attributes and values.

Table 51: sap.m.Title control Attributes and Values.

Attribute	Value
text	Flights
level	H2

Add the `headerToolbar` aggregation to the *sap.m.Table* control and add an *sap.m.Title* control with the listed attributes and values.

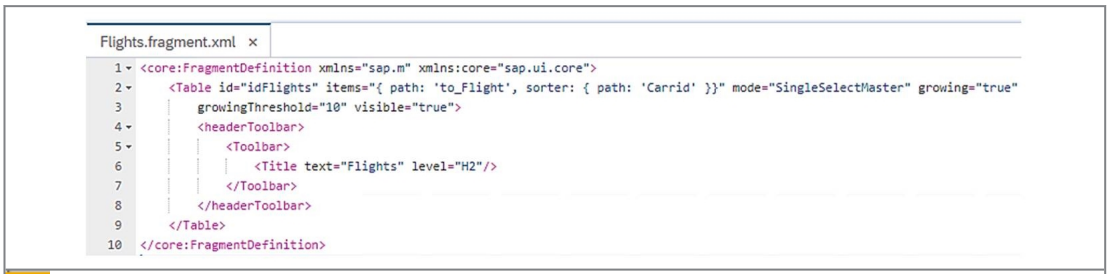


Figure 196: Table and Title Controls

4. Add columns to the table with the listed attributes and values.
   a) First column-control:

Table 52: First column-control Attribute and Value

Attribute	Value
width	12em

   b) Second column-control:

Table 53: Second column-control Attributes and Values

Attribute	Value
minScreenWidth	Tablet
demandPopin	true

c) Third column-control:

Table 54: Third column-control Attributes and Values

minScreenWidth	Tablet
demandPopin	true
hAlign	Right

d) Fourth column-control:

Table 55: Fourth column-control Attributes and Values

Attribute	Value
minScreenWidth	Tablet
demandPopin	true
hAlign	Center

Flights.fragment.xml

```
<core:FragmentDefinition xmlns="sap.m" xmlns:core="sap.ui.core">
    <Table id="idFlights" items="{ path: 'to_Flight', sorter: { path: 'Carrid' }}" mode="SingleSelectMaster" growing="true"
        growingThreshold="10" visible="true">
        <headerToolbar>
            <Toolbar>
                <Title text="Flights" level="H2"/>
            </Toolbar>
        </headerToolbar>
        <columns>
            <Column width="12em">
            </Column>
            <Column minScreenWidth="Tablet" demandPopin="true">
            </Column>
            <Column minScreenWidth="Tablet" demandPopin="true" hAlign="Right">
            </Column>
            <Column minScreenWidth="Tablet" demandPopin="true" hAlign="Center">
            </Column>
            <Column hAlign="Right">
            </Column>
        </columns>
    </Table>
</core:FragmentDefinition>
```

Figure 197: Controls

e) Fifth column-control:

Table 56: Fifth column-control Attribute and Value.

Attribute	Value
hAlign	Right

5. Add one *sap.m.Text* control to each column using the listed attributes.

Table 57: sap.m.Text control Attributes and Values

Attribute	Value
text	Carrier
text	Flight no.
text	Date
text	Max seats
text	Occ. seats

Flights.fragment.xml

```
<core:FragmentDefinition xmlns="sap.m" xmlns:core="sap.ui.core">
    <Table id="idFlights" items="{ path: 'to_Flight', sorter: { path: 'Carrid' }}" mode="SingleSelectMaster" growing="true"
        growingThreshold="10" visible="true">
        <headerToolbar>
            <Toolbar>
                <Title text="Flights" level="H2"/>
            </Toolbar>
        </headerToolbar>
        <columns>
            <Column width="12em">
                <Text text="Carrier"/>
            </Column>
            <Column minScreenWidth="Tablet" demandPopin="true">
                <Text text="Flight no."/>
            </Column>
            <Column minScreenWidth="Tablet" demandPopin="true" hAlign="Right">
                <Text text="Date"/>
            </Column>
            <Column minScreenWidth="Tablet" demandPopin="true" hAlign="Center">
                <Text text="max. seats"/>
            </Column>
            <Column hAlign="Right">
                <Text text="occ. seats"/>
            </Column>
        </columns>
    </Table>
</core:FragmentDefinition>
```

Figure 198: Text Control

Add an *sap.m.Text* control to each of the columns and assign the attributes and values.

6. Add an `sap.m.ColumnListItem` to the items aggregation of the table and add UI controls to the cells aggregation of the `sap.m.ColumnListItem`.
   a) In the first cell, insert a UI control of type `sap.m.ObjectIdentifier` and bind the attribute title to the `Carrid` property of the entity.
   b) In the second cell, insert a UI control of type `sap.m.Text` and bind the attribute text to the property *Connid* of the entity.
   c) In the third cell, insert a UI control of type *sap.m.Text*. Bind the attribute text to the property `Fldate` and declare that the property is of type `sap.ui.model.type.Date` of the entity.
   d) In the fourth cell, insert a UI control of type *sap.m.Text* and bind the attribute text to the property `Seatsmaxx` of the entity.
   e) In the fifth cell, insert a UI control of type *sap.m.Text* and bind the attribute text to the property `Seatsocc` of the entity.
   f) Add the items aggregation to the table.
   g) Add the *sap.m.ColumnListItem* control to the items aggregation and add the controls and binding.

Task 5: Extend the implementation of the Main.view.xml so it displays the details of the selected carrier

1. Add an *sap.f.DynamicPageHeader* to the header aggregation of the `DynamicPage` control with the listed attributes and values.

   Table 58: sap.f.DynamicPageHeader Attributes and Values.

Attribute	Value
pinnable	false

   a) Open the `Main.view.xml` from the view folder of your project.

   b) Add the header aggregation to the `DynamicPage` control and add a control of type *sap.f.DynamicPageHeader* with the listed property.

2. Add an *sap.m.FlexBox* control to the content aggregation with the listed attributes and values.

   Table 59: sap.m.FlexBox control Attributes and Values.

Attribute	Value
alignItems	Start
justifyContent	SpaceBetween
id	idHeaderLayout
visible	false

   a) Add the control aggregation to the `DynamicPageHeader` control.

   b) Add an *sap.m.FlexBox* control to the content aggregation and assign the defined attributes and values.

```
<f:header>
    <f:DynamicPageHeader pinnable="false">
        <f:content>
            <FlexBox alignItems="Start" justifyContent="SpaceBetween" id="idHeaderLayout" visible="false">
                <items>


                </items>
            </FlexBox>
        </f:content>
    </f:DynamicPageHeader>
</f:header>
```

   Figure 199: Items Aggregation

3. Add an *sap.m.Panel* control to the items aggregation of the *sap.m.FlexBox* control and add the listed attributes.

   Table 60: sap.m.Panel control Attributes and Values

Attribute	
backgroundDesign	Transparent

Attribute	
class	sapUiNoContentPadding sapUiSmallMarginTop

a) Add the items aggregation to the FlexBox control.

b) Add a control of type *sap.m.Panel* and add the listed attributes.

4. Add a *sap.ui.layout. HorizontalLayout* with the following attribute and value to the content aggregation of the `sap.m.Panel` control. Use layout as an XML namespace alias.

Table 61: sap.ui.layout. HorizontalLayout Attribute and Value

Attribute	Value
allowWrapping	true

a) Add the XML namespace alias layout pointing to the *UI5-namespace sap.ui.layout*.

b) Add the content aggregation to the panel control.

c) Add a control of type *sap.ui.layout.HorizontalLayout* to the content aggregation.

5. Add an *sap.ui.layout.VerticalLayout* with the listed attributes and values to the content aggregation of the *sap.ui.layout.HorizontalLayout* and add the following UI controls. Add an *sap.m.Text* control with attribute text and assign the value **`Airline`** to the control. Add an *sap.m.ObjectAttribute* control with the attributes title and text. Assign the value **`Carrier`** to the title attribute and bind the text attribute to the property *Carrname* of the entity. Add an *sap.m.ObjectAttribute* control with the attributes title and text. Assign the value **`Carrier id`** to the title attribute and bind the text attribute to the property *Carrid* of the entity.

a) Add a control of type *sap.ui.layout.VerticalLayout*.

b) Add the UI controls to the `VerticalLayout` control and bind the attributes as defined.

6. Add an *sap.ui.layout.VerticalLayout* with the following attributes and values to the content aggregation of the *sap.ui.layout.HorizontalLayout* and add the following UI controls. Add the attribute class to the `VerticalLayout` control and assign the value **`sapUiMediumMarginBegin`** to the attribute. Add an *sap.m.Text* control with attribute text and assign the value **`Airline`** details to the control. Add an *sap.m.ObjectAttribute* control with the attributes title and text. Assign the value **`Currency`** to the title attribute and bind the text attribute to the property *Currcode* of the entity. Add an *sap.m.ObjectAttribute* control with the attributes title and text. Assign the value **`URL`** to the title attribute and bind the text attribute to the property *URL* of the entity.

a) Add another *sap.ui.layout.VerticalLayout* control to the content aggregation.

b) Add the UI controls and the specified binding.

c) At the end your implementation should look like the following figure:

```
<f:header>
    <f:DynamicPageHeader pinnable="false">
        <f:content>
            <FlexBox alignItems="Start" justifyContent="SpaceBetween" id="idHeaderLayout" visible="false">
                <items>
                    <Panel backgroundDesign="Transparent" class="sapUiNoContentPadding sapUiSmallMarginTop">
                        <content>
                            <layout:HorizontalLayout allowWrapping="true">
                                <layout:VerticalLayout>
                                    <Title text="Airline"/>
                                    <ObjectAttribute title="Carrier" text="{Carrname}"/>
                                    <ObjectAttribute title="Carrier Id" text="{Carrid}"/>
                                </layout:VerticalLayout>
                                <layout:VerticalLayout class="sapUiMediumMarginBegin">
                                    <Title text="Airline details"/>
                                    <ObjectAttribute title="Currency" text="{Currcode}"/>
                                    <ObjectAttribute title="URL" text="{Url}"/>
                                </layout:VerticalLayout>
                            </layout:HorizontalLayout>
                        </content>
                    </Panel>
                </items>
            </FlexBox>
        </f:content>
    </f:DynamicPageHeader>
</f:header>
```

Figure 200: Implementation

7. Add an *sap.m.MessagePage* control to the content aggregation of the *sap.f.DynamicPage* control with the listed attributes and values.

   Table 62: sap.m.MessagePage control Attributes and Values.

	Value
showHeader	false
icon	sap-icon://search

   a) Add the content aggregation to the *sap.f.DynamicPage* control.

   b) Add a *sap.m.MessagePage* control to the content aggregation and add the listed attributes to the control.

   c) At the end your implementation should look like the following figure:

```
19      <f:header>
20          <f:DynamicPageHeader pinnable="false">
21              <f:content>
22                  <FlexBox alignItems="Start" justifyContent="SpaceBetween" id="idHeaderLayout" visible="false">
23                      <items>
24                          <Panel backgroundDesign="Transparent" class="sapUiNoContentPadding sapUiSmallMarginTop">
25                              <content>
26                                  <layout:HorizontalLayout allowWrapping="true">
27                                      <layout:VerticalLayout>
28                                          <Title text="Airline"/>
29                                          <ObjectAttribute title="Carrier" text="{Carrname}"/>
30                                          <ObjectAttribute title="Carrier Id" text="{Carrid}"/>
31                                      </layout:VerticalLayout>
32                                      <layout:VerticalLayout class="sapUiMediumMarginBegin">
33                                          <Title text="Airline details"/>
34                                          <ObjectAttribute title="Currency" text="{Currcode}"/>
35                                          <ObjectAttribute title="URL" text="{Url}"/>
36                                      </layout:VerticalLayout>
37                                  </layout:HorizontalLayout>
38                              </content>
39                          </Panel>
40                      </items>
41                  </FlexBox>
42              </f:content>
43          </f:DynamicPageHeader>
44      </f:header>
45      <f:content>
46          <MessagePage showHeader="false" icon="sap-icon://search"/>
47      </f:content>
48  </f:DynamicPage>
49  </pages>
```

Figure 201: Implementation

Task 6: Implement controller logic to display the carrier details when the user selects a carrier from the list of carriers

1. Open the file `Main.controller.js` to declare a member variable *_oFormFragments* and add to the implementation by assigning a literal JavaScript object to the variable. This variable acts as a map for fragments. Fragment instances can be stored by name in the map.

Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/ui/model/Filter",
    "sap/ui/model/FilterOperator"
], function (Controller, Filter, FilterOperator) {
    "use strict";

    return Controller.extend("student00.sap.training.valuehelp.controller.Main", {
        _oFormFragments: {},
        onValueHelpRequest: function (oEvent) {},

        onValueHelpSearch: function (oEvent) {},

        onValueHelpClose: function (oEvent) {}
    });
});
```

Figure 202: Member Variable

2. Add and implement the `onInit` function of the controller from the *Main* view. Obtain the UI control *idHeaderLayout* reference and assign the reference to a member variable of the controller. Name the variable *_oDynamicPage* and set the visibility of the layout to *false*.

Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/ui/model/Filter",
    "sap/ui/model/FilterOperator"
], function (Controller, Filter, FilterOperator) {
    "use strict";

    return Controller.extend("student00.sap.training.valuehelp.controller.Main", {
        _oFormFragments: {},

        onInit: function () {
            this.oDynamicPage = this.byId("idHeaderLayout");
            this.oDynamicPage.setVisible(false);
        },

        onValueHelpRequest: function (oEvent) {},

        onValueHelpSearch: function (oEvent) {},

        onValueHelpClose: function (oEvent) {}
    });
});
```

Figure 203: Function

3. Implement the function named `_createFlightsTable`. The function creates an instance of the flights fragment. The function checks if an instance of the fragment was already created. When the instance of the fragment map exists it should be returned. If not the instance must be created. Assign the instance to the map and it should return to the caller. After instantiation of the fragment, obtain a reference to the UI control with ID *idDynPage*. Call on the obtained reference `destroyContent` function and add the new fragment object by calling the `setContent` function on the dynamic page object.

Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/ui/model/Filter",
    "sap/ui/model/FilterOperator"
], function (Controller, Filter, FilterOperator) {
    "use strict";

    return Controller.extend("student00.sap.training.valuehelp.controller.Main", {
        _oFormFragments: {},

        onInit: function () {},

        onValueHelpRequest: function (oEvent) {},

        onValueHelpSearch: function (oEvent) {},

        onValueHelpClose: function (oEvent) {},

        _createFlightsTable : function() {
            var oFragment = this._oFormFragments["Flights"];
            if (oFragment) {
                return oFragment;
            }
            oFragment = sap.ui.xmlfragment(this.getView().getId(), "student00.sap.training.valuehelp.view.Flights");
            var oDynPage = this.byId("idDynPage");
            oDynPage.destroyContent();
            oDynPage.setContent(oFragment);
            this._oFormFragments["Flights"] = oFragment;
            return oFragment;
        }
    });
});
```

Figure 204: Function

4. Extend the implementation of the event handler `onValueHelpClose` by setting the visibility of the `DynamicPage` object stored in the member variable *_oDynamicPage* to *true*. Call the function `_createFlightsTable` and assign the return value to a local variable. Bind the fragment to the binding path of the selected carrier. Also bind the UI control with id *idHeaderLayout* to that binding path. Implement the function.

Main.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/ui/model/Filter",
    "sap/ui/model/FilterOperator"
], function (Controller, Filter, FilterOperator) {
    "use strict";

    return Controller.extend("student00.sap.training.valuehelp.controller.Main", {
        _oFormFragments: {},

        onInit: function () {},

        onValueHelpRequest: function (oEvent) {},

        onValueHelpSearch: function (oEvent) {},

        onValueHelpClose: function (oEvent) {
            var oSelectedItem = oEvent.getParameter("selectedItem");
            if (oSelectedItem) {
                var oCarridInput = this.getView().byId(this._sInputId);
                oCarridInput.setValue(oSelectedItem.getTitle());
            }
            oEvent.getSource().getBinding("items").filter([]);
            this.oDynamicPage.setVisible(true);
            var oFlightsFragment = this._createFlightsTable();
            var sPath = oSelectedItem.getBindingContext().getPath();
            oFlightsFragment.bindElement(sPath);
            this.byId("idHeaderLayout").bindElement(sPath);
        },

        _createFlightsTable : function() {}
    });
});
```

Figure 205: Implement Function

5. Test your implementation.

a) Restart your application.

b) The application is shown without a preselected carrier.

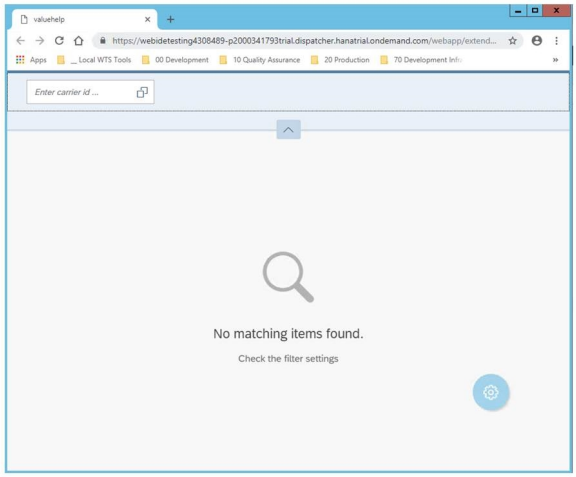


Figure 206: Value Help Application

c) Select a carrier from *F4 value help* as described.

The details of the selected carrier are displayed.

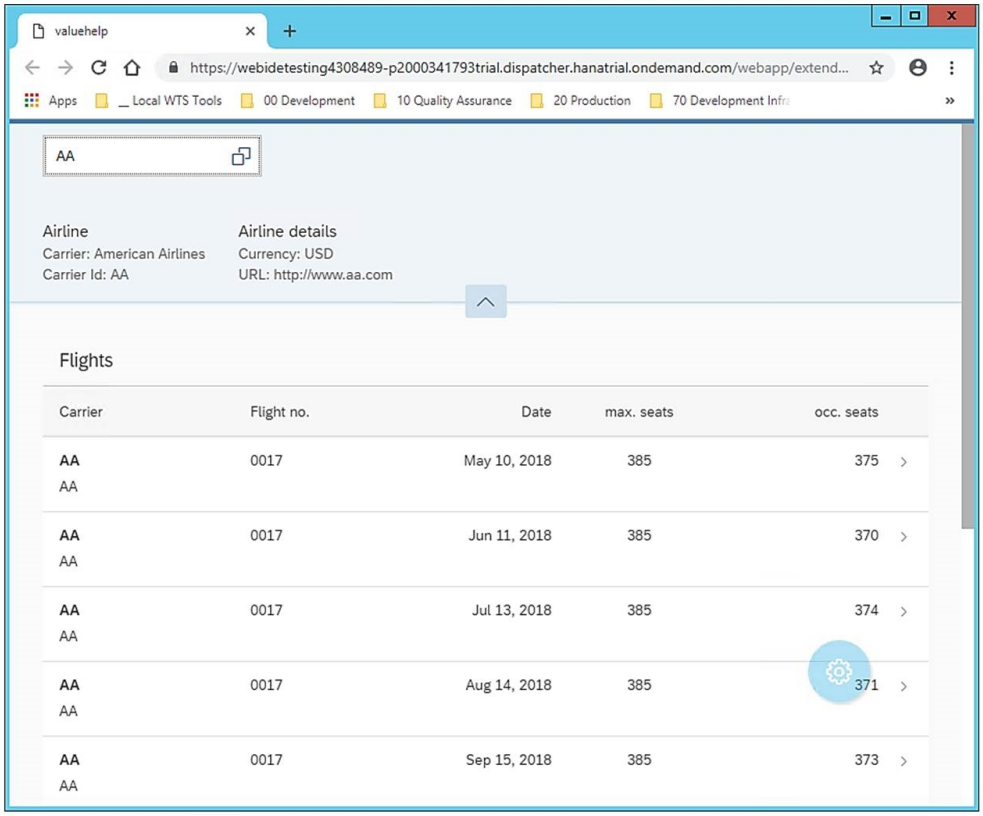


Figure 207: Carrier

Unit 12
Exercise 12

Implement a List Report

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, step procedures and naming of fields and buttons may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

Business Example

In this exercise you learn how to extend a standard SAP Fiori application to create a list report floorplan.

This exercise uses the flexiblecolumnlayout project created in the previous exercise..

Task 1: Create a New Project

Export the project `flexiblecolumnlayout` that you created in the previous project and import the project with the name *listreport* into the workspace of your *SAP Web IDE Full-Stack*.

1. Log on to the SAP Web IDE Full-Stack, and export the project `flexiblecolumnlayout` that you created in one of the previous exercises.
2. Import the content of the file `flexiblecolumnlayout.zip` as a project with the name `listreport`.
3. Add the attribute id to the table control in the Overview-view and assign the value idCarrierList to the newly added attribute. Implement a search functionality to search the table of carriers, as described in the SAP Fiori guidelines. Add an *sap.m.SearchField* control and place the control in the *sap.m.Toolbar*. Add an *sap.m.ToolbarSpacer* control between the existing *sap.m.Title* control and the newly added *sap.m.SearchField* control. Implement the `onSearch` function. Use the listed properties as attributes for the *SearchField* control. The search functionality returns a list of carriers where the `carrid` or the `carrname` matches the specified value.

   Table 63: SearchField Control Properties

Attribute	Value
placeholder	Filter
search	onSearch
width	15rem

Note:
Implement the search functionality in a reusable way as we use the same search and filter aspects in a later step.

Note:
Use the i18n model for the placeholder text. Add the value **`Filter`** with the key **`sfPlaceholder`** and type **`#XTXT`** into the i18n.property file.

4. Test the search implementation.
5. Add a UI control of type `sap.f.DynamicPageHeader` to the header aggregation of the existing `sap.f.DynamicPage` control located in the Overview.view.xml file. Assign the following attribute and value to the newly added `sap.f.DynamicPageHeader` control.

   Table 64: sap.f.DynamicPageHeader attributes and values

Attribute	Value
pinnable	false

6. Add a UI control of type `sap.ui.comp.filterbar.FilterBar` to the content aggregation of the `sap.f.DynamicPageHeader` control. Use an XML namespace with alias **`fb`** to assign the SAPUI5 `sap.ui.comp.filterbar`. Add the listed attribute and values to the newly added `sap.ui.comp.filterbar.FilterBar` control.

   Table 65: sap.ui.comp.filterbar.FilterBar control attribute and values

Attribute	Value
id	idFilterBar
reset	onReset
search	onSearch
clear	onClear
cancel	onCancel
filtersDialogClose	onFiltersDialogClose
useToolbar	true
showGoOnFB	true
showRestoreButton	true
showClearButton	true

7. Add two `sap.ui.comp.filterbar.FilterGroupItem` controls to the *filterItems* aggregation of the `sap.ui.comp.filterbar.FilterBar` with the following attributes and values.

Table 66: sap.ui.comp.filterbar.FilterGroupItem Control Attributes and Values

Arttribute	Value
groupName	G1
visibleInFilterBar	true
groupTitle	carridGroup
name	A
label	Carrier

Note:
Use the i18n *ResourceModel* and add the key *filterCarId* to the i18n.properties file. Assign the value **`Carrier`** to the key and use this key instead of the hardwired text.

Table 67: sap.ui.comp.filterbar.FilterGroupItem Control Attributes and Values

Attribute	Value
groupName	G2
visibleInFilterBar	true
groupTitle	carrnameGroup
name	B
label	Carrier name

8. Add a control of type `sap.m.Input` to each control aggregation of the `sap.ui.comp.filter.FilterGroupItem` controls. Assign the listed attributes and values to the controls.

Table 68: sap.m.Input Control 1 Attributes and Values

Attribute	Value
placeholder	{i18n>filterCarId}
liveChange	onFilterChange
id	idFilterCarrid

Table 69: sap.m.Input Control 2 Attributes and Values

Attribute	Value
placeholder	{i18n>filterCarname}
liveChange	onFilterChange

Attribute	Value
ID	idFilterCarrname

Add a *sap.m.Input* control to each of the *sap.ui.comp.filter.FilterGroupItem* controls and assign the listed attribute and values.

Figure 231: Add an Input Control

9. Declare a member variable with the name **`oFilterBar`** in the `onInit` function of the *Overview* controller and assign a reference to the *sap.ui.comp.filter.FilterBar* control. Declare another member with the name **`oCarIdIF`** and assign the reference to the UI control with ID *idFilterCarrid*. Declare a member variable with the name **`oCarNameIF`** and assign a reference to the UI control with the ID *idFilterCarrname*.

10. Implement the function `onClear` in the controller of the `Overview.controller.js` as shown.

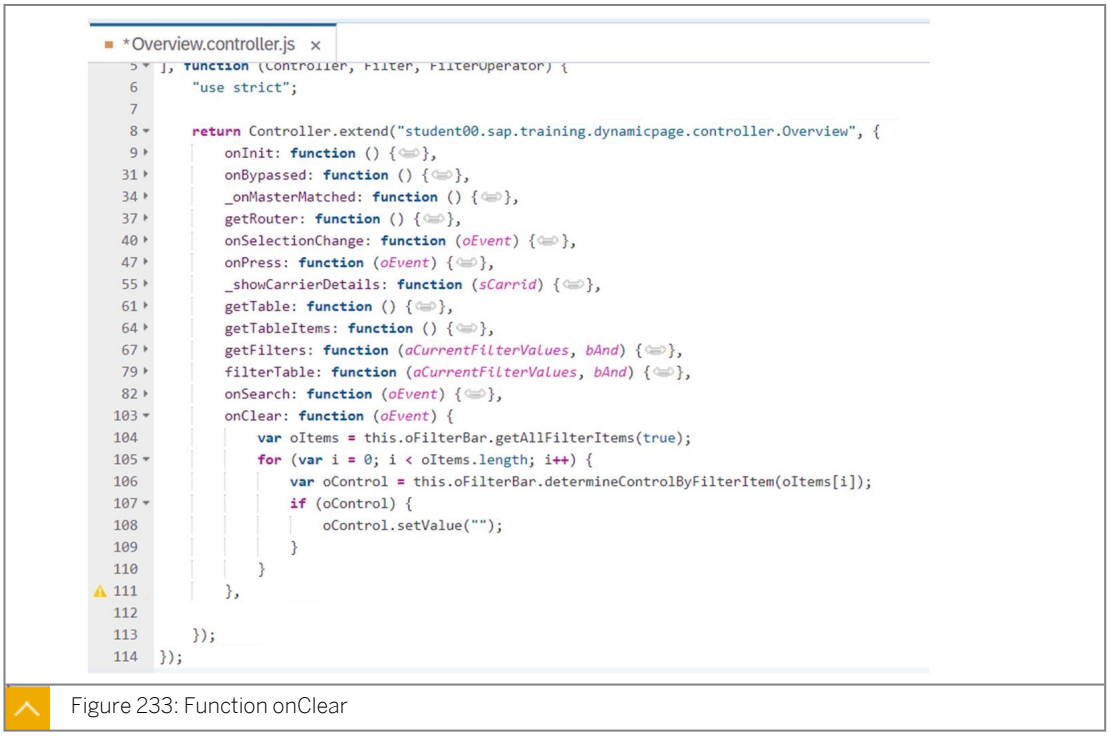


Figure 233: Function onClear

11. Implement a function `onFilterChange` to fire the `onFilterChange` event.

Overview.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/ui/model/Filter",
    "sap/ui/model/FilterOperator"
], function (Controller, Filter, FilterOperator) {
    "use strict";

    return Controller.extend("student00.sap.training.dynamicpage.controller.Overview", {
        onInit: function () {},
        onBypassed: function () {},
        _onMasterMatched: function () {},
        getRouter: function () {},
        onSelectionChange: function (oEvent) {},
        onPress: function (oEvent) {},
        _showCarrierDetails: function (sCarrid) {},
        getTable: function () {},
        getTableItems: function () {},
        getFilters: function (aCurrentFilterValues, bAnd) {},
        filterTable: function (aCurrentFilterValues, bAnd) {},
        onSearch: function (oEvent) {},
        onClear: function (oEvent) {},
        onFilterChange: function (oEvent) {
            this.oFilterBar.fireFilterChange(oEvent);
        }

    });
});
```

Figure 234: Implement Function onFilterChange

12. Implement the function `onReset` and reset the applied filters on the table.

Overview.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/ui/model/Filter",
    "sap/ui/model/FilterOperator"
], function (Controller, Filter, FilterOperator) {
    "use strict";

    return Controller.extend("student00.sap.training.dynamicpage.controller.Overview", {
        onInit: function () {},
        onBypassed: function () {},
        _onMasterMatched: function () {},
        getRouter: function () {},
        onSelectionChange: function (oEvent) {},
        onPress: function (oEvent) {},
        _showCarrierDetails: function (sCarrid) {},
        getTable: function () {},
        getTableItems: function () {},
        getFilters: function (aCurrentFilterValues, bAnd) {},
        filterTable: function (aCurrentFilterValues, bAnd) {},
        onSearch: function (oEvent) {},
        onClear: function (oEvent) {},
        onFilterChange: function (oEvent) {},
        onReset : function(oEvent) {
            this.getTableItems().filter([]);
        }

    });
});
```

Figure 235: Implement Function onReset

13. Implement a function with the name `fGetFiltersWithValues` as shown in the following figure, and assign the function as a callback for the `GetFiltersWithValues` event in the `onInit` function. Use the `registerGetFiltersWithValues` function on the `FilterBar` control.

Overview.controller.js

```
37  getRouter: function () {},
40  onSelectionChange: function (oEvent) {},
47  onPress: function (oEvent) {},
55  _showCarrierDetails: function (sCarrid) {},
61  getTable: function () {},
64  getTableItems: function () {},
67  getFilters: function (aCurrentFilterValues, bAnd) {},
79  filterTable: function (aCurrentFilterValues, bAnd) {},
82  onSearch: function (oEvent) {},
103 onClear: function (oEvent) {},
112 onFilterChange: function (oEvent) {},
115 onReset : function(oEvent) {},
118 fGetFiltersWithValues: function () {
119     var i;
120     var oControl;
121     var aFilters = this.getFilterGroupItems();
122     var aFiltersWithValue = [];
123     for (i = 0; i < aFilters.length; i++) {
124         oControl = this.determineControlByFilterItem(aFilters[i]);
125         if (oControl && oControl.getValue && oControl.getValue()) {
126             aFiltersWithValue.push(aFilters[i]);
127         }
128     }
129     return aFiltersWithValue;
130 }
131 });
132 });
```

Figure 236: Implement Function

14. Extend the `onSearch` function as shown and save the changes.

Overview.controller.js

```
55  _showCarrierDetails: function (sCarrid) {},
61  getTable: function () {},
64  getTableItems: function () {},
67  getFilters: function (aCurrentFilterValues, bAnd) {},
79  filterTable: function (aCurrentFilterValues, bAnd) {},
82  onSearch: function (oEvent) {
83      var aCurrentFilterValues = [];
84      this.aKeys = [];
85      if(oEvent.getParameter("query")) {
86          var sQuery = oEvent.getParameter("query");
87          this.aKeys = ["Carrid", "Carrname"];
88          aCurrentFilterValues.push(sQuery);
89          aCurrentFilterValues.push(sQuery);
90          this.filterTable(aCurrentFilterValues,false) ;
91      } else if(oEvent.getParameter("selectionSet")) {
92          if(this.oCarIdIF.getValue() !== "") {
93              this.aKeys.push("Carrid");
94              aCurrentFilterValues.push(this.oCarIdIF.getValue());
95          }
96          if(this.oCarNameIF.getValue() !== "") {
97              this.aKeys.push("Carrname");
98              aCurrentFilterValues.push(this.oCarNameIF.getValue());
99          }
100         this.filterTable(aCurrentFilterValues, true);
101     }
102 },
```

Figure 238: Extend the onSearch Function

15. Test your application.

Task 2: Show the Number of Carriers Listed in the Carrier Table.

1. Add an event handler to the carrier table located in the `Overview.view.xml` for the event `updateFinished`. Assign an event handler with the name `onUpdateFinished`.

2. Open the controller implementation of the *Overview* and add the following implementation of the `onUpdateFinished` function.

Overview.controller.js

```
], function (Controller, Filter, FilterOperator) {
    "use strict";

    return Controller.extend("student00.sap.training.dynamicpage.controller.Overview
        onInit: function () {},
        onBypassed: function () {},
        _onMasterMatched: function () {},
        getRouter: function () {},
        onSelectionChange: function (oEvent) {},
        onPress: function (oEvent) {},
        _showCarrierDetails: function (sCarrid) {},
        getTable: function () {},
        getTableItems: function () {},
        getFilters: function (aCurrentFilterValues, bAnd) {},
        filterTable: function (aCurrentFilterValues, bAnd) {},
        onSearch: function (oEvent) {},
        onClear: function (oEvent) {},
        onFilterChange: function (oEvent) {},
        onReset : function(oEvent) {},
        fGetFiltersWithValues: function () {},
        onUpdateFinished: function(oEvent) {
            if(this.getTableItems().isLengthFinal()) {
                var iCount = oEvent.getParameter("total");
                var oJSONModel = this.getOwnerComponent().getModel("appView");
                oJSONModel.setProperty("/count", iCount);
                this.getOwnerComponent().setModel(oJSONModel,"appView");
            }
        }
```

Figure 240: onUpdateFinished Implementation

3. Go back to the `Overview.view.xml` and update the binding of property text from the `sap.m.Title` control as shown.

Overview.controller.js | Overview.view.xml

```
<mvc:View xmlns:core="sap.ui.core" xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m" xmlns:f="sap.f" xmlns:fb="sap.ui.comp.filterbar"
    controllerName="student00.sap.training.dynamicpage.controller.Overview" xmlns:html="http://www.w3.org/1999/xhtml">
    <f:DynamicPage id="idDynamicPage" headerExpanded="true" toggleHeaderOnTitleClick="true">
        <f:header></f:header>
        <f:content>
            <Table items="{/ZBC_C_Carrier_TP}" mode="{= ${device>/system/phone} ? 'None' : 'SingleSelectMaster'}"
                selectionChange="onSelectionChange" updateFinished="onUpdateFinished"
                id="idCarrierList">
                <headerToolbar>
                    <Toolbar>
                        <content>
                            <Title text="{i18n>overviewPageTitle} ({appView>/count})" level="H2"/>
                            <ToolbarSpacer/>
                            <SearchField placeholder="{i18n>sfPlaceholder}" search="onSearch" width="15rem"/>
                        </content>
                    </Toolbar>
                </headerToolbar>
                <columns></columns>
                <items></items>
            </Table>
        </f:content>
    </f:DynamicPage>
</mvc:View>
```

Figure 241: Update the Binding of Property Text

4. Test your application.

Task 3: Implement Sorting and Grouping Functionality

An important aspect of implementing the list report floorplan is to provide a capability to sort and group the data in the list. In the next task you add basic and simple sorting and grouping capability to your list report.

1. Replace the `sap.m.Toolbar` control in the `Overview.view.xml` file and use a `sap.m.OverflowToolbar` control instead.

2. Add two `sap.m.OverflowToolbarButton` controls to the `sap.m.OverflowToolbar` control located in the *headerToolbar* aggregation of the `sap.m.Table` control in the *Overview*. Use the following attribute and values.

Table 70: Attributes of the first sap.m.OverflowToolbarButton Control

Attribute	Value
tooltip	Sort
press	onOpenViewSettings
type	Transparent
icon	sap-icon://sort

Table 71: Attributes of the second sap.m.OverflowToolbarButton Control

Attribute	Value
tooltip	Group
press	onOpenViewSettings
type	Transparent
icon	sap-icon://group-2

Note:
Include the tooltip attribute in the i18n property file. Use the key **`btnSort`** for the sort button tooltip text and the key **`btnGroup`** for the group button. Use the i18n editor of the *SAP Web IDE* as shown in previous steps.

3. Implement a dialog where the user is able to select if the content of the table should be sorted and/or grouped by the ID or the name of a carrier. Implement the dialog as a fragment. Add the implementation in a file with the name **`ViewSettingsDialog.fragment.xml`**. The file should be located in the view folder of your project. Use the `sap.m.ViewSettingsDialog` implementation of SAPUI5. Add the following implementation to the `ViewSettingsDialog.fragment.xml` file.

ViewSettingsDialog.fragment.xml

```
<core:FragmentDefinition xmlns="sap.m" xmlns:core="sap.ui.core">
    <ViewSettingsDialog id="viewSettingsDialog" confirm="onConfirmViewSettingsDialog">
        <sortItems>
            <ViewSettingsItem text="{i18n>filterCarId}" key="Carrid"/>
            <ViewSettingsItem text="{i18n>filterCarname}" key="Carrname"/>
        </sortItems>
        <groupItems>
            <ViewSettingsItem text="{i18n>filterCarId}" key="Carrid"/>
            <ViewSettingsItem text="{i18n>filterCarname}" key="Carrname"/>
        </groupItems>
    </ViewSettingsDialog>
</core:FragmentDefinition>
```

Figure 245: Implementation Fragment

4. Implement the `onOpenViewSettings` function inside the controller of the *Overview* in the figure.

Figure 249: Function onOpenViewSettings

5. Implement the function `onConfirmViewSettingsDialog` in the controller of the *Overview* as shown.

Figure 250: Function onConfirmViewSettingsDialog

6. Test your application.

Unit 12
Solution 12

Implement a List Report

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, step procedures and naming of fields and buttons may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

Business Example

In this exercise you learn how to extend a standard SAP Fiori application to create a list report floorplan.

This exercise uses the flexiblecolumnlayout project created in the previous exercise..

Task 1: Create a New Project

Export the project `flexiblecolumnlayout` that you created in the previous project and import the project with the name *listreport* into the workspace of your *SAP Web IDE Full-Stack*.

1. Log on to the SAP Web IDE Full-Stack, and export the project `flexiblecolumnlayout` that you created in one of the previous exercises.
   a) Select the project `flexiblecolumnlayout` in the workspace and choose *Export* from the context menu.
   b) Find the file downloaded in the footer area of your browser.

Figure 208: Zip File

2. Import the content of the file `flexiblecolumnlayout.zip` as a project with the name `listreport`.
   a) In the workspace folder, choose *Import* → *File or Project* from the context menu.
   b) Use the *Browse* button to choose the exported file and change the file name in the *Import* field to **`/listreport`** and choose *OK*.
   c) On successful import find the newly imported project with the given name.
3. Add the attribute id to the table control in the Overview-view and assign the value idCarrierList to the newly added attribute. Implement a search functionality to search the table of carriers, as described in the SAP Fiori guidelines. Add an *sap.m.SearchField*

control and place the control in the *sap.m.Toolbar*. Add an *sap.m.ToolbarSpacer* control between the existing *sap.m.Title* control and the newly added *sap.m.SearchField* control. Implement the `onSearch` function. Use the listed properties as attributes for the *SearchField* control. The search functionality returns a list of carriers where the `carrid` or the `carrname` matches the specified value.

Table 63: SearchField Control Properties

Attribute	Value
placeholder	Filter
search	onSearch
width	15rem

Note:
Implement the search functionality in a reusable way as we use the same search and filter aspects in a later step.

Note:
Use the i18n model for the placeholder text. Add the value **`Filter`** with the key **`sfPlaceholder`** and type **`#XTXT`** into the i18n.property file.

a) Open the file `Overview.view.xml` and add an *sap.m.ToolbarSpacer* control immediately after the `sap.m.Title` control in the `sap.m.Toolbar`.

*Overview.view.xml ×

```
1  <mvc:View xmlns:core="sap.ui.core" xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m" xmlns:f="sap.f"
2      controllerName="student00.sap.training.dynamicpage.controller.Overview" xmlns:html="http://www.w3.org/1999/xhtml">
3      <f:DynamicPage id="idDynamicPage" headerExpanded="true" toggleHeaderOnTitleClick="true">
4          <f:content>
5              <Table items="{/ZBC_C_Carrier_TP}" mode="{= ${device>/system/phone} ? 'None' : 'SingleSelectMaster'}"
6                  selectionChange="onSelectionChange" id="idCarrierList">
7                  <headerToolbar>
8                      <Toolbar>
9                          <content>
10                             <Title text="{i18n>overviewPageTitle}" level="H2"/>
11                             <ToolbarSpacer/>
12                         </content>
13                     </Toolbar>
```

Figure 209: Add an sap.m.ToolbarSpacer

b) Add an `sap.m.SearchField` to the existing `sap.m.Toolbar` control and insert the control immediately after the `sap.m.ToolbarSpacer` control. Add the listed attributes to the control and assign the given values for the attribute search and width. Assign an empty string to the placeholder attribute.

Figure 210: Placeholder Attribute

c) Put the cursor in the empty string at the level of the placeholder attribute and choose *Create i18n String* from the context menu.

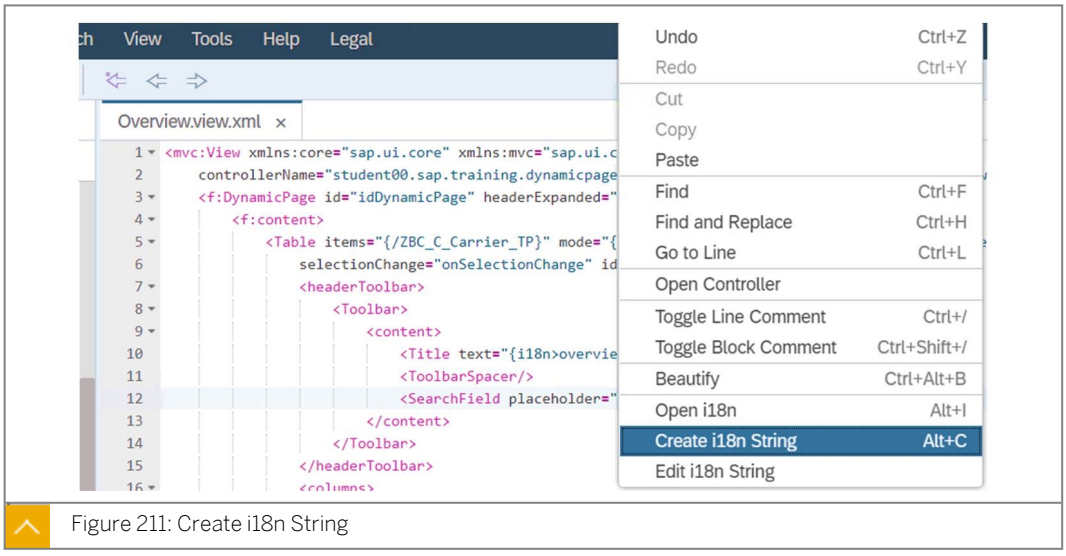


Figure 211: Create i18n String

d) In the new string dialog, enter **`sfPlaceholder`** in the *String Key* field, **`Filter`** in the *Value* field and choose the entry **`#XTXT`** from the *Type* list. Choose *OK* to confirm the dialog.

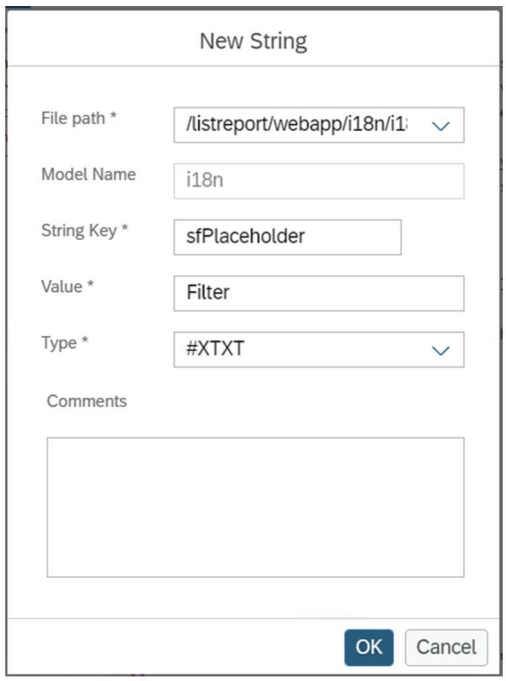


Figure 212: Enter Details

e) Check your *sap.m.SearchField* control appears as shown.

Figure 213: sap.m.SearchField control

f) Save your work.

g) Open the file `Overview.controller.js` located in the controller folder of your project.

h) Add a reference to the `sap.ui.model.Filter` and to `sap.ui.model.FilterOperator`. Follow the general naming conventions for the Asynchronous Module Definition (AMD) syntax.

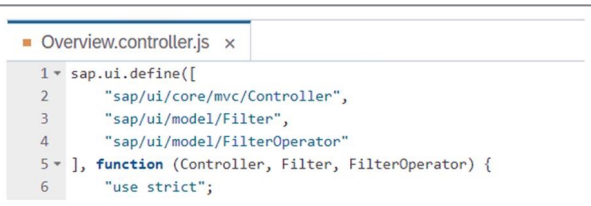


Figure 214: Add References

i) Implement a function named `getTable` that returns the reference to the *sap.m.Table* control with the ID **`idCarrierList`**.

Overview.controller.js

```
8   return Controller.extend("student00.sap.training.dynamicpage.controller.Overview", {
9       onInit: function () {},
27
28      onBypassed: function () {},
31
32      _onMasterMatched: function () {},
35
36      getRouter: function () {},
39
40      onSelectionChange: function (oEvent) {},
47
48      onPress: function (oEvent) {},
56
57      _showCarrierDetails: function (sCarrid) {},
63
64      getTable: function () {
65          return this.getView().byId("idCarrierList");
66      },
```

Figure 215: Function to get Carrier List

j) Implement a function `getTableItems`. The function returns the items aggregation of the table returned by the `getTable` function.

Overview.controller.js

```
8   return Controller.extend("student00.sap.training.dynamicpage.controller.Overview", {
9       onInit: function () {},
27
28      onBypassed: function () {},
31
32      _onMasterMatched: function () {},
35
36      getRouter: function () {},
39
40      onSelectionChange: function (oEvent) {},
47
48      onPress: function (oEvent) {},
56
57      _showCarrierDetails: function (sCarrid) {},
63
64      getTable: function () {
65          return this.getView().byId("idCarrierList");
66      },
67      getTableItems: function () {
68          return this.getTable().getBinding("items");
69      },
```

Figure 216: Function getTableItems

k) Implement a function named `getFilters`. The function gets two parameters. The first parameter is an array containing the filter values and the second holds the value to configure if the filters should be combined with an AND or OR instruction. Implement the function as shown below.

Overview.controller.js ×

```
3     "sap/ui/model/Filter",
4     "sap/ui/model/FilterOperator"
5 ], function (Controller, Filter, FilterOperator) {
6     "use strict";
7
8     return Controller.extend("student00.sap.training.dynamicpage.controller.Overview", {
9         onInit: function () {},
31        onBypassed: function () {},
34        _onMasterMatched: function () {},
37        getRouter: function () {},
40        onSelectionChange: function (oEvent) {},
47        onPress: function (oEvent) {},
55        _showCarrierDetails: function (sCarrid) {},
61        getTable: function () {},
64        getTableItems: function () {},
67        getFilters: function (aCurrentFilterValues, bAnd) {
68            this.aFilters = [];
69            this.aFilters = this.aKeys.map(function (sCriteria, i) {
70                return new Filter(sCriteria, FilterOperator.Contains, aCurrentFilterValues[i]);
71            });
72            return new Filter({
73                filters: this.aFilters,
74                and: bAnd
75            });
76        },
```

Figure 217: getFilters Function

l) Implement a function `filterTable`. The function gets two parameters. The first parameter is the array containing the input for search and the second defines if the search filters are combined with AND or OR.

Overview.controller.js ×

```
3     "sap/ui/model/Filter",
4     "sap/ui/model/FilterOperator"
5 ], function (Controller, Filter, FilterOperator) {
6     "use strict";
7
8     return Controller.extend("student00.sap.training.dynamicpage.controller.Overview", {
9         onInit: function () {},
31        onBypassed: function () {},
34        _onMasterMatched: function () {},
37        getRouter: function () {},
40        onSelectionChange: function (oEvent) {},
47        onPress: function (oEvent) {},
55        _showCarrierDetails: function (sCarrid) {},
61        getTable: function () {},
64        getTableItems: function () {},
67        getFilters: function (aCurrentFilterValues, bAnd) {},
77        filterTable: function (aCurrentFilterValues, bAnd) {
78            this.getTableItems().filter(this.getFilters(aCurrentFilterValues,bAnd));
79        },
```

Figure 218: filterTable Function

m) Implement the function `onSearch`.

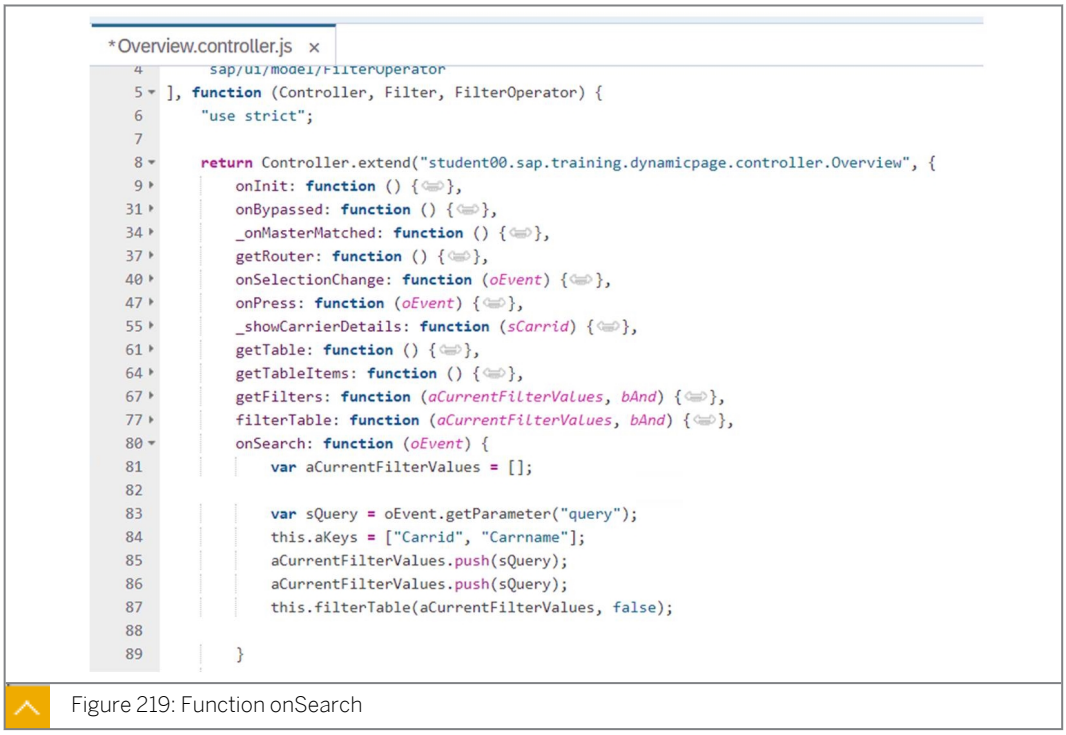


Figure 219: Function onSearch

n) Save your work.

4. Test the search implementation.

a) Select the `index.html` file located in the webapp folder of your project and choose *Run → Run as Web Application* from the context menu.

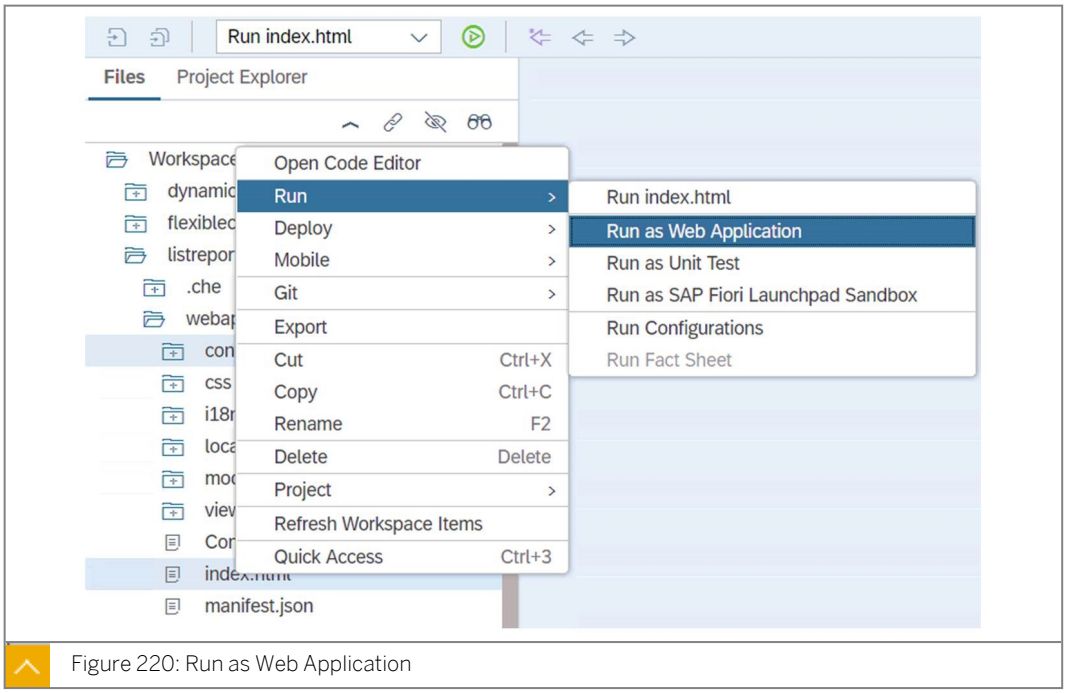


Figure 220: Run as Web Application

b) If prompted to enter your front-end-server credentials, use **`Train-##`**, where ## is your user group number with your password and choose *Sign In*.

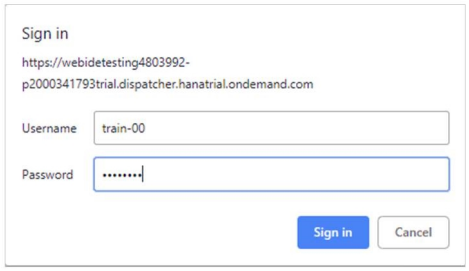


Figure 221: Sign In

c) Enter a value in the *search* field and press *Enter*.

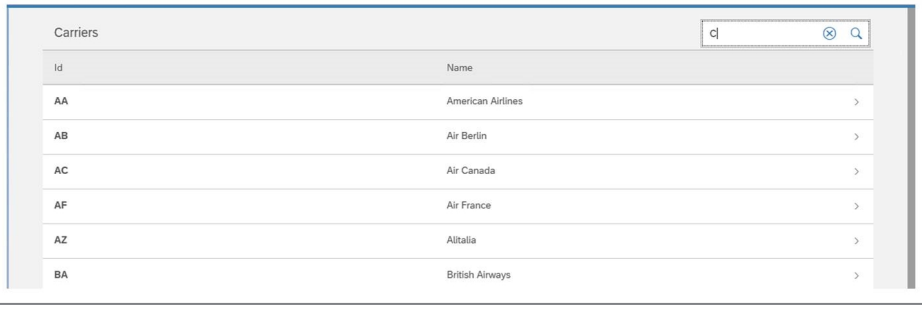


Figure 222: Enter a Value in the Search Field

d) After the filter request is processed, find the filtered list of carriers.

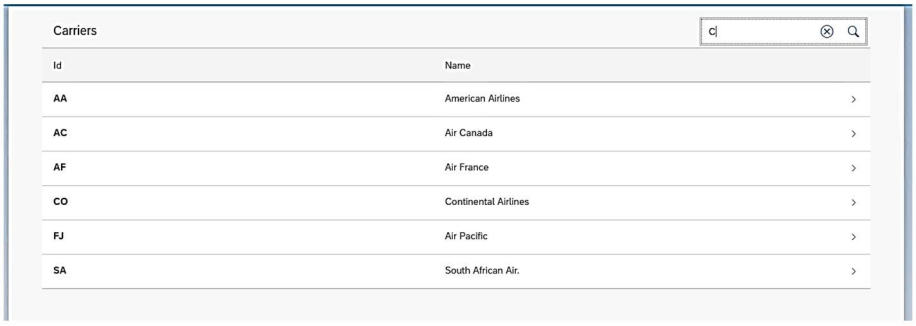


Figure 223: List of Carriers

e) Close the browser tab.

5. Add a UI control of type `sap.f.DynamicPageHeader` to the header aggregation of the existing `sap.f.DynamicPage` control located in the Overview.view.xml file. Assign the following attribute and value to the newly added `sap.f.DynamicPageHeader` control.

Table 64: sap.f.DynamicPageHeader attributes and values

Attribute	Value
pinnable	false

a) Open the file `Overview.view.xml`. The file is located in the view folder of your project.

b) Add the header aggregation to the existing `sap.f.DynamicPage` control.

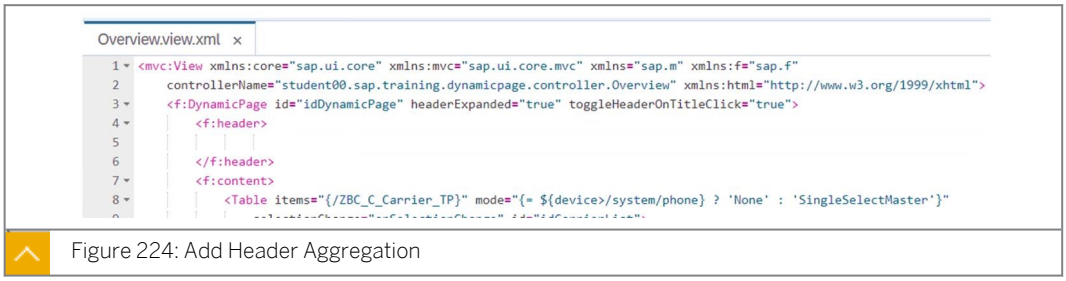


Figure 224: Add Header Aggregation

c) Add a control of type `sap.f.DynamicPageHeader` to the header aggregation with the attribute and value listed in the table.

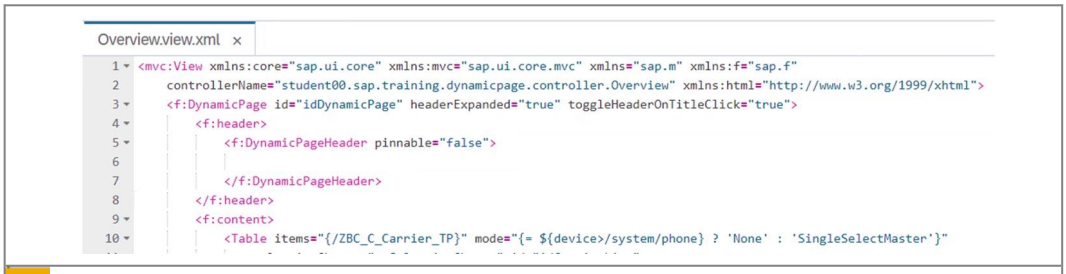


Figure 225: Add to Header Aggregation

6. Add a UI control of type `sap.ui.comp.filterbar.FilterBar` to the content aggregation of the `sap.f.DynamicPageHeader` control. Use an XML namespace with alias **`fb`** to assign the SAPUI5 `sap.ui.comp.filterbar`. Add the listed attribute and values to the newly added `sap.ui.comp.filterbar.FilterBar` control.

Table 65: sap.ui.comp.filterbar.FilterBar control attribute and values

Attribute	Value
id	idFilterBar
reset	onReset
search	onSearch
clear	onClear
cancel	onCancel
filtersDialogClose	onFiltersDialogClose
useToolbar	true
showGoOnFB	true
showRestoreButton	true
showClearButton	true

a) Add the XML namespace alias **`fb`** to the *View* control and assign the SAPUI5 namespace `sap.ui.comp.filterbar`.

Overview.view.xml

```
<mvc:View xmlns:core="sap.ui.core" xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m" xmlns:f="sap.f"
    xmlns:fb="sap.ui.comp.filterbar"
    controllerName="student00.sap.training.dynamicpage.controller.Overview" xmlns:html="http://www.w3.org/1999/xhtml">
    <f:DynamicPage id="idDynamicPage" headerExpanded="true" toggleHeaderOnTitleClick="true">
```

Figure 226: fb Filterbar

b) Add the content aggregation to the newly added `sap.f.DynamicPageHeader` control.

Overview.view.xml

```
<mvc:View xmlns:core="sap.ui.core" xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m" xmlns:f="sap.f"
    xmlns:fb="sap.ui.comp.filterbar"
    controllerName="student00.sap.training.dynamicpage.controller.Overview" xmlns:html="http://www.w3.org/1999/xhtml">
    <f:DynamicPage id="idDynamicPage" headerExpanded="true" toggleHeaderOnTitleClick="true">
        <f:header>
            <f:DynamicPageHeader pinnable="false">
                <f:content>

                </f:content>
            </f:DynamicPageHeader>
        </f:header>
```

Figure 227: Add content aggregation to the Dynamic Page Header

c) Add the `sap.ui.comp.filterbar.FilterBar` control to the content aggregation of the `sap.f.DynamicPageHeader` control and add the attribute and values to the UI control.

Overview.view.xml

```
<mvc:View xmlns:core="sap.ui.core" xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m" xmlns:f="sap.f"
    xmlns:fb="sap.ui.comp.filterbar"
    controllerName="student00.sap.training.dynamicpage.controller.Overview" xmlns:html="http://www.w3.org/1999/xhtml">
    <f:DynamicPage id="idDynamicPage" headerExpanded="true" toggleHeaderOnTitleClick="true">
        <f:header>
            <f:DynamicPageHeader pinnable="false">
                <f:content>
                    <fb:FilterBar reset="onReset" search="onSearch" clear="onClear" cancel="onCancel"
                        filtersDialogClosed="onFiltersDialogClosed" useToolbar="true" showGoOnFB="true"
                        showRestoreButton="true" showClearButton="true" id="idFilterBar">

                    </fb:FilterBar>
                </f:content>
            </f:DynamicPageHeader>
        </f:header>
```

Figure 228: Add Attributes and Values to the UI Control

d) Save your changes.

7. Add two `sap.ui.comp.filterbar.FilterGroupItem` controls to the *filterItems* aggregation of the `sap.ui.comp.filterbar.FilterBar` with the following attributes and values.

Table 66: sap.ui.comp.filterbar.FilterGroupItem Control Attributes and Values

Arttribute	Value
groupName	G1
visibleInFilterBar	true
groupTitle	carridGroup
name	A
label	Carrier

Note:
Use the i18n *ResourceModel* and add the key *filterCarId* to the i18n.properties file. Assign the value **`Carrier`** to the key and use this key instead of the hardwired text.

Table 67: sap.ui.comp.filterbar.FilterGroupItem Control Attributes and Values

Attribute	Value
groupName	G2
visibleInFilterBar	true
groupTitle	carrnameGroup
name	B
label	Carrier name

a) Add `filterGroupItems` aggregation to the `sap.ui.comp.filterbar.FilterBar` control.

Overview.view.xml

```
<mvc:View xmlns:core="sap.ui.core" xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m" xmlns:f="sap.f" xmlns:fb="sap.ui.comp.filterbar"
    controllerName="student00.sap.training.dynamicpage.controller.Overview" xmlns:html="http://www.w3.org/1999/xhtml">
    <f:DynamicPage id="idDynamicPage" headerExpanded="true" toggleHeaderOnTitleClick="true">
        <f:header>
            <f:DynamicPageHeader pinnable="false">
                <f:content>
                    <fb:FilterBar reset="onReset" search="onSearch" clear="onClear" cancel="onCancel" filtersDialogClosed="onFiltersDialogClosed"
                        useToolbar="true" showGoOnFB="true" showRestoreButton="true" showClearButton="true" id="idFilterBar">
                        <fb:filterGroupItems>

                        </fb:filterGroupItems>
                    </fb:FilterBar>
                </f:content>
            </f:DynamicPageHeader>
        </f:header>
```

Figure 229: Add filterGroupItems

b) Add two `sap.ui.comp.filterbar.FilterGroupItem` controls to the *filterGroupItems* aggregation. Use the i18n tools to add the label attribute (as shown in the previous task step 3c and 3d).

Overview.view.xml

```
<mvc:View xmlns:core="sap.ui.core" xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m" xmlns:f="sap.f" xmlns:fb="sap.ui.comp.filterbar"
    controllerName="student00.sap.training.dynamicpage.controller.Overview" xmlns:html="http://www.w3.org/1999/xhtml">
    <f:DynamicPage id="idDynamicPage" headerExpanded="true" toggleHeaderOnTitleClick="true">
        <f:header>
            <f:DynamicPageHeader pinnable="false">
                <f:content>
                    <fb:FilterBar reset="onReset" search="onSearch" clear="onClear" cancel="onCancel" filtersDialogClosed="onFiltersDialogClosed"
                        useToolbar="true" showGoOnFB="true" showRestoreButton="true" showClearButton="true" id="idFilterBar">
                        <fb:filterGroupItems>
                            <fb:FilterGroupItem groupName="G1" visibleInFilterBar="true" groupTitle="carridGroup" name="A"
                                label="{i18n>filterCarId}">

                            </fb:FilterGroupItem>
                            <fb:FilterGroupItem groupName="G2"  visibleInFilterBar="true" groupTitle="carrnameGroup" name="B"
                                label="{i18n>filterCarname}">

                            </fb:FilterGroupItem>
                        </fb:filterGroupItems>
                    </fb:FilterBar>
                </f:content>
            </f:DynamicPageHeader>
```

Figure 230: Add Controls

c) Save your work.

8. Add a control of type `sap.m.Input` to each control aggregation of the `sap.ui.comp.filter.FilterGroupItem` controls. Assign the listed attributes and values to the controls.

Table 68: sap.m.Input Control 1 Attributes and Values

Attribute	Value
placeholder	{i18n>filterCarId}
liveChange	onFilterChange
id	idFilterCarrid

Table 69: sap.m.Input Control 2 Attributes and Values

Attribute	Value
placeholder	{i18n>filterCarname}
liveChange	onFilterChange
ID	idFilterCarrname

Add a *sap.m.Input* control to each of the *sap.ui.comp.filter.FilterGroupItem* controls and assign the listed attribute and values.

Overview.view.xml

```
<mvc:View xmlns:core="sap.ui.core" xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m" xmlns:f="sap.f" xmlns:fb="sap.ui.comp.filterbar"
    controllerName="student00.sap.training.dynamicpage.controller.Overview" xmlns:html="http://www.w3.org/1999/xhtml">
    <f:DynamicPage id="idDynamicPage" headerExpanded="true" toggleHeaderOnTitleClick="true">
        <f:header>
            <f:DynamicPageHeader pinnable="false">
                <f:content>
                    <fb:FilterBar reset="onReset" search="onSearch" clear="onClear" cancel="onCancel" filtersDialogClosed="onFiltersDialogClosed"
                        useToolbar="true" showGoOnFB="true" showRestoreButton="true" showClearButton="true" id="idFilterBar">
                        <fb:filterGroupItems>
                            <fb:FilterGroupItem groupName="G1" visibleInFilterBar="true" groupTitle="carridGroup" name="A"
                                label="{i18n>filterCarId}">
                                <fb:control>
                                    <Input id="idFilterCarrid" placeholder="{i18n>filterCarId}" liveChange="onFilterChange"/>
                                </fb:control>
                            </fb:FilterGroupItem>
                            <fb:FilterGroupItem groupName="G2"  visibleInFilterBar="true" groupTitle="carrnameGroup" name="B"
                                label="{i18n>filterCarname}">
                                <fb:control>
                                    <Input id="idFilterCarrname" placeholder="{i18n>filterCarname}" liveChange="onFilterChange"/>
                                </fb:control>
                            </fb:FilterGroupItem>
                        </fb:filterGroupItems>
                    </fb:FilterBar>
                </f:content>
            </f:DynamicPageHeader>
        </f:header>
```

Figure 231: Add an Input Control

9. Declare a member variable with the name **`oFilterBar`** in the `onInit` function of the *Overview* controller and assign a reference to the *sap.ui.comp.filter.FilterBar* control. Declare another member with the name **`oCarIdIF`** and assign the reference to the UI control with ID *idFilterCarrid*. Declare a member variable with the name **`oCarNameIF`** and assign a reference to the UI control with the ID *idFilterCarrname*.

   a) Open the file `Overview.controller.js`.

   b) Add the mentioned member variables and assign the references.

Overview.controller.js

```
8   return Controller.extend("student00.sap.training.dynamicpage.controller.Overview", {
9       onInit: function () {
10          this._oList = this.byId("idCarrierList");
11          this.getView().addEventDelegate({
12              onBeforeFirstShow: function () {
13                  this.getOwnerComponent().oListSelector.setBoundMasterList(this._oList);
14              }.bind(this)
15          });
16          this.getView().addEventDelegate({
17              onBeforeFirstShow: function () {
18                  this.getOwnerComponent().oListSelector.setBoundMasterList(this._oList);
19              }.bind(this)
20          });
21
22          this.getRouter().getRoute("Overview").attachPatternMatched(this._onMasterMatched, this);
23          this.getRouter().attachBypassed(this.onBypassed, this);
24
25          this.aKeys = ["Carrid", "Carrname"];
26          this.oFilterBar = this.byId("idFilterBar");
27          this.oCarIdIF = this.byId("idFilterCarrid");
28          this.oCarNameIF = this.byId("idFilterCarrname");
29      },
```

Figure 232: Add the Variables and Assign the References

10. Implement the function `onClear` in the controller of the `Overview.controller.js` as shown.

*Overview.controller.js

```
5   ], function (Controller, Filter, FilterOperator) {
6       "use strict";
7
8       return Controller.extend("student00.sap.training.dynamicpage.controller.Overview", {
9           onInit: function () {},
31          onBypassed: function () {},
34          _onMasterMatched: function () {},
37          getRouter: function () {},
40          onSelectionChange: function (oEvent) {},
47          onPress: function (oEvent) {},
55          _showCarrierDetails: function (sCarrid) {},
61          getTable: function () {},
64          getTableItems: function () {},
67          getFilters: function (aCurrentFilterValues, bAnd) {},
79          filterTable: function (aCurrentFilterValues, bAnd) {},
82          onSearch: function (oEvent) {},
103         onClear: function (oEvent) {
104             var oItems = this.oFilterBar.getAllFilterItems(true);
105             for (var i = 0; i < oItems.length; i++) {
106                 var oControl = this.oFilterBar.determineControlByFilterItem(oItems[i]);
107                 if (oControl) {
108                     oControl.setValue("");
109                 }
110             }
111         },
112
113     });
114 });
```

Figure 233: Function onClear

11. Implement a function `onFilterChange` to fire the `onFilterChange` event.

Overview.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/ui/model/Filter",
    "sap/ui/model/FilterOperator"
], function (Controller, Filter, FilterOperator) {
    "use strict";

    return Controller.extend("student00.sap.training.dynamicpage.controller.Overview", {
        onInit: function () {},
        onBypassed: function () {},
        _onMasterMatched: function () {},
        getRouter: function () {},
        onSelectionChange: function (oEvent) {},
        onPress: function (oEvent) {},
        _showCarrierDetails: function (sCarrid) {},
        getTable: function () {},
        getTableItems: function () {},
        getFilters: function (aCurrentFilterValues, bAnd) {},
        filterTable: function (aCurrentFilterValues, bAnd) {},
        onSearch: function (oEvent) {},
        onClear: function (oEvent) {},
        onFilterChange: function (oEvent) {
            this.oFilterBar.fireFilterChange(oEvent);
        }

    });
});
```

Figure 234: Implement Function onFilterChange

12. Implement the function `onReset` and reset the applied filters on the table.

Overview.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/ui/model/Filter",
    "sap/ui/model/FilterOperator"
], function (Controller, Filter, FilterOperator) {
    "use strict";

    return Controller.extend("student00.sap.training.dynamicpage.controller.Overview", {
        onInit: function () {},
        onBypassed: function () {},
        _onMasterMatched: function () {},
        getRouter: function () {},
        onSelectionChange: function (oEvent) {},
        onPress: function (oEvent) {},
        _showCarrierDetails: function (sCarrid) {},
        getTable: function () {},
        getTableItems: function () {},
        getFilters: function (aCurrentFilterValues, bAnd) {},
        filterTable: function (aCurrentFilterValues, bAnd) {},
        onSearch: function (oEvent) {},
        onClear: function (oEvent) {},
        onFilterChange: function (oEvent) {},
        onReset : function(oEvent) {
            this.getTableItems().filter([]);
        }

    });
});
```

Figure 235: Implement Function onReset

a) Add the shown implementation of the function onReset to the controller implementation.

13. Implement a function with the name `fGetFiltersWithValues` as shown in the following figure, and assign the function as a callback for the `GetFiltersWithValues` event in the `onInit` function. Use the `registerGetFiltersWithValues` function on the `FilterBar` control.

Overview.controller.js

```
        _onMasterMatched: function () {},
        getRouter: function () {},
        onSelectionChange: function (oEvent) {},
        onPress: function (oEvent) {},
        _showCarrierDetails: function (sCarrid) {},
        getTable: function () {},
        getTableItems: function () {},
        getFilters: function (aCurrentFilterValues, bAnd) {},
        filterTable: function (aCurrentFilterValues, bAnd) {},
        onSearch: function (oEvent) {},
        onClear: function (oEvent) {},
        onFilterChange: function (oEvent) {},
        onReset : function(oEvent) {},
        fGetFiltersWithValues: function () {
            var i;
            var oControl;
            var aFilters = this.getFilterGroupItems();
            var aFiltersWithValue = [];
            for (i = 0; i < aFilters.length; i++) {
                oControl = this.determineControlByFilterItem(aFilters[i]);
                if (oControl && oControl.getValue && oControl.getValue()) {
                    aFiltersWithValue.push(aFilters[i]);
                }
            }
            return aFiltersWithValue;
        }
    });
});
```

Figure 236: Implement Function

a) Add the shown implementation of the function `fGetFiltersWithValues` to the controller implementation.

b) Go to the `onInit` function and assign the function as a handler for the `GetFiltersWithValues` event using the `registerGetFiltersWithValues` function on the filterbar control.

*Overview.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/ui/model/Filter",
    "sap/ui/model/FilterOperator"
], function (Controller, Filter, FilterOperator) {
    "use strict";

    return Controller.extend("student00.sap.training.dynamicpage.controller.Overview", {
        onInit: function () {
            this._oList = this.byId("idCarrierList");
            this.getView().addEventDelegate(...);
            this.getView().addEventDelegate(...);

            this.getRouter().getRoute("Overview").attachPatternMatched(this._onMasterMatched, this);
            this.getRouter().attachBypassed(this.onBypassed, this);

            this.aKeys = ["Carrid", "Carrname"];
            this.oFilterBar = this.byId("idFilterBar");
            this.oCarIdIF = this.byId("idFilterCarrid");
            this.oCarNameIF = this.byId("idFilterCarrname");
            this.oFilterBar.registerGetFiltersWithValues(this.fGetFiltersWithValues);
        },
```

Figure 237: Assign as a Handler

14. Extend the `onSearch` function as shown and save the changes.

Overview.controller.js

```
55  _showCarrierDetails: function (sCarrid) {},
61  getTable: function () {},
64  getTableItems: function () {},
67  getFilters: function (aCurrentFilterValues, bAnd) {},
79  filterTable: function (aCurrentFilterValues, bAnd) {},
82  onSearch: function (oEvent) {
83      var aCurrentFilterValues = [];
84      this.aKeys = [];
85      if(oEvent.getParameter("query")) {
86          var sQuery = oEvent.getParameter("query");
87          this.aKeys = ["Carrid", "Carrname"];
88          aCurrentFilterValues.push(sQuery);
89          aCurrentFilterValues.push(sQuery);
90          this.filterTable(aCurrentFilterValues,false) ;
91      } else if(oEvent.getParameter("selectionSet")) {
92          if(this.oCarIdIF.getValue() !== "") {
93              this.aKeys.push("Carrid");
94              aCurrentFilterValues.push(this.oCarIdIF.getValue());
95          }
96          if(this.oCarNameIF.getValue() !== "") {
97              this.aKeys.push("Carrname");
98              aCurrentFilterValues.push(this.oCarNameIF.getValue());
99          }
100         this.filterTable(aCurrentFilterValues, true);
101     }
102 },
```

Figure 238: Extend the onSearch Function

15. Test your application.

Task 2: Show the Number of Carriers Listed in the Carrier Table.

1. Add an event handler to the carrier table located in the `Overview.view.xml` for the event `updateFinished`. Assign an event handler with the name `onUpdateFinished`.
   a) Open the file `Overview.view.xml` located in the view folder.
   b) Add the event handler `onUpdateFinished` to the event `updateFinished`.

Overview.view.xml

```
1  <mvc:View xmlns:core="sap.ui.core" xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m" xmlns:f="sap.f" xmlns:fb="sap.ui.comp.filterbar"
2      controllerName="student00.sap.training.dynamicpage.controller.Overview" xmlns:html="http://www.w3.org/1999/xhtml">
3      <f:DynamicPage id="idDynamicPage" headerExpanded="true" toggleHeaderOnTitleClick="true">
4          <f:header>
5              <f:DynamicPageHeader pinnable="false">
6                  <f:content></f:content>
25             </f:DynamicPageHeader>
26         </f:header>
27         <f:content>
28             <Table items="{/ZBC_C_Carrier_TP}" mode="{= ${device>/system/phone} ? 'None' : 'SingleSelectMaster'}"
29                 selectionChange="onSelectionChange" updateFinished="onUpdateFinished"
30                 id="idCarrierList">
31                 <headerToolbar>
```

Figure 239: Add Event Handler

2. Open the controller implementation of the *Overview* and add the following implementation of the `onUpdateFinished` function.

Overview.controller.js

```
], function (Controller, Filter, FilterOperator) {
    "use strict";

    return Controller.extend("student00.sap.training.dynamicpage.controller.Overview", {
        onInit: function () {},
        onBypassed: function () {},
        _onMasterMatched: function () {},
        getRouter: function () {},
        onSelectionChange: function (oEvent) {},
        onPress: function (oEvent) {},
        _showCarrierDetails: function (sCarrid) {},
        getTable: function () {},
        getTableItems: function () {},
        getFilters: function (aCurrentFilterValues, bAnd) {},
        filterTable: function (aCurrentFilterValues, bAnd) {},
        onSearch: function (oEvent) {},
        onClear: function (oEvent) {},
        onFilterChange: function (oEvent) {},
        onReset : function(oEvent) {},
        fGetFiltersWithValues: function () {},
        onUpdateFinished: function(oEvent) {
            if(this.getTableItems().isLengthFinal()) {
                var iCount = oEvent.getParameter("total");
                var oJSONModel = this.getOwnerComponent().getModel("appView");
                oJSONModel.setProperty("/count", iCount);
                this.getOwnerComponent().setModel(oJSONModel,"appView");
            }
        }
```

Figure 240: onUpdateFinished Implementation

a) Open the file `Overview.controller.js` and implement the function `onUpdateFinished` as shown.

3. Go back to the `Overview.view.xml` and update the binding of property text from the `sap.m.Title` control as shown.

Overview.controller.js | Overview.view.xml

```
<mvc:View xmlns:core="sap.ui.core" xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m" xmlns:f="sap.f" xmlns:fb="sap.ui.comp.filterbar"
    controllerName="student00.sap.training.dynamicpage.controller.Overview" xmlns:html="http://www.w3.org/1999/xhtml">
    <f:DynamicPage id="idDynamicPage" headerExpanded="true" toggleHeaderOnTitleClick="true">
        <f:header></f:header>
        <f:content>
            <Table items="{/ZBC_C_Carrier_TP}" mode="{= ${device>/system/phone} ? 'None' : 'SingleSelectMaster'}"
                selectionChange="onSelectionChange" updateFinished="onUpdateFinished"
                id="idCarrierList">
                <headerToolbar>
                    <Toolbar>
                        <content>
                            <Title text="{i18n>overviewPageTitle} ({appView>/count})" level="H2"/>
                            <ToolbarSpacer/>
                            <SearchField placeholder="{i18n>sfPlaceholder}" search="onSearch" width="15rem"/>
                        </content>
                    </Toolbar>
                </headerToolbar>
                <columns></columns>
                <items></items>
            </Table>
        </f:content>
    </f:DynamicPage>
</mvc:View>
```

Figure 241: Update the Binding of Property Text

4. Test your application.

a) To test your application run the `index.html` file as before.

b) Use the filter or the search functions for your table to show in the header-area entity-counter.

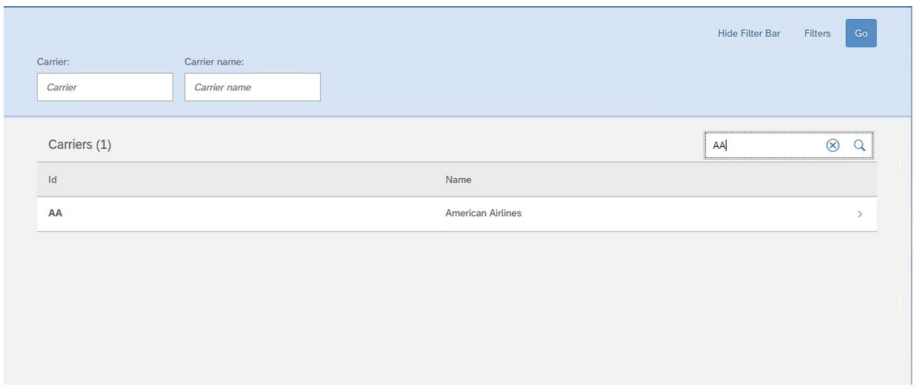


Figure 242: Use Filter and Search Function

c) Close your browser tab.

Task 3: Implement Sorting and Grouping Functionality

An important aspect of implementing the list report floorplan is to provide a capability to sort and group the data in the list. In the next task you add basic and simple sorting and grouping capability to your list report.

1. Replace the `sap.m.Toolbar` control in the `Overview.view.xml` file and use a `sap.m.OverflowToolbar` control instead.

   a) Open the file `Overview.view.xml` located in the view folder of your project.

   b) Replace the `sap.m.Toolbar` with the `sap.m.OverflowToolbar` control.

Figure 243: Replace Toolbar

2. Add two `sap.m.OverflowToolbarButton` controls to the `sap.m.OverflowToolbar` control located in the *headerToolbar* aggregation of the `sap.m.Table` control in the *Overview*. Use the following attribute and values.

Table 70: Attributes of the first sap.m.OverflowToolbarButton Control

Attribute	Value
tooltip	Sort
press	onOpenViewSettings
type	Transparent
icon	sap-icon://sort

Table 71: Attributes of the second sap.m.OverflowToolbarButton Control

Attribute	Value
tooltip	Group
press	onOpenViewSettings
type	Transparent
icon	sap-icon://group-2

Note:
Include the tooltip attribute in the i18n property file. Use the key **`btnSort`** for the sort button tooltip text and the key **`btnGroup`** for the group button. Use the i18n editor of the *SAP Web IDE* as shown in previous steps.

a) Add two UI controls of type `sap.m.OverflowToolbarButton` to the content aggregation of `sap.m.OverflowToolbar` control that is located in the headerToolbar aggregation of `sap.m.Table`. Use the listed attributes.

Figure 244: Add Toolbar Button Controls

b) Use the *i18n* tools of the *SAP Web IDE* to add the *i18n* resource properties for both `tooltip` attributes.

3. Implement a dialog where the user is able to select if the content of the table should be sorted and/or grouped by the ID or the name of a carrier. Implement the dialog as a fragment. Add the implementation in a file with the name **`ViewSettingsDialog.fragment.xml`**. The file should be located in the view folder of your project. Use the `sap.m.ViewSettingsDialog` implementation of SAPUI5. Add the following implementation to the `ViewSettingsDialog.fragment.xml` file.

ViewSettingsDialog.fragment.xml

```
<core:FragmentDefinition xmlns="sap.m" xmlns:core="sap.ui.core">
    <ViewSettingsDialog id="viewSettingsDialog" confirm="onConfirmViewSettingsDialog">
        <sortItems>
            <ViewSettingsItem text="{i18n>filterCarId}" key="Carrid"/>
            <ViewSettingsItem text="{i18n>filterCarname}" key="Carrname"/>
        </sortItems>
        <groupItems>
            <ViewSettingsItem text="{i18n>filterCarId}" key="Carrid"/>
            <ViewSettingsItem text="{i18n>filterCarname}" key="Carrname"/>
        </groupItems>
    </ViewSettingsDialog>
</core:FragmentDefinition>
```

Figure 245: Implementation Fragment

a) Select the view folder of your project and choose *New* → *File* from the context menu.

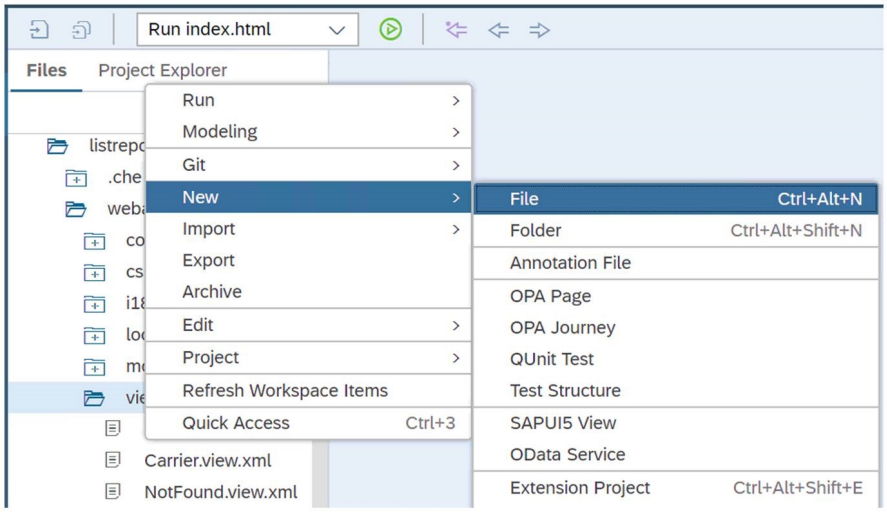


Figure 246: New File

b) Enter the value **`ViewSettingsDialog.fragment.xml`** in the upcoming dialog and choose *OK*.

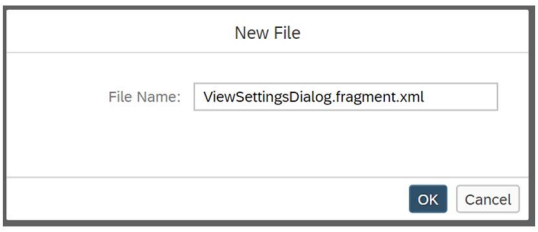


Figure 247: Enter Value

c) Add the previously shown implementation to the newly created fragment.

ViewSettingsDialog.fragment.xml

```
<core:FragmentDefinition xmlns="sap.m" xmlns:core="sap.ui.core">
    <ViewSettingsDialog id="viewSettingsDialog" confirm="onConfirmViewSettingsDialog">
        <sortItems>
            <ViewSettingsItem text="{i18n>filterCarId}" key="Carrid"/>
            <ViewSettingsItem text="{i18n>filterCarname}" key="Carrname"/>
        </sortItems>
        <groupItems>
            <ViewSettingsItem text="{i18n>filterCarId}" key="Carrid"/>
            <ViewSettingsItem text="{i18n>filterCarname}" key="Carrname"/>
        </groupItems>
    </ViewSettingsDialog>
</core:FragmentDefinition>
```

Figure 248: Add the Implementation to the New Fragment

d) Save your work.

4. Implement the `onOpenViewSettings` function inside the controller of the *Overview* in the figure.

Overview.controller.js

```
        onSearch: function (oEvent) {},
        onClear: function (oEvent) {},
        onFilterChange: function (oEvent) {},
        onReset : function(oEvent) {},
        fGetFiltersWithValues: function () {},
        onUpdateFinished: function(oEvent) {},
        onOpenViewSettings : function(oEvent) {
            if (!this._oViewSettingsDialog) {
                this._oViewSettingsDialog = sap.ui.xmlfragment("student00.sap.training.dynamicpage.view.ViewSettingsDialog", this);
                this.getView().addDependent(this._oViewSettingsDialog);
                jQuery.sap.syncStyleClass("sapUiSizeCompact", this.getView(), this._oViewSettingsDialog);
                }
                var sDialogTab = "sort";
                if (oEvent.getSource() instanceof sap.m.Button) {
                    var sButtonId = oEvent.getSource().getId();
                    if (sButtonId.match("group")) {
                        sDialogTab = "group";
                    }
                }
                this._oViewSettingsDialog.open(sDialogTab);
            }
        });
});
```

Figure 249: Function onOpenViewSettings

a) Open the file `Overview.controller.js` located in the controller folder of your project.

b) Add the implementation of the `onOpenViewSettings` function as already shown.

c) Save your work.

5. Implement the function `onConfirmViewSettingsDialog` in the controller of the *Overview* as shown.

*Overview.controller.js ×

```
111 ▸      onFilterChange: function (oEvent) {⋯},
114 ▸      onReset : function(oEvent) {⋯},
117 ▸      fGetFiltersWithValues: function () {⋯},
130 ▸      onUpdateFinished: function(oEvent) {⋯},
138 ▸      onOpenViewSettings : function(oEvent) {⋯},
153 ▾      onConfirmViewSettingsDialog : function (oEvent) {
154            var mParams = oEvent.getParameters(),
155                    sPath,
156                    bDescending,
157                    aSorters = [];
158 ▾          if (mParams.groupItem) {
159                sPath = mParams.groupItem.getKey();
160                bDescending = mParams.groupDescending;
161                aSorters.push(new Sorter(sPath, bDescending, true));
162            }
163 ▾          if(mParams.sortItem) {
164                sPath = mParams.sortItem.getKey();
165                bDescending = mParams.sortDescending;
166                aSorters.push(new Sorter(sPath, bDescending));
167            }
168            this.getTableItems().sort(aSorters);
169        }
170    });
171 });
```

Figure 250: Function onConfirmViewSettingsDialog

a) Open the file `Overview.controller.js` if not already open.

b) Add a reference to the `sap.ui.model.Sorter` implementation to your *Overview* controller.

*Overview.controller.js ×

```
1 ▾ sap.ui.define([
2       "sap/ui/core/mvc/Controller",
3       "sap/ui/model/Filter",
4       "sap/ui/model/FilterOperator",
5       "sap/ui/model/Sorter"
6 ▾ ], function (Controller, Filter, FilterOperator, Sorter) {
7       "use strict";
8
9 ▾     return Controller.extend("student00.sap.training.dynamicpage.controller.Overview", {
```

Figure 251: Add a Reference to the sap.ui.model.Sorter Implementation

c) Add the implementation of *onConfirmViewSettingsDialog* to your controller. Implement the function as shown.

```
*Overview.controller.js ×
111 ▸      onFilterChange: function (oEvent) {…},
114 ▸      onReset : function(oEvent) {…},
117 ▸      fGetFiltersWithValues: function () {…},
130 ▸      onUpdateFinished: function(oEvent) {…},
138 ▸      onOpenViewSettings : function(oEvent) {…},
153 ▾      onConfirmViewSettingsDialog : function (oEvent) {
154            var mParams = oEvent.getParameters(),
155                sPath,
156                bDescending,
157                aSorters = [];
158 ▾          if (mParams.groupItem) {
159                sPath = mParams.groupItem.getKey();
160                bDescending = mParams.groupDescending;
161                aSorters.push(new Sorter(sPath, bDescending, true));
162            }
163 ▾          if(mParams.sortItem) {
164                sPath = mParams.sortItem.getKey();
165                bDescending = mParams.sortDescending;
166                aSorters.push(new Sorter(sPath, bDescending));
167            }
168            this.getTableItems().sort(aSorters);
169        }
170    });
171 });
```

Figure 252: Implement the Function

d) Save your work.

6. Test your application.

a) Restart your application.

b) Choose a configuration to test the sort and group dialog and apply by choosing *OK*.

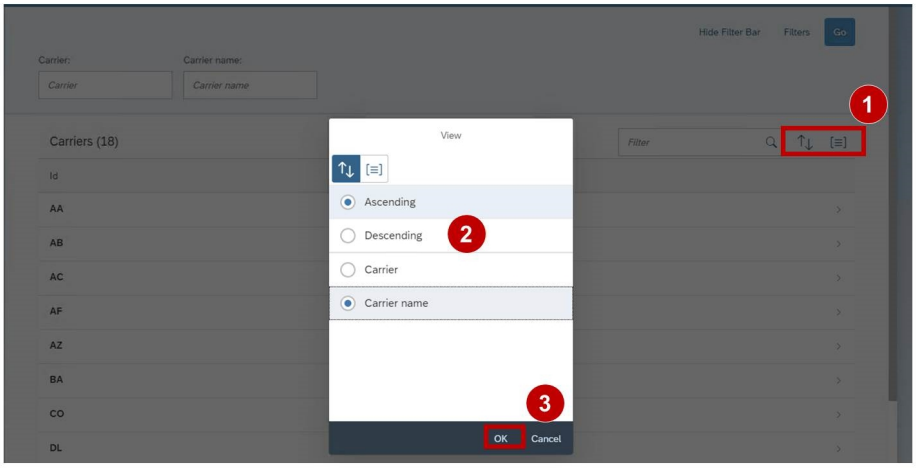


Figure 253: Sort and Group Test

The listed carriers should be sorted or grouped as expected.

c) Close the browser tab of the application after review.

Unit 12
Exercise 13

Implement an Object Page

Object Page

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, step procedures and naming of fields and buttons may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

Business Example

In this exercise, you will learn how to implement the Object Page floorplan.

Task 1: Object Page

1. Export the project *listreport* from the previous exercise and import the project with the name *objectpage* into the workspace of your *SAP Web IDE Full Stack*.
2. Import the content of the file `listreport.zip` as a project with the name **`objectpage`**.

Task 2: Implement the object page floorplan to show the flights and connections as sections.

Note:
For your convenience you will find a template at `S:\Courses\UX410_20\Templates\Ex_ObjectPage_CarrierView.txt`. Be aware that you need to change the path to your controller by inserting your own group number into the path.

1. Change the implementation of the *Carrier* view. Remove the `sap.f.semantic.SemanticPage` control and use the `sap.uxap.ObjectPageLayout` instead. Leave the look and feel of the *Carrier* view unchanged.
2. Add a section to the `ObjectPageLayout` control and display the flights of the selected carrier. The section should look as shown. You can access the flights by using the navigation property to_Flight.

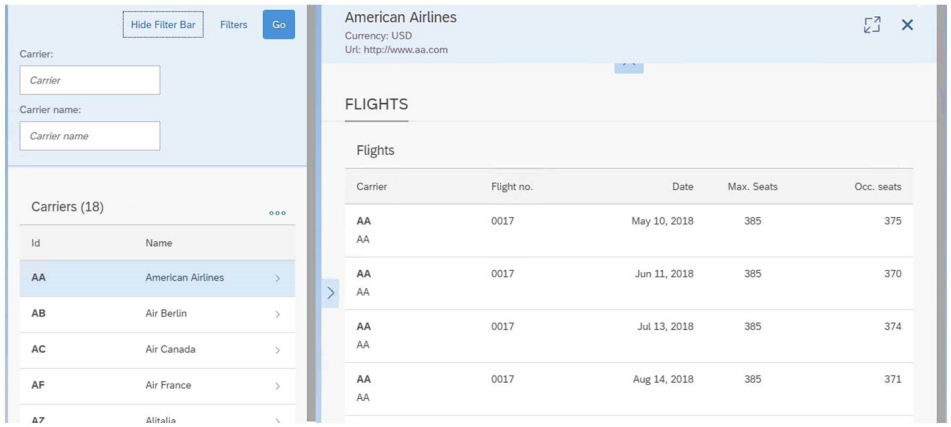


Figure 257: Flights of Carrier

3. Add another section to the object page and display the available connections. You can get the connections with the navigation property `to_Connection`.

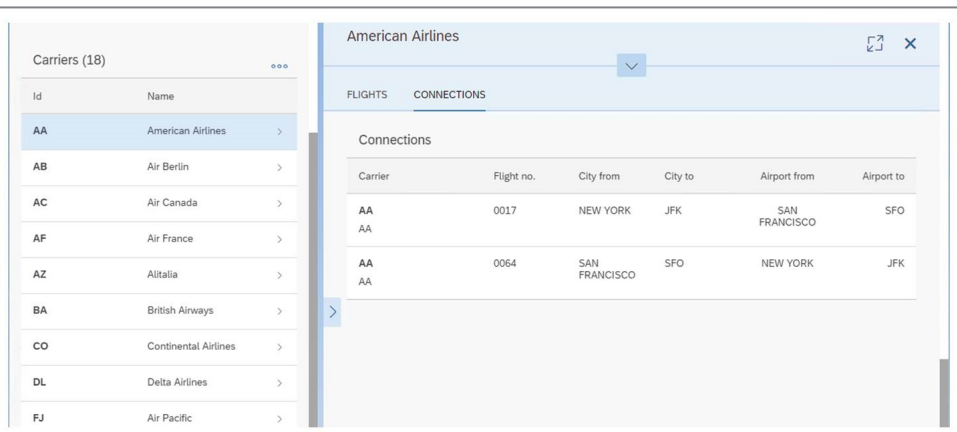


Figure 263: Display available Connections

4. Test your implementation.

Unit 12
Solution 13

Implement an Object Page

Object Page

> Note:
> SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, step procedures and naming of fields and buttons may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

Business Example

In this exercise, you will learn how to implement the Object Page floorplan.

Task 1: Object Page

1. Export the project *listreport* from the previous exercise and import the project with the name *objectpage* into the workspace of your *SAP Web IDE Full Stack*.
   a) Log on to the *SAP Web IDE Full-Stack*.
   b) Choose the project *listreport* from the workspace and select *Export* from the context menu.
   c) When downloaded successfully, the package displays in the bottom of your browser window.

Figure 254: listreport Zip File

2. Import the content of the file `listreport.zip` as a project with the name **`objectpage`**.
   a) Select the workspace folder and from the context menu, choose *Import* → *File or Project*.
   b) Select the exported file using the *Browse* button and change the file name in the *Import* field to **`/objectpage`** and choose *OK*.
   c) When the import finishes successful you will see the newly imported project with the given name.

Task 2: Implement the object page floorplan to show the flights and connections as sections.

Note:
For your convenience you will find a template at `S:\Courses\UX410_20\Templates\Ex_ObjectPage_CarrierView.txt`. Be aware that you need to change the path to your controller by inserting your own group number into the path.

1. Change the implementation of the *Carrier* view. Remove the `sap.f.semantic.SemanticPage` control and use the `sap.uxap.ObjectPageLayout` instead. Leave the look and feel of the *Carrier* view unchanged.
   a) Open the file `Carrier.view.xml` located in the view folder of your project.
   b) Open the template `S:\Courses\UX410_20\Templates\Ex_ObjectPage_CarrierView.txt` using notepad or another editor.
   c) Copy the content of the template file.
   d) Paste the content of the template `S:\Courses\UX410_20\Templates\Ex_ObjectPage_CarrierView.txt`. to the `Carrier.view.xml` in the *SAP Web IDE*.
   e) Change the value of the `controllerName` attribute to use your own group number.
   f) The code should look as shown.

*Carrier.view.xml ×

```
<mvc:View xmlns:core="sap.ui.core" xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m" xmlns:f="sap.f" xmlns:layout="sap.ui.layout"
    xmlns:uxap="sap.uxap" controllerName="student00.sap.training.dynamicpage.controller.Carrier" xmlns:html="http://www.w3.org/1999/xhtml">
    <uxap:ObjectPageLayout id="dynamicPageId" showTitleInHeaderContent="true" alwaysShowContentHeader="false"
        preserveHeaderStateOnScroll="false" headerContentPinnable="true" isChildPage="true" enableLazyLoading="false">
        <uxap:headerTitle>
            <uxap:ObjectPageDynamicHeaderTitle>
                <uxap:expandedHeading>
                    <Title text="{Carrname}"/>
                </uxap:expandedHeading>
                <uxap:snappedHeading>
                    <Title text="{Carrname}"/>
                </uxap:snappedHeading>
                <uxap:expandedContent>
                    <FlexBox alignItems="Start" justifyContent="SpaceBetween">
                        <items>
                            <layout:HorizontalLayout allowWrapping="true">
                                <layout:VerticalLayout class="sapUiMediumMarginEnd">
                                    <ObjectAttribute title="{i18n>currLabelText}" text="{Currcode}"/>
                                    <ObjectAttribute title="{i18n>urlLabelText}" text="{Url}"/>
                                </layout:VerticalLayout>
                            </layout:HorizontalLayout>
                        </items>
                    </FlexBox>
                </uxap:expandedContent>
                <uxap:navigationActions></uxap:navigationActions>
            </uxap:ObjectPageDynamicHeaderTitle>
        </uxap:headerTitle>
    </uxap:ObjectPageLayout>
</mvc:View>
```

Figure 255: Code

*Carrier.view.xml ×

```
<mvc:View
    xmlns:uxap="sap.uxap" controllerName="student00.sap.training.dynamicpage.controller.Carrier" xmlns:html="http://www.w3.org/1999/xhtml"
    <uxap:ObjectPageLayout id="dynamicPageId" showTitleInHeaderContent="true" alwaysShowContentHeader="false"
        preserveHeaderStateOnScroll="false" headerContentPinnable="true" isChildPage="true" enableLazyLoading="false">
        <uxap:headerTitle>
            <uxap:ObjectPageDynamicHeaderTitle>
                <uxap:expandedHeading></uxap:expandedHeading>
                <uxap:snappedHeading></uxap:snappedHeading>
                <uxap:expandedContent></uxap:expandedContent>
                <uxap:navigationActions>
                    <OverflowToolbarButton type="Transparent"
                        icon="sap-icon://full-screen" tooltip="Leave Full Screen Mode" id="idExitFullScreen"
                        visible="{= !${device>/system/phone} &amp;&amp; ${appView>/actionButtonsInfo/midColumn/fullScreen}}"
                        press="onToggleFullScreen"/>
                    <OverflowToolbarButton type="Transparent" icon="sap-icon://full-screen"
                        tooltip="Enter Full Screen Mode"
                        visible="{= !${device>/system/phone} &amp;&amp; !${appView>/actionButtonsInfo/midColumn/fullScreen}}"
                        press="onToggleFullScreen"/>
                    <OverflowToolbarButton type="Transparent"
                    press="onCloseDetailPress" icon="sap-icon://decline" tooltip="Close column"/>
                </uxap:navigationActions>
            </uxap:ObjectPageDynamicHeaderTitle>
        </uxap:headerTitle>
    </uxap:ObjectPageLayout>
</mvc:View>
```

Figure 256: Code

Note:
The implementation shows the controller of group 00. Your implementation should show your group number.

2. Add a section to the `ObjectPageLayout` control and display the flights of the selected carrier. The section should look as shown. You can access the flights by using the navigation property to_Flight.

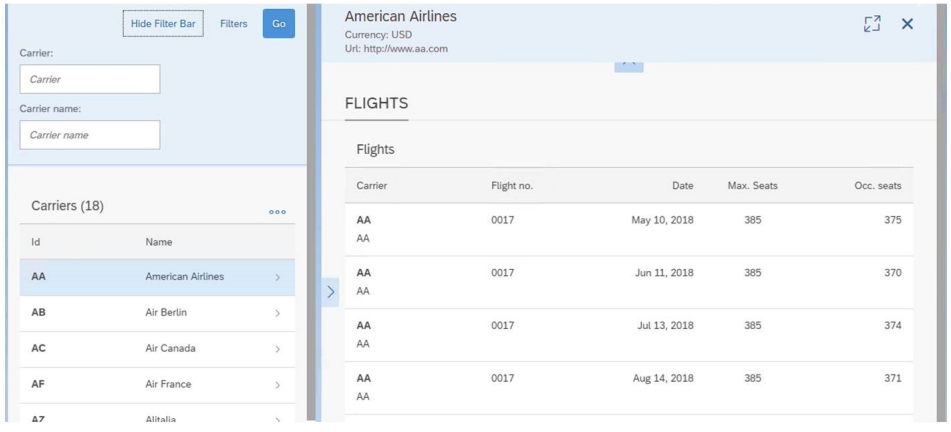


Figure 257: Flights of Carrier

a) Open the `i18n.properties` file located in the `i18n` folder of your project.

b) Add the following key and value pairs to the `i18n.properties` file.

```
i18n.properties ×
1  #XTIT:
2  tblFlights=Flights
3  #XCOL:
4  columnCarrier=Carrier
5  columnFlightNo=Flight no.
6  columnDate=Date
7  columnMaxseats=Max. Seats
8  columnOccseats=Occ. seats
```

Figure 258: Properties File

c) Add the **sections** aggregation to the `sap.uxap.ObjectPageLayout` immediately after the **headerTitle** aggregation. Insert a control of type `sap.uxap.ObjectPageSection` to the **sections** aggregation and add the attribute `title` to it. Assign the property **`i18n>tblFlights`** to the `title` attribute. Add the **subSections** aggregation to the `sap.uxap.ObjectPageSection` control and insert a control of type `sap.uxap.ObjectPageSection` in the **subSections** aggregation. Add the **blocks** aggregation to the newly added `ObjectPageSubSection` control.

```
Carrier.view.xml ×
<mvc:View xmlns:core="sap.ui.core" xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m" xmlns:f="sap.f" xmlns:layout="sap.ui.layout"
    xmlns:uxap="sap.uxap" controllerName="student00.sap.training.dynamicpage.controller.Carrier" xmlns:html="http://www.w3.org/1999/xhtml">
    <uxap:ObjectPageLayout id="dynamicPageId" showTitleInHeaderContent="true" alwaysShowContentHeader="false"
        preserveHeaderStateOnScroll="false" headerContentPinnable="true" isChildPage="true" enableLazyLoading="false">
        <uxap:headerTitle></uxap:headerTitle>
        <uxap:sections>
            <uxap:ObjectPageSection title="{i18n>tblFlights}">
                <uxap:subSections>
                    <uxap:ObjectPageSubSection>
                        <uxap:blocks>

                        </uxap:blocks>
                    </uxap:ObjectPageSubSection>
                </uxap:subSections>
            </uxap:ObjectPageSection>
        </uxap:sections>
    </uxap:ObjectPageLayout>
</mvc:View>
```

Figure 259: Add Sections

d) Add an `sap.m.Table` with the following implementation to the blocks aggregation.

```
*Carrier.view.xml ×
<mvc:View xmlns:core="sap.ui.core" xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m" xmlns:f="sap.f" xmlns:layout="sap.ui.layout"
    xmlns:uxap="sap.uxap" controllerName="student00.sap.training.dynamicpage.controller.Carrier" xmlns:html="http://www.w3.org/1999/xhtml">
    <uxap:ObjectPageLayout id="dynamicPageId" showTitleInHeaderContent="true" alwaysShowContentHeader="false"
        preserveHeaderStateOnScroll="false" headerContentPinnable="true" isChildPage="true" enableLazyLoading="false">
        <uxap:headerTitle></uxap:headerTitle>
        <uxap:sections>
            <uxap:ObjectPageSection title="{i18n>tblFlights}">
                <uxap:subSections>
                    <uxap:ObjectPageSubSection>
                        <uxap:blocks>
                            <Table id="idFlights" items="{ path: 'to_Flight', sorter: { path: 'Carrid' }}" growing="true"
                                growingThreshold="10" visible="true">
                                <headerToolbar>
                                    <OverflowToolbar>
                                        <Title text="{i18n>tblFlights}" level="H2"/>
                                    </OverflowToolbar>
                                </headerToolbar>
                                <columns></columns>
                                <items></items>
                            </Table>
                        </uxap:blocks>
                    </uxap:ObjectPageSubSection>
                </uxap:subSections>
            </uxap:ObjectPageSection>
        </uxap:sections>
```

Figure 260: Add Table

e) Add an implementation of the columns aggregation to the table.

Figure 261: Add Columns

f) Add an implementation to the **items** aggregation of the table.

Figure 262: Items Aggregation

3. Add another section to the object page and display the available connections. You can get the connections with the navigation property `to_Connection`.

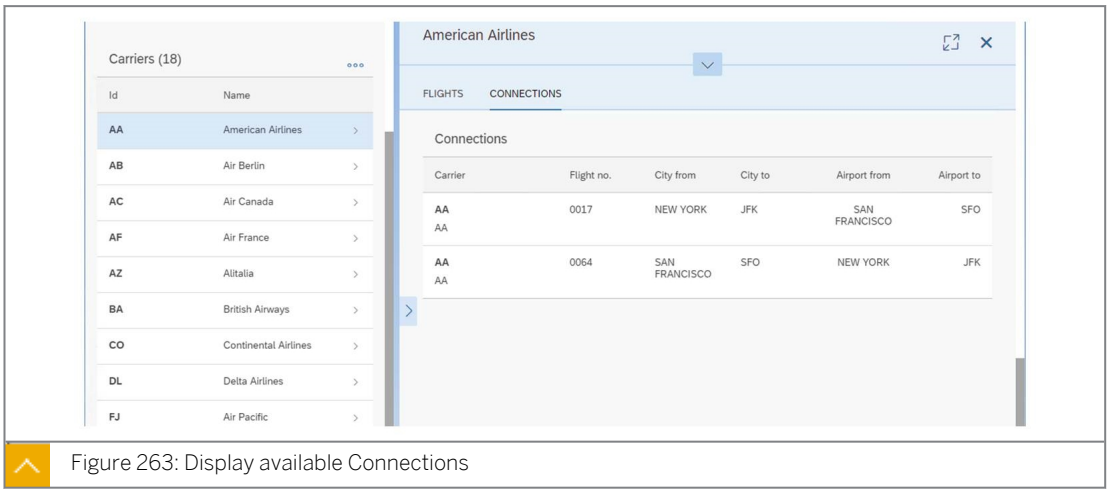


Figure 263: Display available Connections

a) Open the `i18n.properties` file located in the i18n folder of your project and add the following implementation.

i18n.properties

```
#XTIT:
tblFlights=Flights
tblConnections=Connections
#XCOL:
columnCarrier=Carrier
columnFlightNo=Flight no.
columnDate=Date
columnMaxseats=Max. Seats
columnOccseats=Occ. seats
columnCityForm=City from
columnCityTo=City to
columnAirpFrom=Airport from
columnAirpTo=Airport to
```

Figure 264: Open Properties File

b) Insert a control of type `sap.uxap.ObjectPageSection` to the **sections** aggregation immediately after the `sap.uxap.ObjectPageSection` created in the previous step. Add the attribute `title` to it. Assign the property **`{i18n>tblConnections}`** to the **title** attribute. Add the **subSections** aggregation to the `sap.uxap.ObjectPageSection` control and insert a control of type `sap.uxap.ObjectPageSection` into the **subSections** aggregation. Add the **blocks** aggregation to the newly added `ObjectPageSubSection` control.

*Carrier.view.xml

```
<mvc:View xmlns:core="sap.ui.core" xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m" xmlns:f="sap.f" xmlns:layout="sap.ui.layout"
    xmlns:uxap="sap.uxap" controllerName="student00.sap.training.dynamicpage.controller.Carrier" xmlns:html="http://www.w3.org/1999/xhtml">
    <uxap:ObjectPageLayout id="dynamicPageId" showTitleInHeaderContent="true" alwaysShowContentHeader="false"
        preserveHeaderStateOnScroll="false" headerContentPinnable="true" isChildPage="true" enableLazyLoading="false">
        <uxap:headerTitle></uxap:headerTitle>
        <uxap:sections>
            <uxap:ObjectPageSection title="{i18n>tblFlights}"></uxap:ObjectPageSection>
            <uxap:ObjectPageSection title="{i18n>tblConnections}">
            <uxap:subSections>
                <uxap:ObjectPageSubSection>
                    <uxap:blocks>

                    </uxap:blocks>
                </uxap:ObjectPageSubSection>
            </uxap:subSections>
        </uxap:ObjectPageSection>
        </uxap:sections>
    </uxap:ObjectPageLayout>
</mvc:View>
```

Figure 265: Add control

c) Add an `sap.m.Table` control to the **blocks** aggregation of the newly added `sap.uxap.ObjectPageSubSection`. Add the shown attributes and aggregations to the `sap.m.Table` control.

*Carrier.view.xml ×

```
preserveHeaderStateOnScroll="false" headerContentPinnable="true" isChildPage="true" enableLazyLoading="false">
<uxap:headerTitle></uxap:headerTitle>
<uxap:sections>
    <uxap:ObjectPageSection title="{i18n>tblFlights}"></uxap:ObjectPageSection>
        <uxap:ObjectPageSection title="{i18n>tblConnections}">
        <uxap:subSections>
            <uxap:ObjectPageSubSection>
                <uxap:blocks>
                    <Table id="idConnections" items="{ path: 'to_Connection', sorter: { path: 'Connid' }}" growing="true"
                        growingThreshold="10" visible="true">
                        <headerToolbar>
                            <Toolbar>
                                <Title text="Connections" level="H2"/>
                            </Toolbar>
                        </headerToolbar>
                        <columns></columns>
                        <items></items>
                    </Table>
                </uxap:blocks>
            </uxap:ObjectPageSubSection>
        </uxap:subSections>
    </uxap:ObjectPageSection>
</uxap:sections>
</uxap:ObjectPageLayout>
```

Figure 266: Add a Table Control to the Blocks Aggregation

d) Add an implementation to the columns aggregation of the `sap.m.Table` control.

*Carrier.view.xml ×

```
<uxap:ObjectPageSubSection>
    <uxap:blocks>
        <Table id="idConnections" items="{ path: 'to_Connection', sorter: { path: 'Connid' }}" growing="true"
            growingThreshold="10" visible="true">
            <headerToolbar></headerToolbar>
            <columns>
                <Column width="12em">
                    <Text text="{i18n>columnCarrier}"/>
                </Column>
                <Column minScreenWidth="Tablet" demandPopin="true">
                    <Text text="{i18n>columnFlightNo}"/>
                </Column>
                <Column minScreenWidth="Tablet" demandPopin="true">
                    <Text text="{i18n>columnCityFrom}"/>
                </Column>
                <Column minScreenWidth="Tablet" demandPopin="true">
                    <Text text="{i18n>columnCityTo}"/>
                </Column>
                <Column minScreenWidth="Tablet" demandPopin="true" hAlign="Center">
                    <Text text="{i18n>columnAirpFrom}"/>
                </Column>
                <Column hAlign="Right">
                    <Text text="{i18n>columnAirpTo}"/>
                </Column>
            </columns>
```

Figure 267: Implementation

e) Add an implementation of the items aggregation to the table.

*Carrier.view.xml ×

```
preserveHeaderStateOnScroll="false" headerContentPinnable="true" isChildPage="true" enableLazyLoading="false">
<uxap:headerTitle></uxap:headerTitle>
<uxap:sections>
    <uxap:ObjectPageSection title="{i18n>tblFlights}"></uxap:ObjectPageSection>
        <uxap:ObjectPageSection title="{i18n>tblConnections}">
        <uxap:subSections>
            <uxap:ObjectPageSubSection>
                <uxap:blocks>
                    <Table id="idConnections" items="{ path: 'to_Connection', sorter: { path: 'Connid' }}" growing="true"
                        growingThreshold="10" visible="true">
                        <headerToolbar></headerToolbar>
                        <columns></columns>
                        <items>
                            <ColumnListItem>
                                <cells>
                                    <ObjectIdentifier title="{Carrid}" text="{Carrid}"/>
                                    <Text text="{Connid}"/>
                                    <Text text="{Cityfrom}"/>
                                    <Text text="{Airpfrom}"/>
                                    <Text text="{Cityto}"/>
                                    <Text text="{Airpto}"/>
                                </cells>
                            </ColumnListItem>
                        </items>
                    </Table>
```

Figure 268: Add items Aggregation to Table

4. Test your implementation.

a) Select the `index.html` file located in the webapp folder of your project and choose from the context menu *Run → Run as Web Application*.

b) If you are asked to enter your front-end server credentials input the user **`Train-##`** with your password and choose *Sign in*.

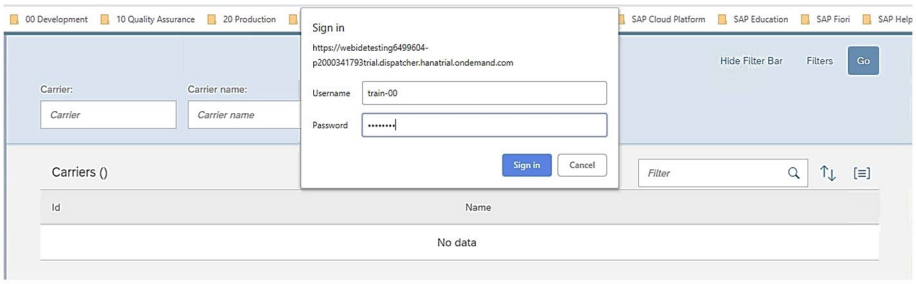


Figure 269: Enter Credentials

c) Select a carrier from the list of carriers.

d) Explore your implementation by selecting the CONNECTIONS tab to display the available list of connections.

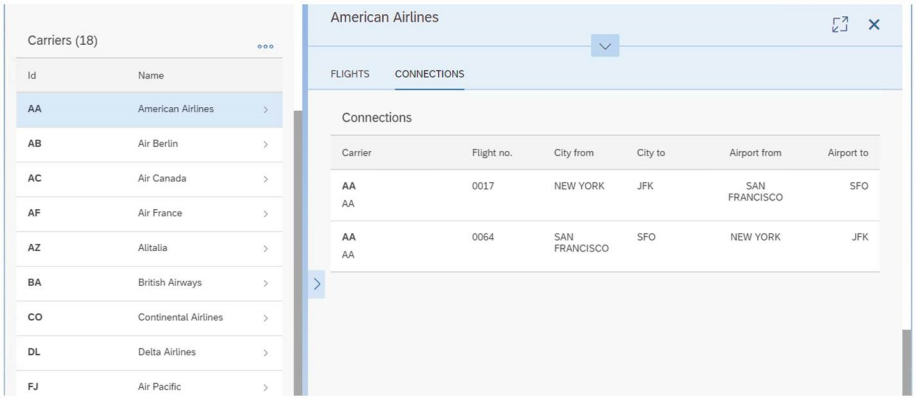


Figure 270: Connections Tab

Note:
There may be carriers in the list with neither flights nor connections. In these cases, there are no result in the tables. Try another carrier.

Unit 13
Exercise 14

Implement and Extend an Extension Point

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, step procedures and naming of fields and buttons may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

In this exercise you will work with SAP Fiori Extension techniques. The basis for this exercise is the implementation of the exercise **Object Page**.

Extend your deployed application *ZUX410_APP## (system is FSD)*.

1. Open the result of the exercise **Object Page**. Add the ID `idCarrierDetails` to the *sap.m.FlexBox* control located in the *sap.uxap.ObjectPageLayout* control.

   Note:
   Create a new separate project by using the export and import function of the SAP Web IDE. Export your objectpage project and import it with the new name of **`objectPageWithExtension`**.

2. Add an *sap.ui.core.ExtensionPoint* with the name `ux410_extension` immediately after the *sap.m.FlexBox* control with the new ID `idCarrierDetails`.

3. Add a controller `hook` function with the name `extUX410HookFunction` to the controller of view *Carrier*. Call, if it exists, the `hook` function at the end of the `onInit` function of the controller. Add the following documentation before the invocation.

   Table 72: extUX410HookFunction Properties and Documentation

JSDoc property	Documentation
@ControllerHook	This hook method can be used to add some extra functionality
@callback	student##.sap.training.dynamicpage.controller.Carrier~extUX410HookFunction
@return	{void}

4. Deploy your application.

Table 73: Application Properties and Values

Property	Value
Application name	ZUX410_App##
Description	Flight app
System	FSD_100 - FSD
Package	ZTRAIN_##
Transport request	Choose the transport request provide by your teacher.

Extend the Deployed Application.

1. Extend your deployed application *ZUX410_APP##* (system is FSD). Create an extension project with the name **`ZUX410_APP##Extension`**. Hide the UI control with ID `idCarrierDetails` by using the extension pane. Create a view extension for the extension point `ux410_extension` using the extension pane. Add the code provided in the example for the extension. Test your extended application.

Figure 275: Extension Code

2. Add an i18n-resource customizing extension to your project and enter the new text properties listed. Replace the hard coded text literals located in the view extension. Test your application.

Table 74: Text Properties

Key	Value	Replace the following hard coded string in view extension
idLabelText	This is the carrier id	Carrier id
nameLabelText	This is the carrier name	Carrier name

3. Implement a controller extension for the controller of *Carrier* view and implement the function `extUX410HookFunction`. The function shows an *sap.m.MessageToast* with the message **`Called from Init`**. Test if your changes are applied.

Implement and Extend an Extension Point

> Note:
> SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, step procedures and naming of fields and buttons may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

In this exercise you will work with SAP Fiori Extension techniques. The basis for this exercise is the implementation of the exercise **Object Page**.

Extend your deployed application *ZUX410_APP## (system is FSD)*.

1. Open the result of the exercise **Object Page**. Add the ID `idCarrierDetails` to the *sap.m.FlexBox* control located in the *sap.uxap.ObjectPageLayout* control.

   > Note:
   > Create a new separate project by using the export and import function of the SAP Web IDE. Export your objectpage project and import it with the new name of **`objectPageWithExtension`**.

   a) Open the file `Carrier.view.xml` located in the view folder of the project.

   b) Add the `idCarrierDetails` to the *sap.m.FlexBox* control, located in the `expandedContent` aggregation of the *sap.uxap.ObjectPageLayout* control.

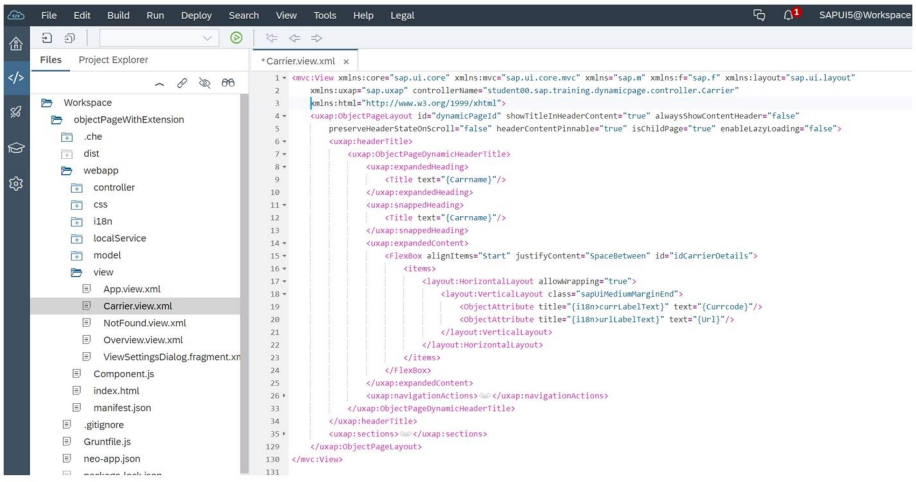


Figure 271: Add idCarrierDetails

2. Add an *sap.ui.core.ExtensionPoint* with the name `ux410_extension` immediately after the *sap.m.FlexBox* control with the new ID `idCarrierDetails`.

a) Add the *sap.ui.core.ExtensionPoint* with the name `ux410_extension` immediately after the *sap.m.FlexBox* control with the new ID `idCarrierDetails`.

Figure 272: Add Extension

3. Add a controller `hook` function with the name `extUX410HookFunction` to the controller of view *Carrier*. Call, if it exists, the `hook` function at the end of the `onInit` function of the controller. Add the following documentation before the invocation.

Table 72: extUX410HookFunction Properties and Documentation

JSDoc property	Documentation
@ControllerHook	This hook method can be used to add some extra functionality
@callback	student##.sap.training.dynamicpage.controller.Carrier~extUX410HookFunction
@return	{void}

a) Open the controller `Carrier.controller.js` located in the controller folder of your project.

b) Add the `extUX410HookFunction` declaration to the controller and assign **`null`**. Use your own user group number.

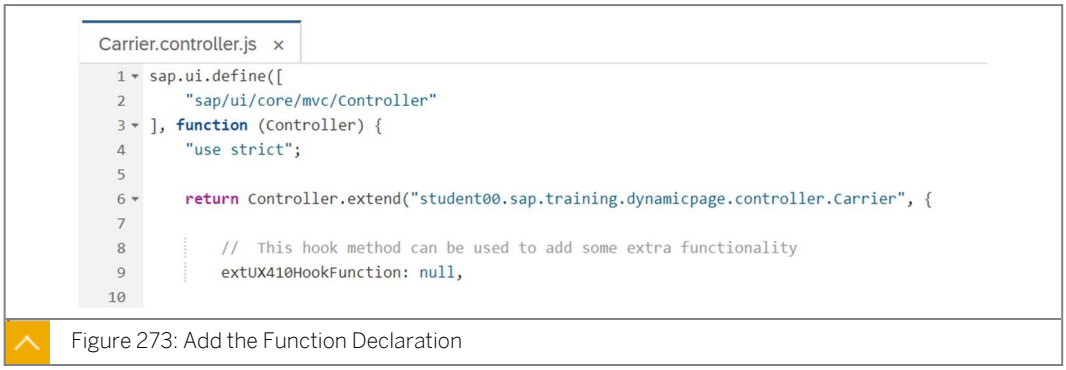


Figure 273: Add the Function Declaration

c) Go to the `onInit` function of the controller and check that the function `extUX410HookFunction` exists. Call the function. Add the listed comments before the function.

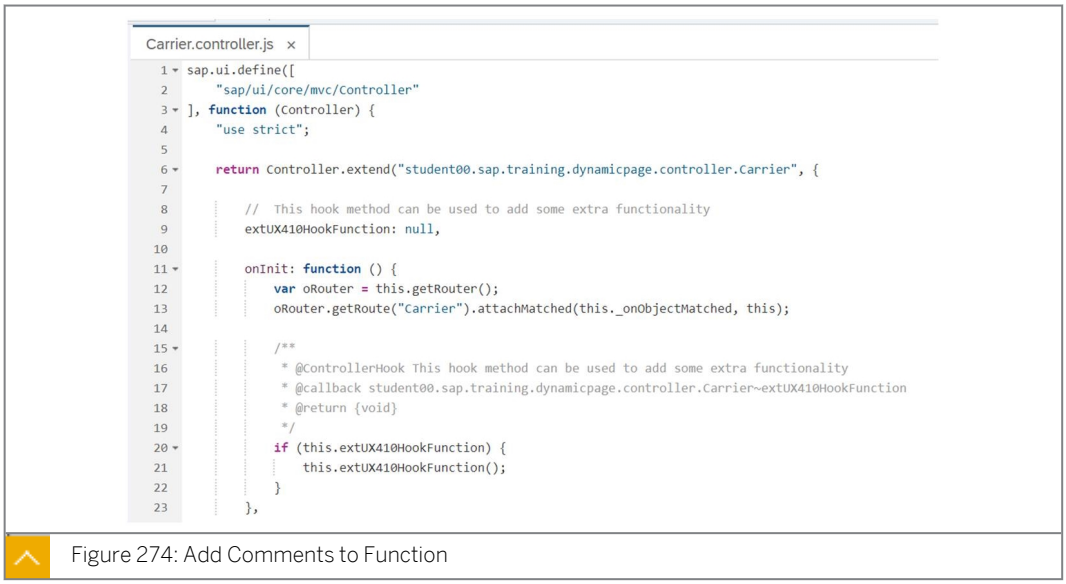


Figure 274: Add Comments to Function

d) Save the changes.

4. Deploy your application.

Table 73: Application Properties and Values

Property	Value
Application name	ZUX410_App##
Description	Flight app
System	FSD_100 - FSD
Package	ZTRAIN_##
Transport request	Choose the transport request provide by your teacher.

a) From the context menu of your project, select *Deploy → Deploy to SAPUI5 ABAP Repository*.

b) Choose the system *FSD_100 – FSD* from the list of available systems in the upcoming dialog. Ensure that the radio button *Deploy a new application* is selected and choose *Next*.

Enter your username **`Train-##`** and password if requested.

c) Enter the value **`ZUX410_App##`** in the *Name* field and the value **`Flight app`** in the *Description* field. Browse to choose the package.

d) Search for your package *ZTRAIN_##* from the list and choose *OK*.

Ensure you use your group packages. The packages listed may vary.

e) Choose *Next*.

f) Select the *transport request* provided by your teacher. The dialog transport request selection may vary.

g) Choose *Finish*

h) Wait until your project is deployed.

Extend the Deployed Application.

1. Extend your deployed application *ZUX410_APP##* (system is FSD). Create an extension project with the name **`ZUX410_APP##Extension`**. Hide the UI control with ID `idCarrierDetails` by using the extension pane. Create a view extension for the extension point `ux410_extension` using the extension pane. Add the code provided in the example for the extension. Test your extended application.

Carrier_ux410_extensionCusto... ×

```
<core:FragmentDefinition xmlns:core="sap.ui.core" xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m"
    xmlns:f="sap.f" xmlns:layout="sap.ui.layout"
    xmlns:uxap="sap.uxap" xmlns:html="http://www.w3.org/1999/xhtml">
    <FlexBox alignItems="Start" justifyContent="SpaceBetween">
        <items >
            <layout:HorizontalLayout allowWrapping="true">
                <layout:VerticalLayout class="sapUiMediumMarginEnd">
                    <ObjectAttribute title="{i18n>currLabelText}" text="{Currcode}"/>
                    <ObjectAttribute title="{i18n>urlLabelText}" text="{Url}"/>
                    <ObjectAttribute title="Carrier id" text="{Carrid}"/>
                    <ObjectAttribute title="Carrier name" text="{Carrname}"/>
                </layout:VerticalLayout >
            </layout:HorizontalLayout >
        </items>
    </FlexBox>
</core:FragmentDefinition>
```

Figure 275: Extension Code

a) Select the *Workspace* folder from your project explorer and from the context menu choose *New → Extension Project*.

b) In the *Select Application* selection box of the new dialog that appears, choose *SAPUI5 ABAP Repository*.

c) Select the system FSD from the list of available systems and enter **`zux410_`** in the *Search* field. Select your application from the list of available applications and choose *OK*. Enter your user **`Train-##`** and password if requested.

d) Leave the upcoming dialog unchanged and choose *Next*.

e) Choose *Finish* to confirm the project creation. Select the *Open extension project in extensibility pane* checkbox.

f) The extension pane opens when it is created. Enter your credentials for the front-end server and choose *Sign In*.

g) Expand the *Carrier* view in the outline and drill down to the UI control with the ID `idCarrierDetails`.

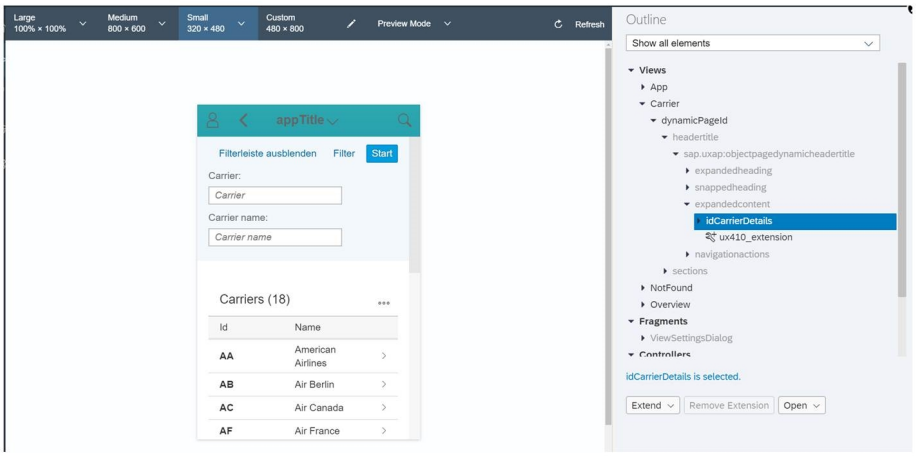


Figure 276: UI idCarrierDetails

h) Choose the *Extend-Selection* button and *Hide Control*.

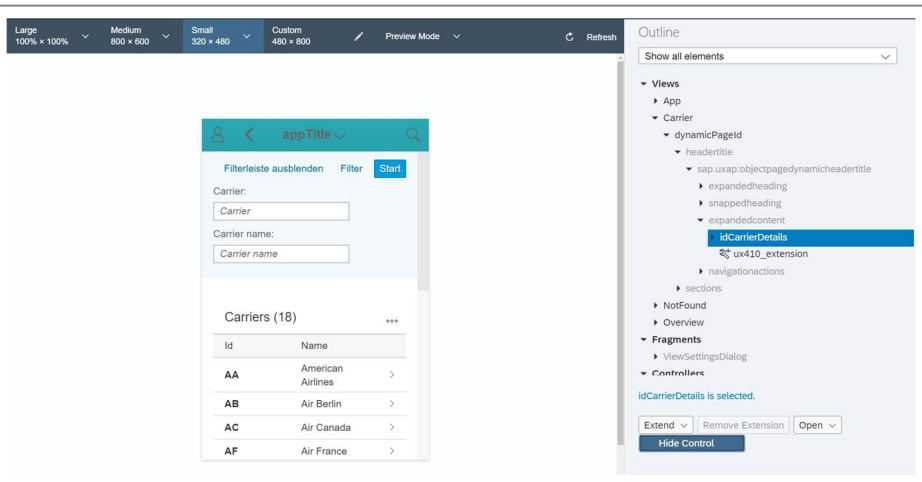


Figure 277: Choose Buttons

i) Close the dialog box that appears.

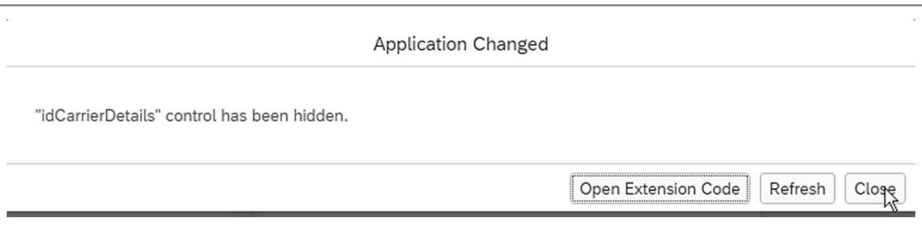


Figure 278: Close Dialog

j) Choose the extension point *ux410_extension* and the *Extend View/Fragment* button from the selection.

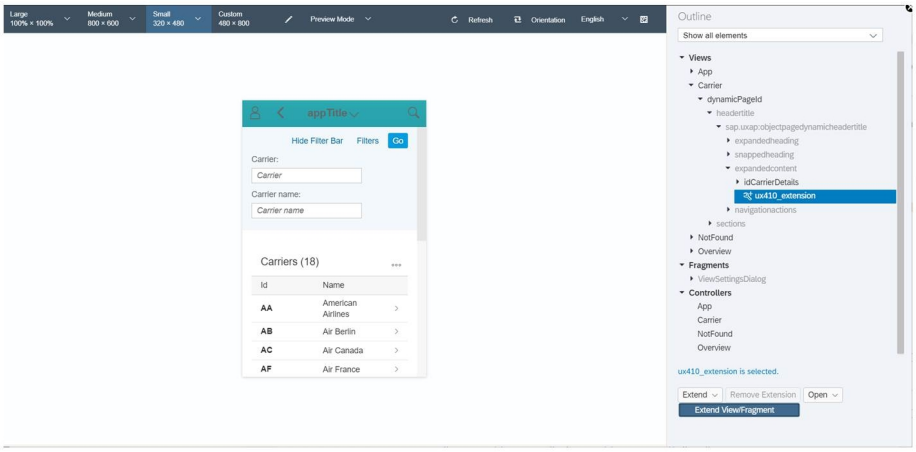


Figure 279: Choose Extension Options

k) Choose *Open Extension Code*.

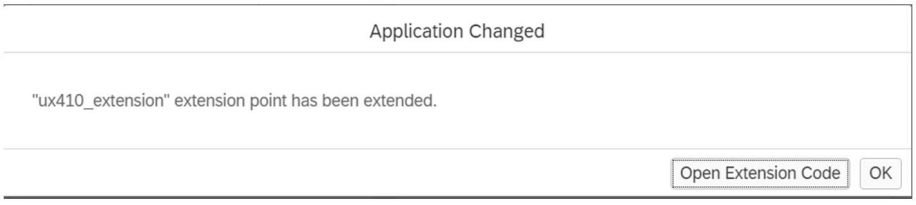


Figure 280: Open Extension Code

l) After creation, the view extension is opened by the SAP Web IDE.

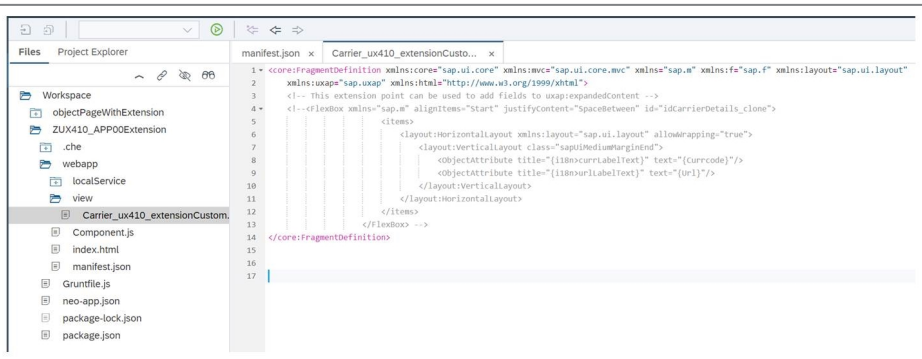


Figure 281: View Extension

m) Add view fragments details.

Carrier_ux410_extensionCusto... ×

```
<core:FragmentDefinition xmlns:core="sap.ui.core" xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m"
    xmlns:f="sap.f" xmlns:layout="sap.ui.layout"
    xmlns:uxap="sap.uxap" xmlns:html="http://www.w3.org/1999/xhtml">
    <FlexBox alignItems="Start" justifyContent="SpaceBetween">
        <items >
            <layout:HorizontalLayout allowWrapping="true">
                <layout:VerticalLayout class="sapUiMediumMarginEnd">
                    <ObjectAttribute title="{i18n>currLabelText}" text="{Currcode}"/>
                    <ObjectAttribute title="{i18n>urlLabelText}" text="{Url}"/>
                    <ObjectAttribute title="Carrier id" text="{Carrid}"/>
                    <ObjectAttribute title="Carrier name" text="{Carrname}"/>
                </layout:VerticalLayout >
            </layout:HorizontalLayout >
        </items>
    </FlexBox>
</core:FragmentDefinition>
```

Figure 282: View Fragments

n) Save your changes.

o) To test your application select the `index.html` file and from the context menu, choose *Run → Run as Web Application*.

p) Allow *SAP Web IDE* to store files on your device when requested.

q) Choose the tile *ZUX410_APP##Extension*.

r) Enter your credentials for the front-end server.

s) Select a carrier from the list and check the result.

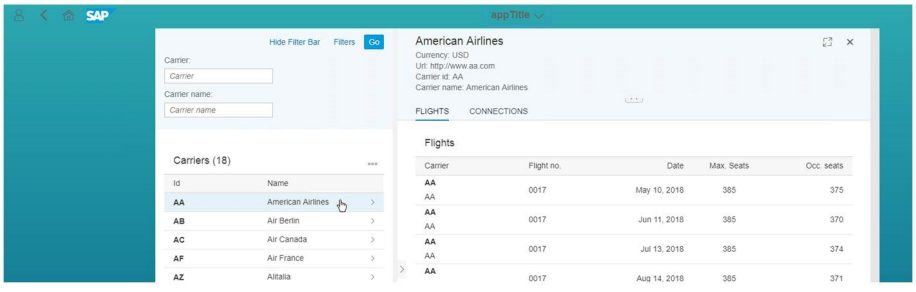


Figure 283: Select a Carrier

2. Add an i18n-resource customizing extension to your project and enter the new text properties listed. Replace the hard coded text literals located in the view extension. Test your application.

Table 74: Text Properties

Key	Value	Replace the following hard coded string in view extension
idLabelText	This is the carrier id	Carrier id
nameLabelText	This is the carrier name	Carrier name

a) Choose your project root folder and from the context menu, choose *New → Extension*.

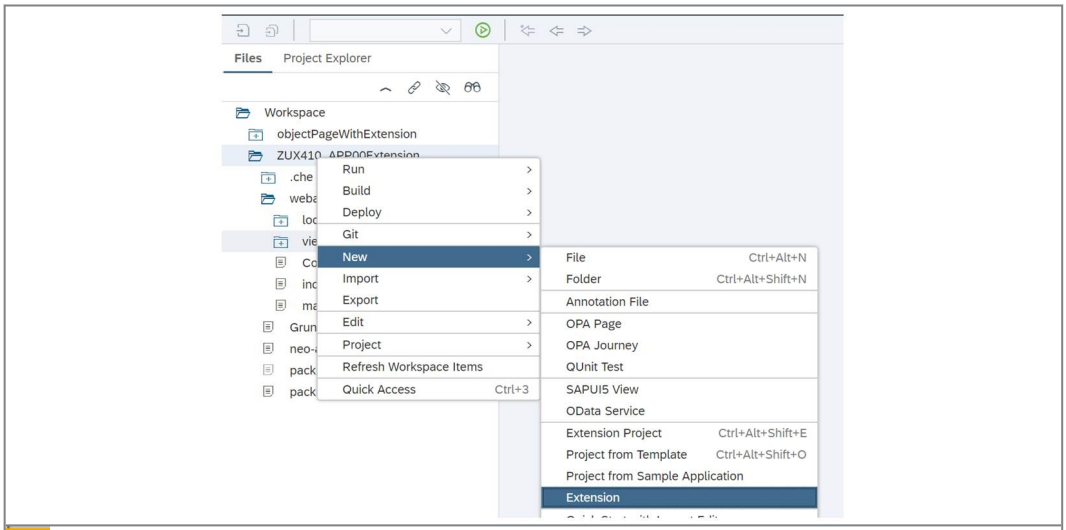


Figure 284: New Extension

b) Confirm the selection by choosing *Next* in the dialog box.

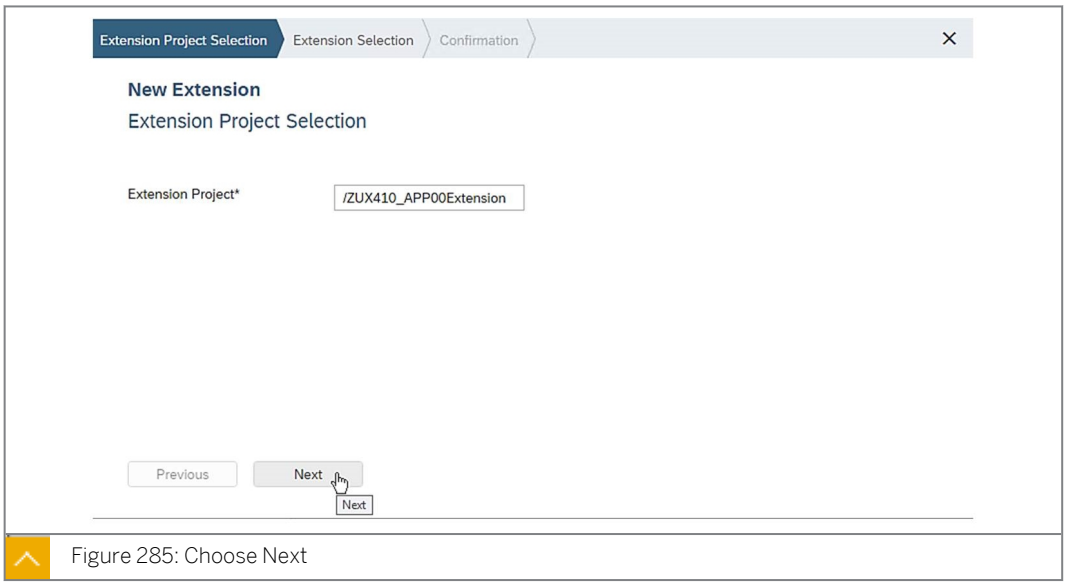


Figure 285: Choose Next

c) Choose *i18n Resource Text Customization* and *Finish*.

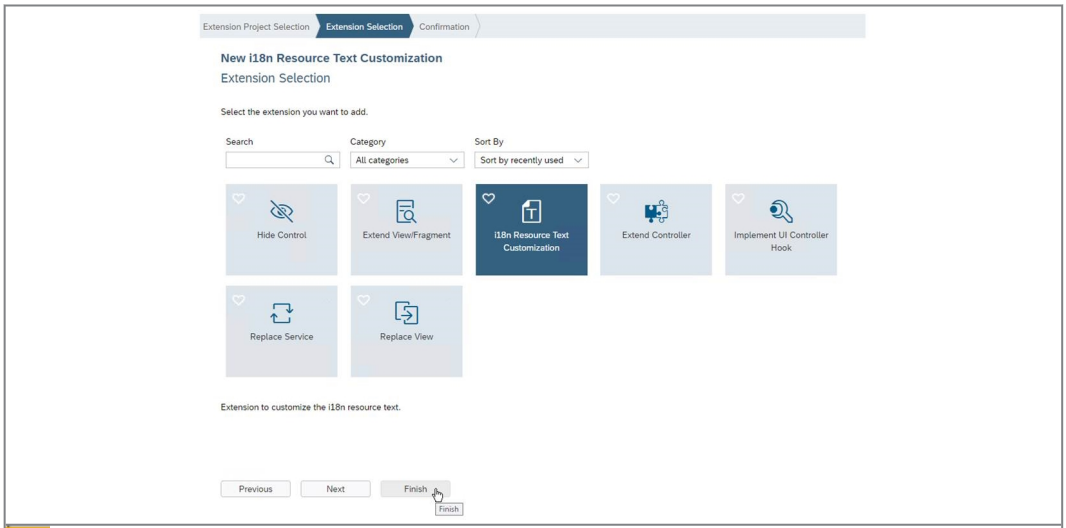


Figure 286: i18n Resource Text Customization

d) Change the value key of the i18n resource file.

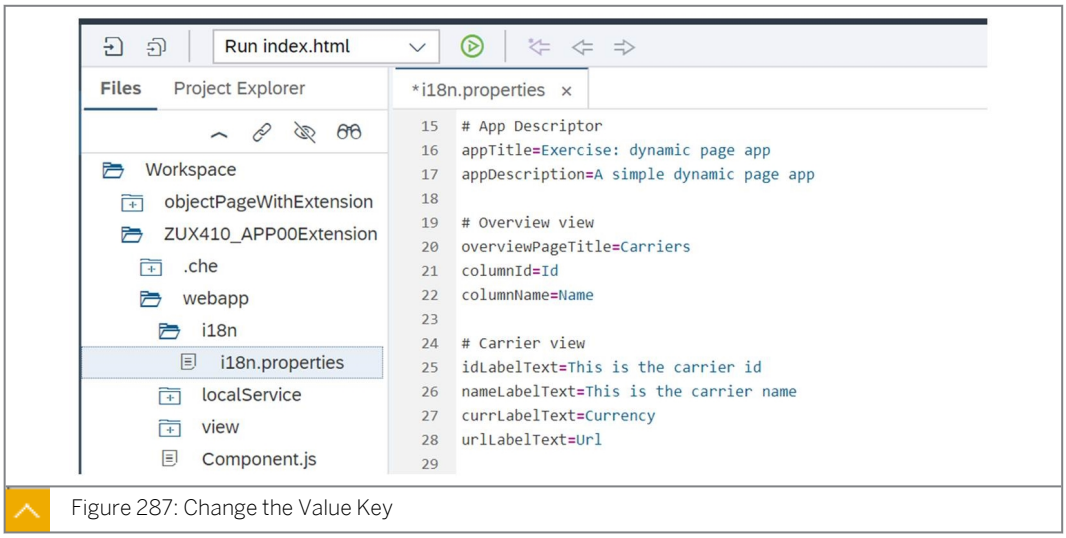


Figure 287: Change the Value Key

e) Add the i18n resource binding to the view extension.

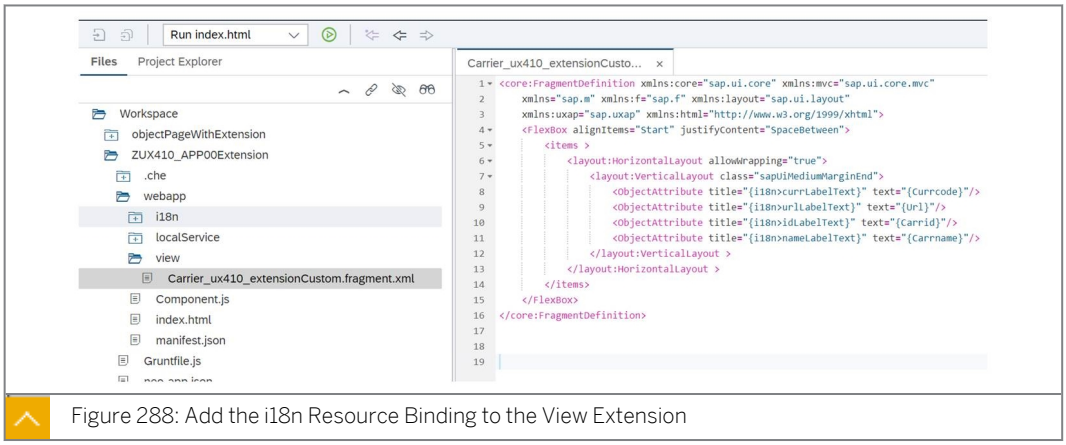


Figure 288: Add the i18n Resource Binding to the View Extension

f) Save your work.

g) Start your application again to test if the changes are applied.

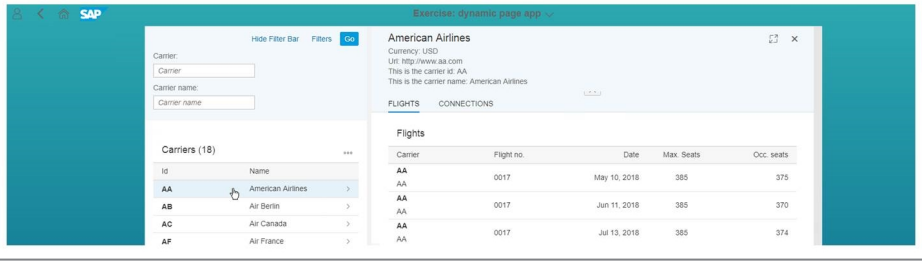


Figure 289: Changes Applied

3. Implement a controller extension for the controller of *Carrier* view and implement the function `extUX410HookFunction`. The function shows an *sap.m.MessageToast* with the message **`Called from Init`**. Test if your changes are applied.

   a) In the root folder of your project, from the context menu, choose *New* → *Extension* .

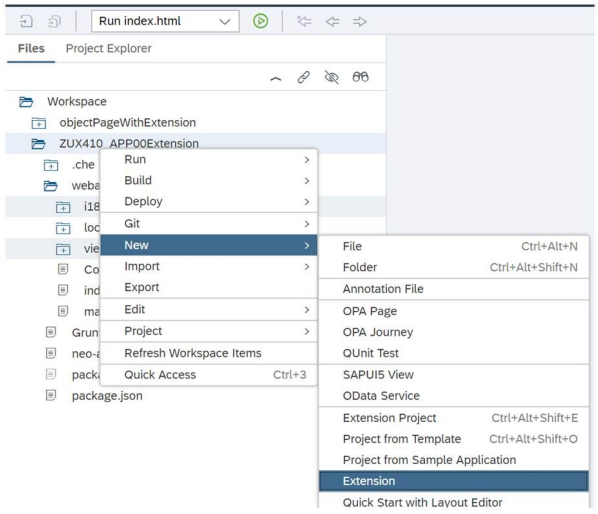


Figure 290: Choose New Extension

   b) Confirm by choosing *Next*.

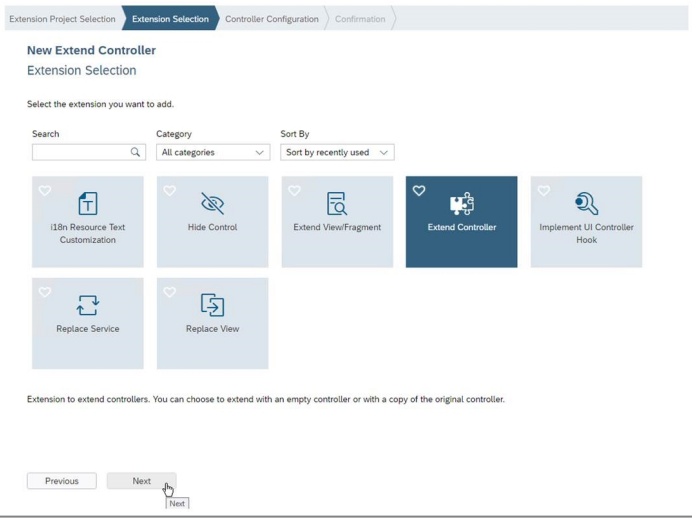


Figure 291: Choose Next

   c) Choose the controller of the *Carrier* view from the available list of controllers. Also choose *Empty Controller* from the *Replace with selection* list and choose *Finish*.

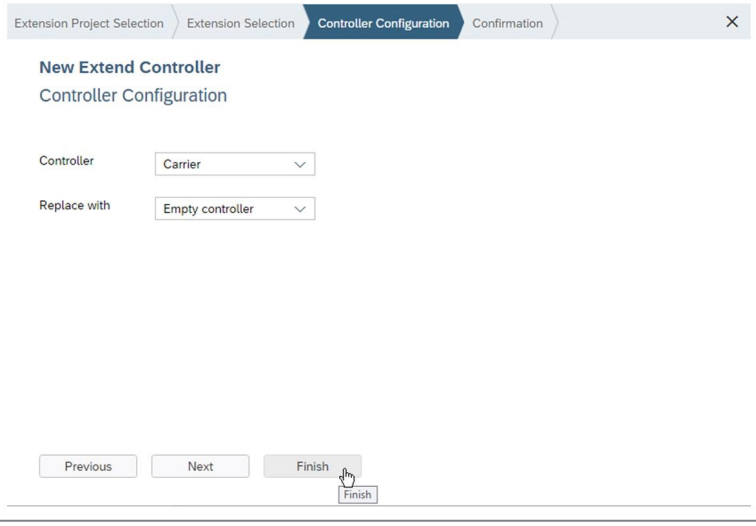


Figure 292: Controller Configuration

d) Implement the hook function `extUX410HookFunction`.

Figure 293: Hook Implementation

e) Test your application.

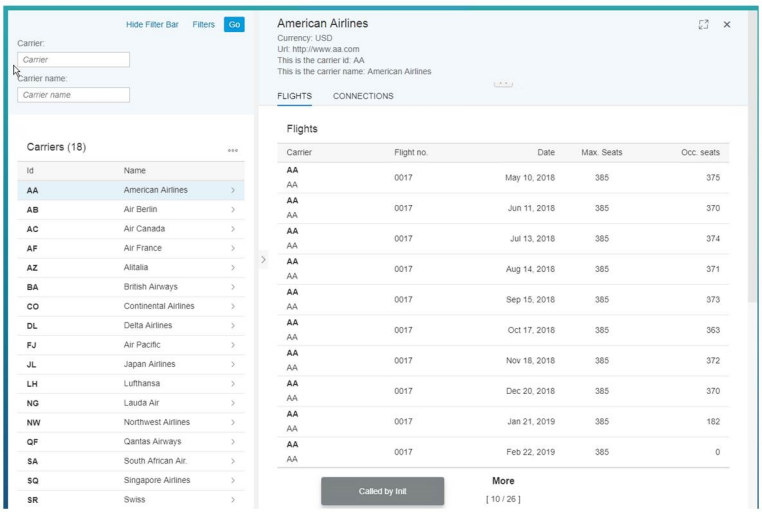


Figure 294: Display Extension

f) Change the *sap_bluecrystal* theme to *sap_belize*.

Note:
Should you wish to change the sap_bluecrystal theme in your extension project, open the `index.html` file and navigate to the bootstrapping.

```
<script id="sap-ui-bootstrap"
        src="resources/sap-ui-core.js"
        data-sap-ui-libs="sap.ushell, sap.collaboration, sap.m, sap.ui.layout"
        data-sap-ui-theme="sap_belize"
        data-sap-ui-xx-bindingSyntax="complex"
        data-sap-ui-compatVersion="edge"
        data-sap-ui-resourceroots='{"student00.sap.training.dynamicpage.ZUX410_APP00Extension": "./"}'
        data-sap-ui-frameOptions='allow'> // NON-SECURE setting for testing environment
</script>
```

Figure 295: Change Theme Option

g) Restart the application to see the SAP Fiori Launchpad of the *SAP Web IDE* in the `sap_belize` theme.

h) Select the tile.

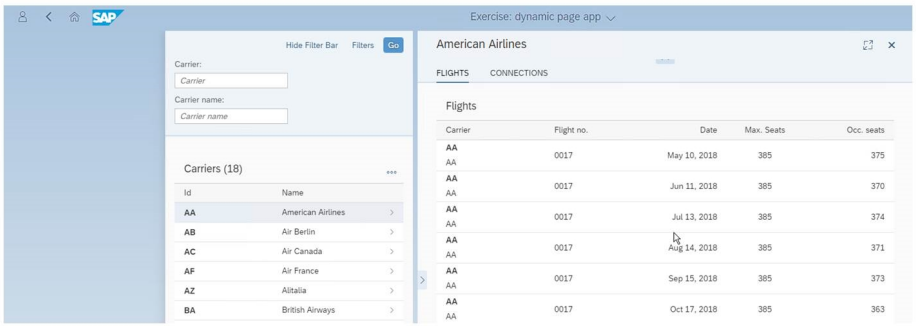


Figure 296: Extension View

Unit 14
Exercise 15

Implement List Report using SAP Fiori Elements

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, step procedures and naming of fields and buttons may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

Note:
Sometimes after the annotations are changed in the **ABAP CDS**, they are not reflected in the *SAP Fiori Elements Application*, or in the *SAP Web IDE*. If this happens, clean the browser caches and restart the application.

Create an ABAP project in *Eclipse JEE*.

1. Create an eclipse project in *Eclipse JEE* with the following values.

   Table 75: Project Values

Property	Value
system	S4D
client	100
user	Train-## (replace ## with your group number)
password	The password for your train-## user from the back-end system.
project name	Use the default project name S4D_100_train##_en (replace ## with your group number)
package	ZTRAIN_## (replace ## with your group number)

Task 1: Create a CDS View

1. Add a new *CDS* view with the following values.

   Table 76: CDS View Values

Property	Value
name	ZUX410_C_FlightUI## (replace ## with your group number)
description	UX410 Fiori Elements ##

Property	Value
package	ZTRAIN_## (replace ## with your group number)
transport Request	Use the transport request provided by the trainer.
template for creating New Data Definition	No template

2. Copy the content of the file `S:\Courses\UX410_20\Templates\ZUX410_C_FLIGHTUI##.txt` to the newly created *ABAP CDS* view. Replace ## with your group number.
3. Activate your changes.

Task 2: Register the *OData Service* on the front-end server

1. For the next step, your need to register the *OData Service* on the front-end server of the training environment. Use the value `ZUX410_C_FLIGHTUI##_CDS` as a technical service name.

Task 3: Create a project in the *SAP Web IDE*.

1. Use the SAPUI5 project template *List Report Application* to create a new SAPUI5 project. Use the following settings.

   Table 77: SAPUI5 Project Settings

Property	Value
Project name	fiorielements
Title	FlightExercise
Namespace	student##.sap.training
Service Catalog	FSD_100 - FSD
OData-Service	ZUX410_C_FLIGHTUI##_CDS (replace ## with your group number)
Description	<leave empty>
Annotation Selection	ZUX410_C_FLIGHTUI##_CDS_VAN (replace ## with your group number)
OData Collection	ZUX410_C_FLIGHTUI## (replace ## with your group number)
Smart Variant Management for List Report	true
Flexible Column Layout	true

2. Test your application. The application shows no data in the list initially. Select columns from the list of available columns.

Task 4: Show flights in the List Report Using UI Annotations

In the next step, you enhance your *ABAP-CDS* to display columns in the list report.

1. Add annotation for the `ZUX410_C_FLIGHTUI##` entity type using the term **`UI.headerInfo`**. Use the information in the table to describe basic information of the data you want to display.

   Table 78: Data to Display

Property	Datatype	Value
typeNamePlural	string	Flights
typeName	string	Flight

2. Use the term `UI.lineItem` to add the following information to your *CDS* and activate your changes.

   Table 79: UI.lineItem Information

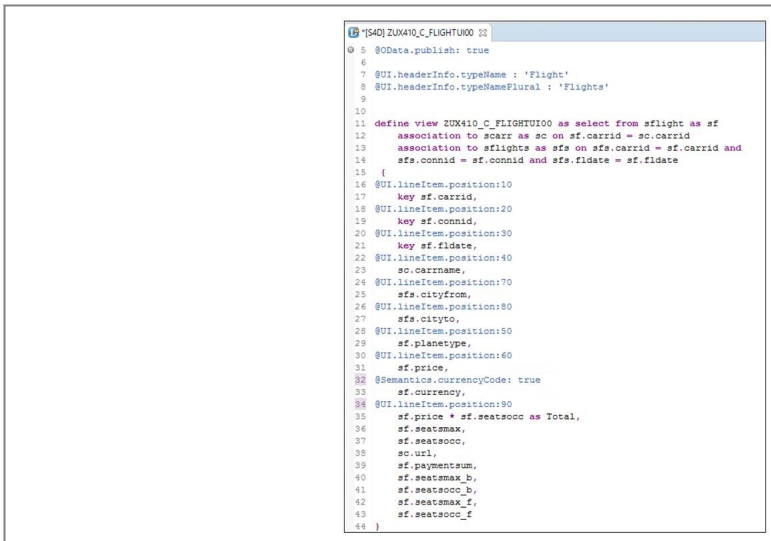
Property	Position
carrrid	10
connid	20
fldate	30
carrname	40
planetype	50
price	60
cityfrom	70
cityto	80

3. To test your implementation, go to the *SAP Web IDE* and start the application created in the previous task.

Task 5: Add Semantic Information to the ABAP-CDS

In the next task you add the *Total* field to the list report and assign semantic information to the *Total* field of your *ABAP-CDS*.

1. Add the semantic term `currencyCode` to the *currency* field of your projection list and assign the value *true*.
2. Add the annotation `@Semantics.currencyCode` to the *Currency* field of your projection list and assign the boolean value *true*.



```
*[S4D] ZUX410_C_FLIGHTUI00
@OData.publish: true

@UI.headerInfo.typeName : 'Flight'
@UI.headerInfo.typeNamePlural : 'Flights'


define view ZUX410_C_FLIGHTUI00 as select from sflight as sf
    association to scarr as sc on sf.carrid = sc.carrid
    association to sflights as sfs on sfs.carrid = sf.carrid and
    sfs.connid = sf.connid and sfs.fldate = sf.fldate
 {
@UI.lineItem.position:10
    key sf.carrid,
@UI.lineItem.position:20
    key sf.connid,
@UI.lineItem.position:30
    key sf.fldate,
@UI.lineItem.position:40
    sc.carrname,
@UI.lineItem.position:70
    sfs.cityfrom,
@UI.lineItem.position:80
    sfs.cityto,
@UI.lineItem.position:50
    sf.planetype,
@UI.lineItem.position:60
    sf.price,
@Semantics.currencyCode: true
    sf.currency,
@UI.lineItem.position:90
    sf.price * sf.seatsocc as Total,
    sf.seatsmax,
    sf.seatsocc,
    sc.url,
    sf.paymentsum,
    sf.seatsmax_b,
    sf.seatsocc_b,
    sf.seatsmax_f,
    sf.seatsocc_f
}
```

Figure 301: Assign Values

3. Add the annotation `@Semantics.amount.currencyCode` to the *Total* field of your projection list and assign the string value **`currency`** to the term. Activate your changes.

4. Test your application in the *SAP Web IDE*.

Task 6: Add Responsive Behaviour to the List Report.

In the next task you will add responsive behaviour to your application.

1. Add the following annotations and information to the fields of the projection list.

   Table 80: Projection List Field Information

Field	Term	Property	Value
planetype	lineItem	importance	#HIGH
Total	lineItem	importance	#MEDIUM
price	lineItem	importance	#LOW

2. Test your application in the *SAP Web IDE*. Use the features of the Google Chrome Development Tools to change the form factor.

Unit 14
Solution 15

Implement List Report using SAP Fiori Elements

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, step procedures and naming of fields and buttons may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

Note:
Sometimes after the annotations are changed in the ABAP CDS, they are not reflected in the *SAP Fiori Elements Application*, or in the *SAP Web IDE*. If this happens, clean the browser caches and restart the application.

Create an ABAP project in *Eclipse JEE*.

1. Create an eclipse project in *Eclipse JEE* with the following values.

   Table 75: Project Values

Property	Value
system	S4D
client	100
user	Train-## (replace ## with your group number)
password	The password for your train-## user from the back-end system.
project name	Use the default project name S4D_100_train##_en (replace ## with your group number)
package	ZTRAIN_## (replace ## with your group number)

   a) Choose *Eclipse Jee 2019-06* from the windows *Start* menu.

   b) When Eclipse opens, choose the *Workbench* button in the upper right corner.

   c) The Eclipse workbench appears.

   d) Choose *File* → *New* → *Other* from the window.

   e) Enter the value **`ABAP`** in the *Search* field, choose *ABAP Project* from the list, and choose *Next*.

f) Choose *00 S4D GUI non SNC [PAS]* from the list of available systems and choose *Next*.

g) Choose *Next* to confirm the settings in the following dialog.

h) Enter **`Client 100`** and your user **`Train-##`** (replace ## with your group number). Enter your password for the S4D system. Choose *Next*.

i) Add the development package *ZTRAIN-00* to the list of favourite packages by choosing the *Add* button.

j) Enter the package name **`ZTRAIN`** in the *Search* field, choose the package assigned to your group from the list of found packages and choose *OK*.

k) Create the package by choosing *Finish*.

l) Confirm the dialog to switch the perspective to the ABAP perspective.

m) Expand the newly created project in the new perspective. Then expand the entry *Favorite Packages*. Here you will see your package.

Task 1: Create a CDS View

1. Add a new *CDS* view with the following values.

   Table 76: CDS View Values

Property	Value
name	ZUX410_C_FlightUI## (replace ## with your group number)
description	UX410 Fiori Elements ##
package	ZTRAIN_## (replace ## with your group number)
transport Request	Use the transport request provided by the trainer.
template for creating New Data Definition	No template

   a) Select the entry *Favorite Packages* in your project and from the context menu, choose *New → Other → ABAP → ABAP Repository Object*.

   b) Expand the *Core Data Services* folder and choose *Data Definition*.

   c) Enter the value **`ZUX410_C_FlightUI##`** in the *Name* field and in the *description* field, enter **`UX410 Fiori Elements ##`** and choose *Next*.

   d) Select the transport request provided by your trainer and choose *Next*.

   e) Deselect the *Use the selected template* checkbox.

   f) Choose *Finish*.

2. Copy the content of the file `S:\Courses\UX410_20\Templates\ZUX410_C_FLIGHTUI##.txt` to the newly created *ABAP CDS* view. Replace ## with your group number.

a) Open *File Explorer* and navigate to `S:\Courses\UX410_20\Templates\ZUX410_C_FLIGHTUI##.txt`.

b) Right click on the file name and choose *Sappad* from the context menu to open the file.

c) Copy the content from the file into the newly created *CDS* view.

d) Replace the ## with your group number.

3. Activate your changes.

a) Activate your changes by choosing the  icon on the toolbar

b) Eclipse shows a warning. This is correct. Ignore the warning and proceed.

Task 2: Register the *OData Service* on the front-end server

1. For the next step, your need to register the *OData Service* on the front-end server of the training environment. Use the value `ZUX410_C_FLIGHTUI##_CDS` as a technical service name.

a) Open *SAP GUI*.

b) Choose the entry *00 FSD* from the folder *00 Development*.

c) Enter the credentials for the FSD System of your training environment. Enter the user **`train-##`** and your password.

d) Expand your *User* menu and expand the entry *SAP Gateway*. Double-click *Activate and Maintain Services* from the list.

e) After the transaction has started, choose the *Add Service* button.

f) Enter the value **`S4HANA`** in the *system alias* field (you can use F4-value help). Enter **`Z*`** in the *Technical Service Name* field, and choose *Enter*.

g) Select *ZUX410_C_FLIGHTUI##_CDS* from the list of available services, and choose the *Add Service* button.

h) Assign your changes to the *Package ZTRAIN_##* (replace ## with your group number).

i) During the generation of the necessary artefacts you will be asked to assign the artefacts to a transport request. Use the request assigned to your user. (The dialog for assignment might pop up more than once).

j) Choose *OK* in the success dialog.

k) Switch to the initial screen of transaction `/IWFND/MAINT_SERVICE` and check your services were added successfully. Use *F3* or the *Back* button of *SAP GUI*.

Task 3: Create a project in the *SAP Web IDE*.

1. Use the SAPUI5 project template *List Report Application* to create a new SAPUI5 project. Use the following settings.

Table 77: SAPUI5 Project Settings

Property	Value
Project name	fiorielements
Title	FlightExercise
Namespace	student##.sap.training
Service Catalog	FSD_100 - FSD
OData-Service	ZUX410_C_FLIGHTUI##_CDS (replace ## with your group number)
Description	<leave empty>
Annotation Selection	ZUX410_C_FLIGHTUI##_CDS_VAN (replace ## with your group number)
OData Collection	ZUX410_C_FLIGHTUI## (replace ## with your group number)
Smart Variant Management for List Report	true
Flexible Column Layout	true

a) Log on to the *SAP Web IDE Full-Stack* in Google Chrome and select the workspace folder and choose from the context menu *New → Project from Template*.

b) Select the *List Report Application* tile and choose *Next*.

c) Enter the project settings defined in the table and choose *Next*.

d) Choose the system *FSD_100 – FSD* from the list of available service catalog systems.

e) Enter your user credentials for the front-end server and choose *Sign In*.

f) Enter **`zux`** in the *Search* field and choose your OData Service *ZUX410_C_FLIGHTUI##_CDS* from the search result. Wait until the *Next* button is enabled and choose *Next*.

g) In the next dialog, select the *ZUX410_C_FLIGHTUI##_CDS_VAN* checkbox and choose *Next*.

h) Choose *ZUX410_C_FLIGHTUI##* as OData collection, check the *select* box for *Flexible Column Layout* and ensure the *select* box for *Smart Variant Management for List Report* is selected. Choose *Finish*.

i) Your new project appears in the workspace of your *SAP Web IDE*.

2. Test your application. The application shows no data in the list initially. Select columns from the list of available columns.

a) To test the application, select the project root folder, and choose from the context menu *Run → Run as SAP Fiori Launchpad Sandbox*.

b) Choose *Store files on the device* in the dialog and enter your user credentials for the front-end server. Enter **`train-##`** using your group number and password and choose *Sign In*.

c) Choose the cogwheel  symbol in the application.

d) Select some of the columns in the new dialog, and choose *OK*.

e) Choose the *Go* button to execute the server request.

f) The table now shows some flight data.

Task 4: Show flights in the List Report Using UI Annotations

In the next step, you enhance your *ABAP-CDS* to display columns in the list report.

1. Add annotation for the `ZUX410_C_FLIGHTUI##` entity type using the term **`UI.headerInfo`**. Use the information in the table to describe basic information of the data you want to display.

   Table 78: Data to Display

Property	Datatype	Value
typeNamePlural	string	Flights
typeName	string	Flight

   a) Start *Eclipse Oxygen* as shown in the first task of the exercise, if not already open.

   b) Open the created *CDS* located in the package *ZTRAIN_##*, if not already displayed.

   c) Add the listed annotations to your CDS.

```
[S4D] ZUX410_C_FLIGHTUI00
@AbapCatalog.sqlViewName: 'ZUX410FLIGHTUI00'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'UX410 Flight UI'
@OData.publish: true

@UI.headerInfo.typeName : 'Flight'
@UI.headerInfo.typeNamePlural : 'Flights'


define view ZUX410_C_FLIGHTUI00 as select from sflight as sf
    association to scarr as sc on sf.carrid = sc.carrid
    association to sflights as sfs on sfs.carrid = sf.carrid and
    sfs.connid = sf.connid and sfs.fldate = sf.fldate
 {
    key sf.carrid,
    key sf.connid,
    key sf.fldate,
    sc.carrname,
    sfs.cityfrom,
    sfs.cityto,
    sf.planetype,
    sf.price,
    sf.currency,
    sf.price * sf.seatsocc as Total,
    sf.seatsmax,
    sf.seatsocc,
    sc.url,
    sf.paymentsum,
    sf.seatsmax_b,
    sf.seatsocc_b,
    sf.seatsmax_f,
    sf.seatsocc_f
}
```

Figure 297: Add Annotations

   d) Save your changes.

2. Use the term `UI.lineItem` to add the following information to your *CDS* and activate your changes.

   Table 79: UI.lineItem Information

Property	Position
carrrid	10

Property	Position
connid	20
fldate	30
carrname	40
planetype	50
price	60
cityfrom	70
cityto	80

a) Add the UI.lineItem term to the fields.

```
@AbapCatalog.sqlViewName: 'ZUX410FLIGHTUI00'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'UX410 Flight UI'
@OData.publish: true

@UI.headerInfo.typeName : 'Flight'
@UI.headerInfo.typeNamePlural : 'Flights'


define view ZUX410_C_FLIGHTUI00 as select from sflight as sf
    association to scarr as sc on sf.carrid = sc.carrid
    association to sflights as sfs on sfs.carrid = sf.carrid and
    sfs.connid = sf.connid and sfs.fldate = sf.fldate
 {
@UI.lineItem.position:10
    key sf.carrid,
@UI.lineItem.position:20
    key sf.connid,
@UI.lineItem.position:30
    key sf.fldate,
@UI.lineItem.position:40
    sc.carrname,
@UI.lineItem.position:70
    sfs.cityfrom,
@UI.lineItem.position:80
    sfs.cityto,
@UI.lineItem.position:50
    sf.planetype,
@UI.lineItem.position:60
    sf.price,
    sf.currency,
    sf.price * sf.seatsocc as Total,
    sf.seatsmax,
    sf.seatsocc,
    sc.url,
    sf.paymentsum,
    sf.seatsmax_b,
    sf.seatsocc_b,
    sf.seatsmax_f,
    sf.seatsocc_f
}
```

Figure 298: Add UI.lineItem Term

b) Save your changes.

c) Activate your changes with the  icon.

3. To test your implementation, go to the *SAP Web IDE* and start the application created in the previous task.

a) Open the *SAP Web IDE Fullstack* if not already open.

b) Start your application as described in the previous task.

c) Confirm the application is allowed to store files on your device.

d) The application displays with no data, which is correct.

Figure 299: Application with No Data Displayed

e) Choose *Go* to send the get request.

f) The results appear in the table when the request is processed successfully.

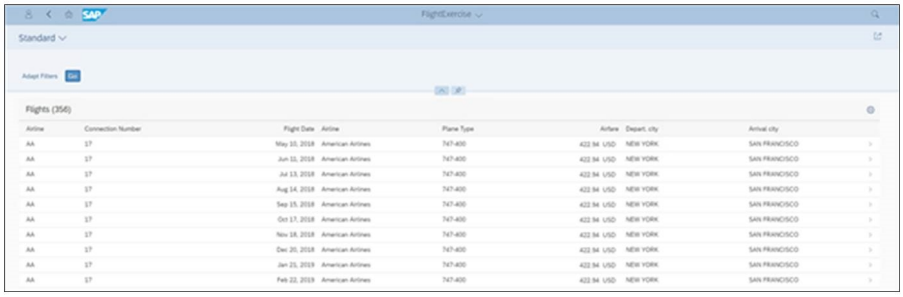


Figure 300: Results Table

Task 5: Add Semantic Information to the ABAP-CDS

In the next task you add the *Total* field to the list report and assign semantic information to the *Total* field of your *ABAP-CDS*.

1. Add the semantic term `currencyCode` to the *currency* field of your projection list and assign the value *true*.
   a) Start *Eclipse* as shown in the first task of the exercise, if not already open.
   b) Open the created *CDS* located in the package *ZTRAIN_##*, if not already displayed.
   c) Add the term lineItem to the calculated *Total* field and assign the *position 90* to the *lineItem*.
2. Add the annotation `@Semantics.currencyCode` to the *Currency* field of your projection list and assign the boolean value *true*.

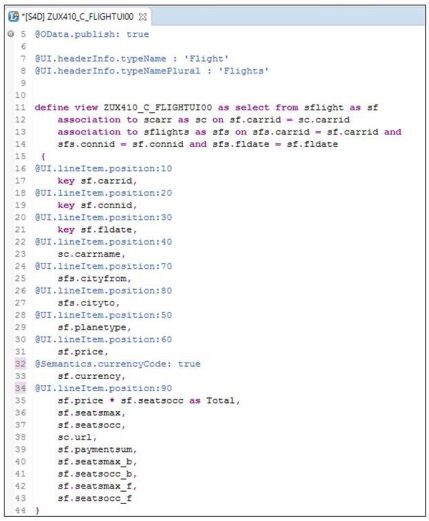


Figure 301: Assign Values

3. Add the annotation `@Semantics.amount.currencyCode` to the *Total* field of your projection list and assign the string value **`currency`** to the term. Activate your changes.

   a) Add the annotation to the total field.

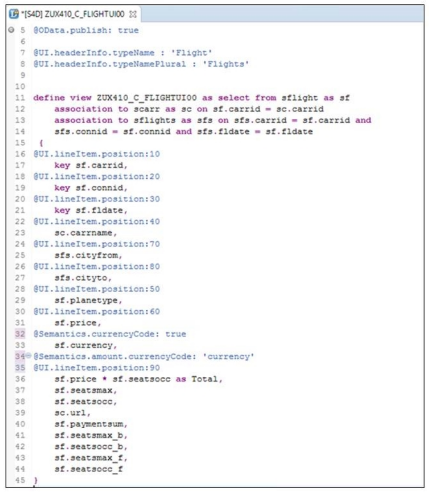


Figure 302: Add Annotation

   b) Activate your changes by choosing  .

4. Test your application in the *SAP Web IDE*.

   a) Open the *SAP Web IDE Fullstack* if not already open.

   b) Start your application as described in the previous task.

   c) Confirm the application is allowed to store files on your device.

   d) The application displays with no data, which is correct.

   e) Choose *Go* to send the get request.

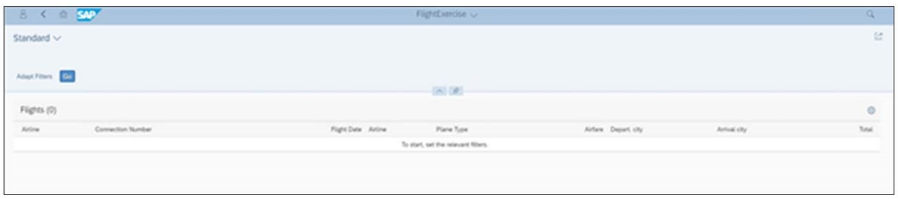


Figure 303: Go Button

f) The results are displayed in the table when the request is processed successfully.

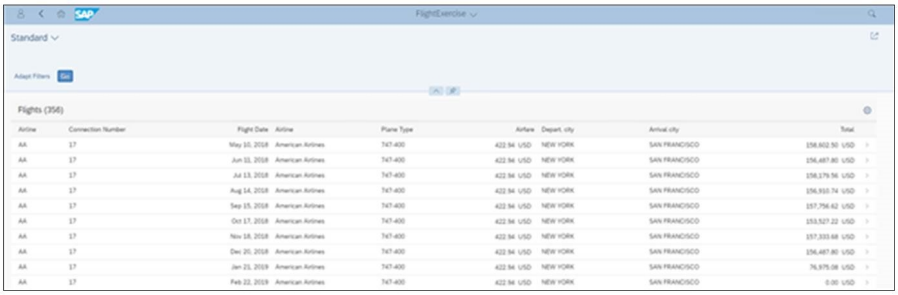


Figure 304: Results Table

Task 6: Add Responsive Behaviour to the List Report.

In the next task you will add responsive behaviour to your application.

1. Add the following annotations and information to the fields of the projection list.

Table 80: Projection List Field Information

Field	Term	Property	Value
planetype	lineItem	importance	#HIGH
Total	lineItem	importance	#MEDIUM
price	lineItem	importance	#LOW

a) Start *Eclipse* as shown in the first task of the exercise, if not already open.

b) Open the created *CDS* located in the package *ZTRAIN_##*, if not already displayed.

c) Add the noted terms to the fields of the projection list.

```
define view ZUX410_C_FLIGHTUI00 as select from sflight as sf
    association to scarr as sc on sf.carrid = sc.carrid
    association to sflights as sfs on sfs.carrid = sf.carrid and
    sfs.connid = sf.connid and sfs.fldate = sf.fldate
 {
@UI.lineItem.position:10
    key sf.carrid,
@UI.lineItem.position:20
    key sf.connid,
@UI.lineItem.position:30
    key sf.fldate,
@UI.lineItem.position:40
    sc.carrname,
@UI.lineItem.position:70
    sfs.cityfrom,
@UI.lineItem.position:80
    sfs.cityto,
@UI.lineItem.position:50
@UI.lineItem.importance: #HIGH
    sf.planetype,
@UI.lineItem.importance: #LOW
@UI.lineItem.position:60
    sf.price,
@Semantics.currencyCode: true
    sf.currency,
@Semantics.amount.currencyCode: 'currency'
@UI.lineItem.importance: #MEDIUM
@UI.lineItem.position:90
    sf.price * sf.seatsocc as Total,
    sf.seatsmax,
    sf.seatsocc,
    sc.url,
    sf.paymentsum,
    sf.seatsmax_b,
    sf.seatsocc_b,
    sf.seatsmax_f,
    sf.seatsocc_f
}
```

Figure 305: Add Terms to Define the View

d) Activate your changes by choosing  .

2. Test your application in the *SAP Web IDE*. Use the features of the Google Chrome Development Tools to change the form factor.

a) Open the *SAP Web IDE* as described in the previous task, if not already open.

b) Start your application as described in the previous task.

c) When your application starts, choose *F12* to access the development tools of *Google Chrome*. Your browser window looks as shown.

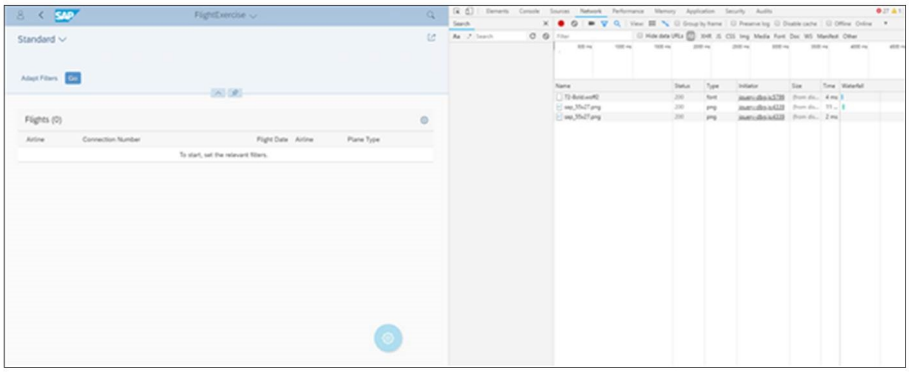


Figure 306: F12 Opens Development Tools

d) Choose the icon  to access the switch form factor feature.

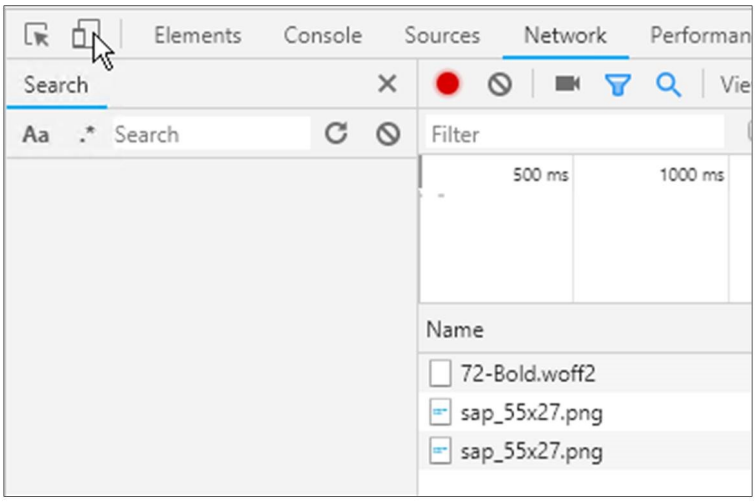


Figure 307: Icon on Toolbar

e) You now see the application switch-form-factor feature.

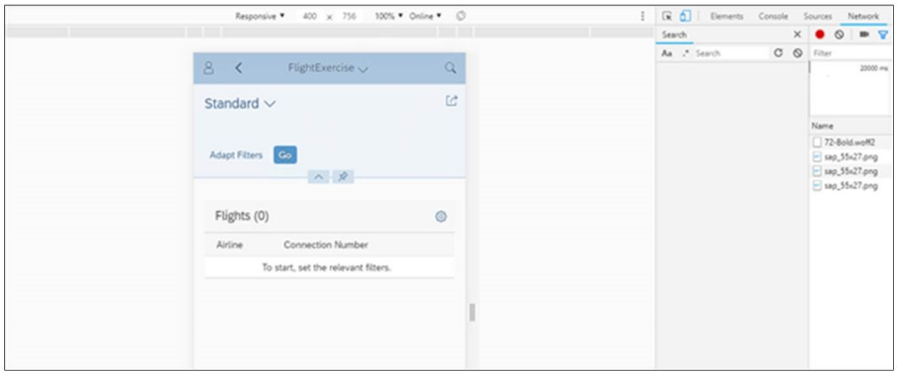


Figure 308: Switch Form Feature

f) Choose *Go* to trigger a get request.

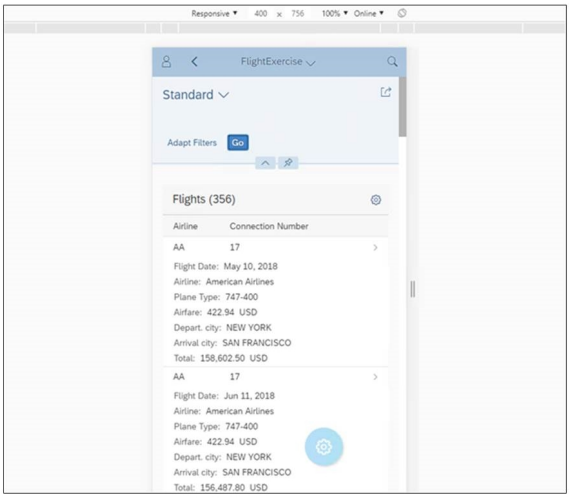


Figure 309: Go Button

g) Now switch the form factor from responsive to, for example, iPhone 5/SE.

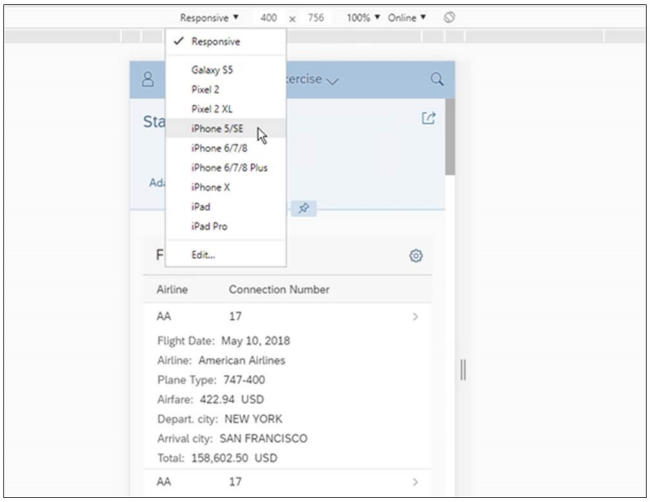


Figure 310: Choose Form

h) After you have switched form factor, reload your application by choosing *F5*.

i) Choose the *Go* button to trigger a new get request.

j) After the request is successfully completed, you see *planetype* is still displayed as importance *#HIGH*.

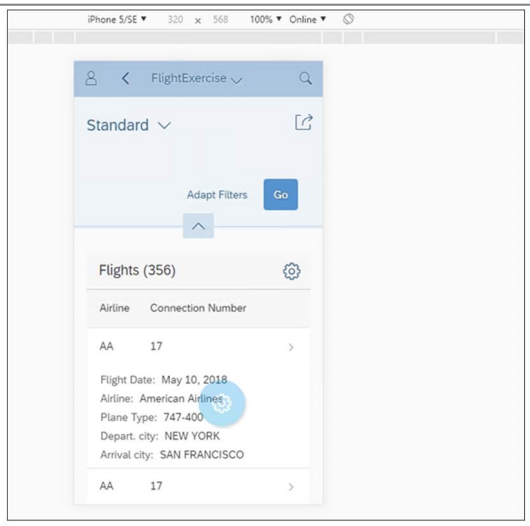


Figure 311: iPhone 5SE Screen

k) Close the current browser tab.

Unit 14
Exercise 16

Implement Search and Filter

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, step procedures and naming of fields and buttons may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

Note:
Sometimes after the annotations are changed in the ABAP CDS, they are not reflected in the *SAP Fiori Elements Application*, or in the *SAP Web IDE*. If this happens, clean the browser cache and restart the application.

The ABAP-CDS view, created in the previous exercise is required.

1. Activate the search capabilities in the ABAP-CDS view, created in the previous exercise. Add the search facility and assign the following information.

   Table 81: Search and Assign the Following Information

Attribute	Datatype	Value
searchable	boolean	true

2. Add an annotation that the field *carrname* is searchable. Therefore add `Search` to the *carrname* field of your projection list and assign the following information.

   Table 82: Assign the Information to the Carrname Field

Attribute	Datatype	Value
defaultSearchElement	boolean	true

3. Start your application to test whether your application now has search capabilities.

Task 1: Fuzzy Search

1. Add the fuzzy search behaviour to the *carrname* field of the *projectlist*. Use the property *fuzzinessThreshold* and assign the float value `0.7` to the property.
2. Test your application in the *SAP Web IDE*.

Task 2: Add filter options to the Filtering Application

1. Activate the fields *carrname*, *cityto*, and *cityfrom* as filter criteria in the application. Use the annotation selectionField for this.

2. To describe basic information of data you want to display, add filter options using the term **`UI.selectionField`** and the information provided in the table.

   Table 83: Basic Data Information

Field	Position
carrname	10
cityto	20
cityfrom	30

3. Test your application in the *SAP Web IDE*.

Unit 14
Solution 16

Implement Search and Filter

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, step procedures and naming of fields and buttons may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

Note:
Sometimes after the annotations are changed in the **ABAP CDS**, they are not reflected in the *SAP Fiori Elements Application*, or in the *SAP Web IDE*. If this happens, clean the browser cache and restart the application.

The ABAP-CDS view, created in the previous exercise is required.

1. Activate the search capabilities in the ABAP-CDS view, created in the previous exercise. Add the search facility and assign the following information.

   Table 81: Search and Assign the Following Information

Attribute	Datatype	Value
searchable	boolean	true

   a) Start eclipse if not already open.

   b) Open the created CDS (Core Data Services) located in the package ZTRAIN_##.

   c) Add the annotations to the ABAP-CDS.

[S4D] ZUX410_C_FLIGHTUI00

```
@AbapCatalog.sqlViewName: 'ZUX410FLIGHTUI00'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'UX410 Flight UI'
@OData.publish: true

@Search.searchable: true
@UI.headerInfo.typeName : 'Flight'
@UI.headerInfo.typeNamePlural : 'Flights'


define view ZUX410_C_FLIGHTUI00 as select from sflight as sf
    association to scarr as sc on sf.carrid = sc.carrid
    association to sflights as sfs on sfs.carrid = sf.carrid and
    sfs.connid = sf.connid and sfs.fldate = sf.fldate
 {
@UI.lineItem.position:10
    key sf.carrid,
@UI.lineItem.position:20
```

Figure 312: Add Annotations

2. Add an annotation that the field *carrname* is searchable. Therefore add **`Search`** to the *carrname* field of your projection list and assign the following information.

   Table 82: Assign the Information to the Carrname Field

Attribute	Datatype	Value
defaultSearchElement	boolean	true

   a) Add the annotation to the carrname field of your projection list.

*[S4D] ZUX410_C_FLIGHTUI00

```
@AbapCatalog.sqlViewName: 'ZUX410FLIGHTUI00'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'UX410 Flight UI'
@OData.publish: true

@Search.searchable: true
@UI.headerInfo.typeName : 'Flight'
@UI.headerInfo.typeNamePlural : 'Flights'


define view ZUX410_C_FLIGHTUI00 as select from sflight as sf
    association to scarr as sc on sf.carrid = sc.carrid
    association to sflights as sfs on sfs.carrid = sf.carrid and
    sfs.connid = sf.connid and sfs.fldate = sf.fldate
 {
@UI.lineItem.position:10
    key sf.carrid,
@UI.lineItem.position:20
    key sf.connid,
@UI.lineItem.position:30
    key sf.fldate,
@Search.defaultSearchElement: true
@UI.lineItem.position:40
    sc.carrname,
@UI.lineItem.position:70
    sfs.cityfrom,
@UI.lineItem.position:80
    sfs.cityto,
@UI.lineItem.position:50
@UI.lineItem.importance: #HIGH
    sf.planetype,
```

Figure 313: Add Annotation

   b) Activate your changes.

3. Start your application to test whether your application now has search capabilities.
   a) Start the *SAP Web IDE* as described in the previous exercise, if not already started.
   b) To test the application, select the project root folder and choose from the context menu *Run → Run an SAP Fiori Launchpad Sandbox*.

c) When the application starts, you see the *search* field and the *Go* button to trigger a get request.

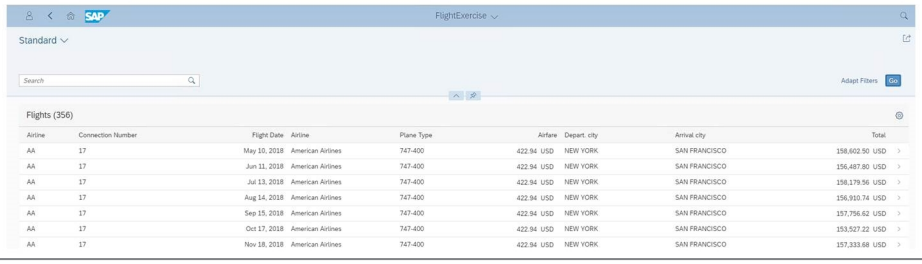


Figure 314: Search and Go

Note:
If the search field is not shown, clear the browser cache (hard reload).

d) Enter the name of a carrier in the search field and choose *Enter*.

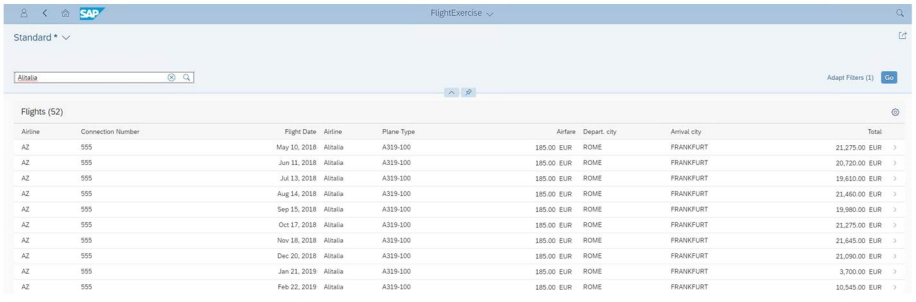


Figure 315: Enter Carrier Name

e) Add a little typo into the name of a carrier. You will get no search result.

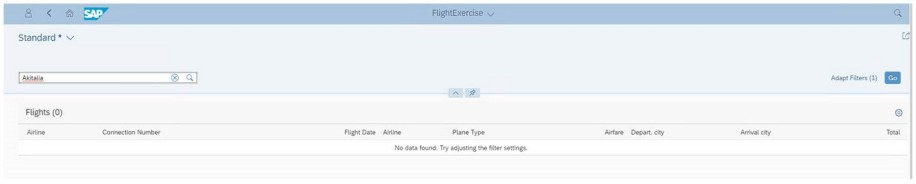


Figure 316: Test with a Typing Error

Task 1: Fuzzy Search

1. Add the fuzzy search behaviour to the *carrname* field of the *projectlist*. Use the property *fuzzinessThreshold* and assign the float value `0.7` to the property.

   a) Add the listed annotation to the *carrname* field of your project list.

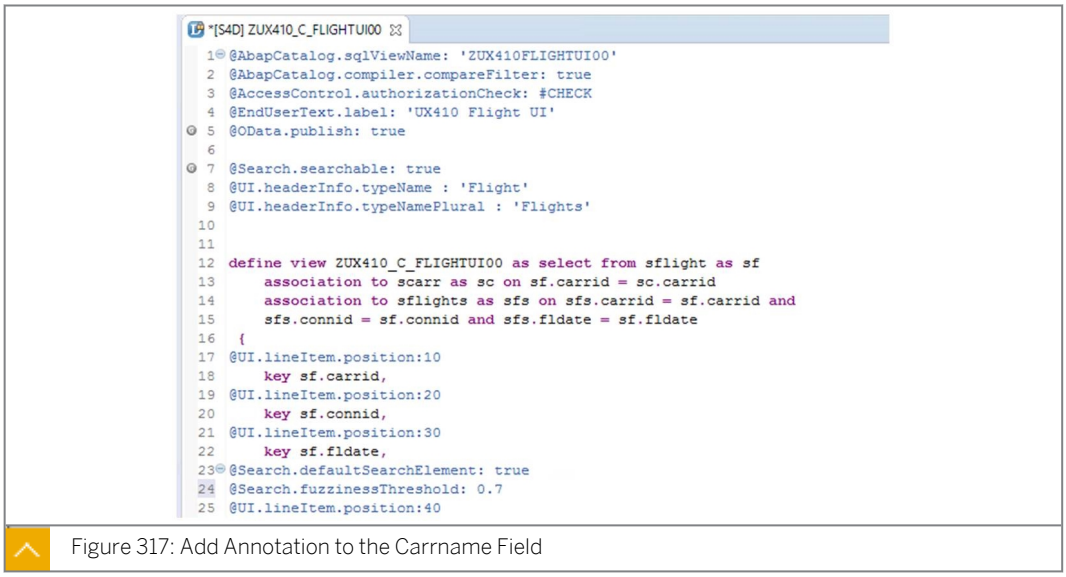


Figure 317: Add Annotation to the Carrname Field

b) Activate your changes.

2. Test your application in the *SAP Web IDE*.

a) Open the *SAP Web IDE*, if not already open.

b) Start your application as previously described.

c) Choose the *Go* button to trigger a get request.

d) Enter the value **`Akitalia`** in the search field and choose *Enter*.

e) Close the current browser tab.

Task 2: Add filter options to the Filtering Application

1. Activate the fields *carrname*, *cityto*, and *cityfrom* as filter criteria in the application. Use the annotation selectionField for this.

2. To describe basic information of data you want to display, add filter options using the term **`UI.selectionField`** and the information provided in the table.

Table 83: Basic Data Information

Field	Position
carrname	10
cityto	20
cityfrom	30

a) Add the UI.*selectionField* annotation to the field of the projectlist and assign the integer value to the property positions.

Figure 318: Add UI.selectionField Annotation

b) Activate your changes.

3. Test your application in the *SAP Web IDE*.

a) Open the *SAP Web IDE*, if not already open.

b) Start your application as previously described.

c) Choose the *Go* button to trigger a get request.

d) Click the *value help* symbol at the *Depart.city* selection field.

e) The *conditions* dialog for departure city will be opened.

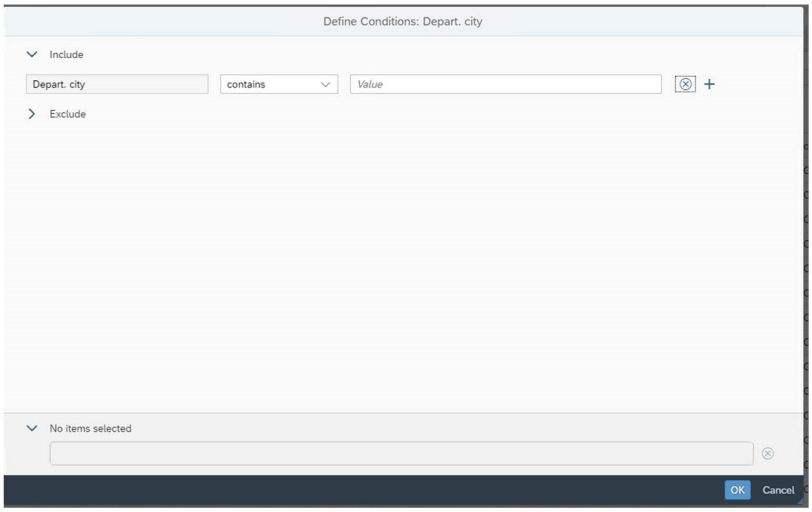


Figure 319: Depart City Dialog

f) Enter the value **SAN** into the *input* field and choose *OK*.

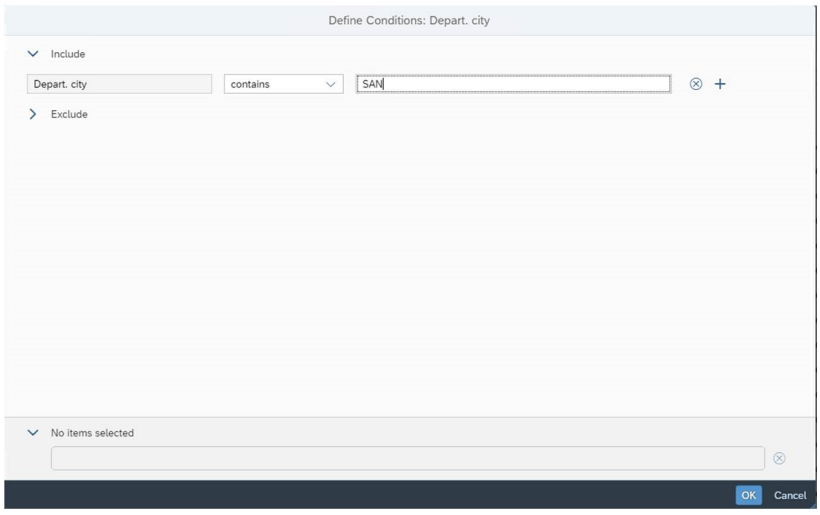


Figure 320: Enter SAN

g) Choose the *Go* button to apply the filter.

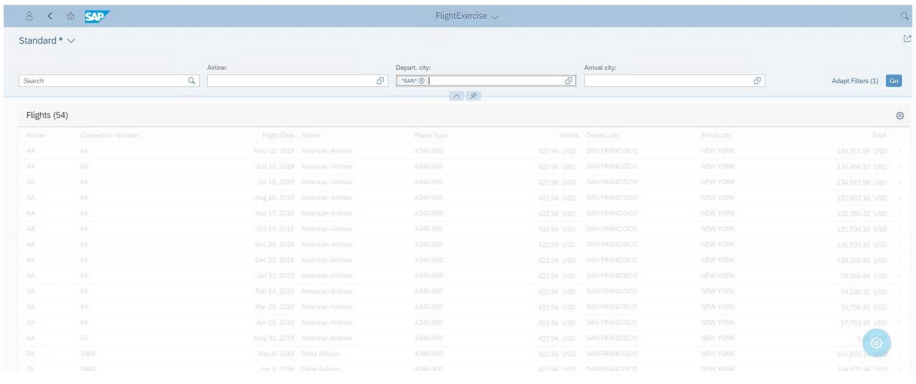


Figure 321: Apply the Filter

h) After the filter is applied, the list only shows flights with a departure city of SAN FRANSICO.

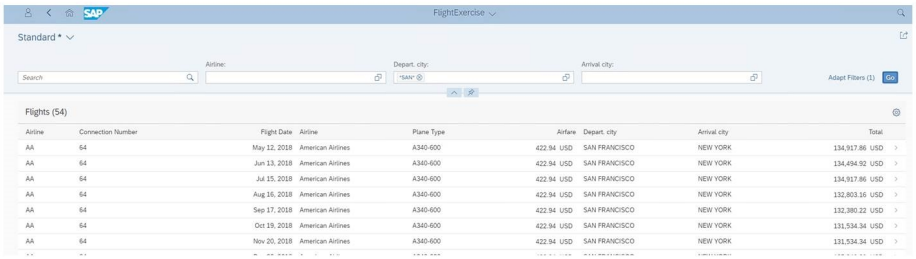


Figure 322: Displays Flights for San Francisco

i) Try testing other selections in the field.

Unit 14
Exercise 17

Implement Object Page SAP Fiori Elements

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, step procedures and names such as *buttons* and *fields* may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

Note:
In some cases the changes in the *ABAP CDS* are not directly reflected in the *SAP Fiori Elements Application*. If this happens, clean the browser cache and restart the application.

Implement Header Data

1. Set the title of every flight to the value of field *carrid*, and the description to the value of field *carrname*.
2. Test your application in the *SAP Web IDE Full-Stack*.

Task 1: Adding a HeaderFacet to the object page

1. Assign the following fields of the projection list to *fieldGroup* with qualifier *flightinfo*, and assign the position as listed in the table.

   Table 84: Assign Positions as Listed for fieldGroup with Qualifier flightinfo

Field	Position
cityfrom	10
cityto	20
url	30

2. Assign the following fields of the projection list to *fieldGroup* with qualifier *seatinfo*, and assign the positions as listed in the table.

   Table 85: Assign Flightifo

Field	Position
seatsmax	10
seatsocc	20

3. Add two facets of purpose #HEADER to your ABAP CDS. The first facet should reference the qualifier *flightinfo* as a target and the second should reference the qualifier *seatinfo* as a target

Task 2: Adding fields to the standard facet

1. Assign the annotation `@UI.identification` to the listed fields in your *ABAP-CDS* and assign the position as an integer. Test the result in the *SAP Web IDE*.

   Table 86: Assign to the Listed Fields in Your ABAP-CDS

Field	Position
carrname	10
flightdate	20
cityfrom	30
cityto	40
planetype	50

Task 3: Add Your Own Facets

1. Assign the following fields of the projection list to a *fieldGroup* with a qualifier *seatdata*, and assign the position as listed in the table.

   Table 87: Assign Fields

Field	Position
seatsmax_b	10
seatsmax_f	20

2. Assign the fields of the projection list to a *fieldGroup* with qualifier *seatdata2*, and assign the positions as listed in the table. Then activate your changes.

   Table 88: Assign Fields

Field	Position
seatsocc_b	10
seatsocc_f	20

3. Add two facets of purpose #STANDARD and type #FIELDGROUP_REFERENCE to your @UI.facet-implementation. The first should reference the targetQualifier seatdata and the second the targetQualifier seatdata2.
4. In the local annotation file of your SAP Web IDE project, add the property label with the following values to the two new UI.ReferenceFacet-entry.

Property	Value
ReferenceFacet for qualifier seatdata	Seatinfo1

Property	Value
ReferenceFacet for qualifier seatdata	Seatinfo2

Note:
You can also add the text to the resource files but, for the sake of simplicity, we are not working with i18n resources.

Implement Object Page SAP Fiori Elements

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, step procedures and names such as *buttons* and *fields* may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

Note:
In some cases the changes in the *ABAP CDS* are not directly reflected in the *SAP Fiori Elements Application*. If this happens, clean the browser cache and restart the application.

Implement Header Data

1. Set the title of every flight to the value of field *carrid*, and the description to the value of field *carrname*.
   a) Open *Eclipse* as described in the previous exercise, if not already open.
   b) Open the *ABAP CDS ZUX410_C_FLIGHTUI##* from your favourite package *ZTRAIN_##*, if not already open.
   c) Add title and description annotation to the *ABAP CDS* and assign the fields *carrid* to the title and *carrname* to the description.

```
[S4D] ZUX410_C_FLIGHTUI00
1  @AbapCatalog.sqlViewName: 'ZUX410FLIGHTUI00'
2  @AbapCatalog.compiler.compareFilter: true
3  @AccessControl.authorizationCheck: #CHECK
4  @EndUserText.label: 'UX410 Flight UI'
5  @OData.publish: true
6
7  @Search.searchable: true
8  @UI.headerInfo.typeName : 'Flight'
9  @UI.headerInfo.typeNamePlural : 'Flights'
10 @UI.headerInfo.title.value: 'carrid'
11 @UI.headerInfo.description.value: 'carrname'
12
13
14 define view ZUX410_C_FLIGHTUI00 as select from sflight as sf
15     association to scarr as sc on sf.carrid = sc.carrid
16     association to sflights as sfs on sfs.carrid = sf.carrid and
17     sfs.connid = sf.connid and sfs.fldate = sf.fldate
18  {
```

Figure 323: Title and Description Annotation

   d) Activate your changes .

2. Test your application in the *SAP Web IDE Full-Stack*.
   a) Open the *SAP Web IDE Full-Stack* as described in previous exercises, if not already open.
   b) Start the application created in the previous exercise.
   c) Choose *Go* on the filter bar to trigger a get-request.
   d) Select a flight from the list.

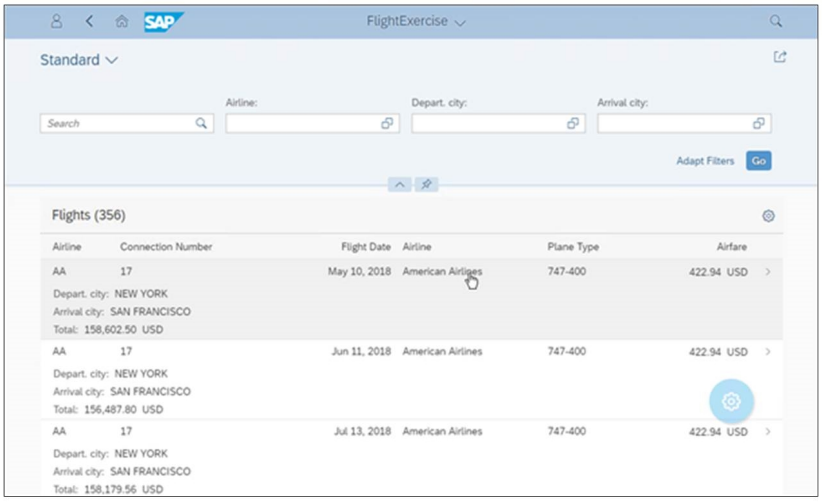


Figure 324: Select a Flight

   e) You now see the *carrid* and *carrname* in the object page.

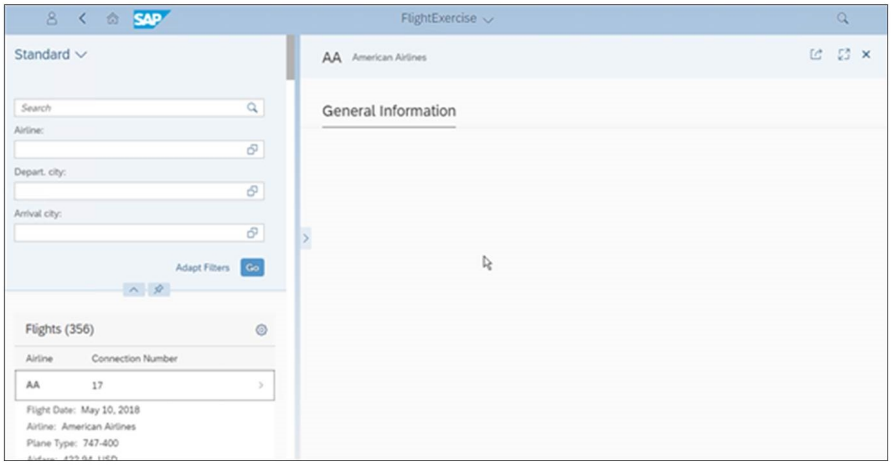


Figure 325: Information Displayed

Task 1: Adding a HeaderFacet to the object page

1. Assign the following fields of the projection list to *fieldGroup* with qualifier *flightinfo*, and assign the position as listed in the table.

Table 84: Assign Positions as Listed for fieldGroup with Qualifier flightinfo

Field	Position
cityfrom	10
cityto	20

Field	Position
url	30

a) Open *Eclipse* as described in previous exercises, if not already open.

b) Open the *ABAP CDS ZUX410_C_FLIGHTUI##*, if not already open.

c) Add **`fieldGroup`** annotation with the qualifier **`flightinfo`** to the listed fields and add the position.

```
@UI.fieldGroup : [
  {
    qualifier:'flightinfo', position:10
  }
]
@UI.selectionField.position:20
@UI.lineItem.position:70
    sfs.cityfrom,

@UI.fieldGroup : [
  {
    qualifier:'flightinfo', position:20
  }
]
@UI.selectionField.position:30
@UI.lineItem.position:80
    sfs.cityto,
@UI.lineItem.position:50
    sf.planetype,
@UI.lineItem.importance: #LOW
    sf.price,
@Semantics.currencyCode: true
    sf.currency,
@Semantics.amount.currencyCode: 'currency'
    sf.price * sf.seatsocc as Total,
    sf.seatsmax,
    sf.seatsocc,

@UI.fieldGroup : [
  {
  qualifier:'flightinfo', position:30
  }
]
```

Figure 326: Field Group Annotation

2. Assign the following fields of the projection list to *fieldGroup* with qualifier *seatinfo*, and assign the positions as listed in the table.

Table 85: Assign Flightifo

Field	Position
seatsmax	10
seatsocc	20

a) Add the *fieldGroup* annotation with the qualifier *seatinfo* to the fields from the table, and assign the listed positions.

```
48  @UI.lineItem.position:50
50      sf.planetype,
51  @UI.lineItem.importance: #LOW
53      sf.price,
54  @Semantics.currencyCode: true
55      sf.currency,
56  @Semantics.amount.currencyCode: 'currency'
59      sf.price * sf.seatsocc as Total,
60
61  @UI.fieldGroup : [
62      {
63          qualifier:'seatinfo', position:10
64      }
65  ]
66      sf.seatsmax,
67  @UI.fieldGroup : [
68      {
69          qualifier:'seatinfo', position:20
70      }
71  ]
72      sf.seatsocc,
```

Figure 327: Add Annotation

b) Activate your changes  .

3. Add two facets of purpose #HEADER to your ABAP CDS. The first facet should reference the qualifier *flightinfo* as a target and the second should reference the qualifier *seatinfo* as a target

a) Open the ABAP CDS ZUX410_C_FLIGHTUI#, if not already open.

b) Add the following facet-implementation at the beginning of the projection list implementation of your ABAP CDS.

```
@UI.facet: [
  {   purpose:#HEADER, type: #FIELDGROUP_REFERENCE, targetQualifier: 'flightinfo', position:10 },
  {   purpose:#HEADER, type: #FIELDGROUP_REFERENCE, targetQualifier: 'seatinfo', position:20 },
  {   purpose:#STANDARD, type: #FIELDGROUP_REFERENCE, targetQualifier: 'seatdata', position:10},
  {   purpose:#STANDARD, type: #FIELDGROUP_REFERENCE, targetQualifier: 'seatdata2', position:20},
  {   purpose:#STANDARD, id: 'seat1', label:'Seatsinfo 1', type:#COLLECTION, position:30},
  {   purpose:#STANDARD, id: 'seat2', label:'Seatsinfo 2', type:#COLLECTION, position:40}
]
```

Figure 328: First Facet

c) Add the second facet to your implementation.

```
23 {
24
25      @UI.facet: [
26        { purpose: #HEADER,type: #FIELDGROUP_REFERENCE, targetQualifier: 'flightinfo', position:10 },
27        { purpose: #HEADER,type: #FIELDGROUP_REFERENCE, targetQualifier: 'seatinfo', position:20 }]
28
29      @UI.lineItem.position:10
30  key sf.carrid,
31      @UI.lineItem.position:20
32  key sf.connid,
```

Figure 329: Second Facet

d) Save and activate your changes.

e) Start the application as described in previous exercises.

f) Choose *Go* to trigger a get request and select a flight from the list.

g) You will now see the new HeaderFacet.

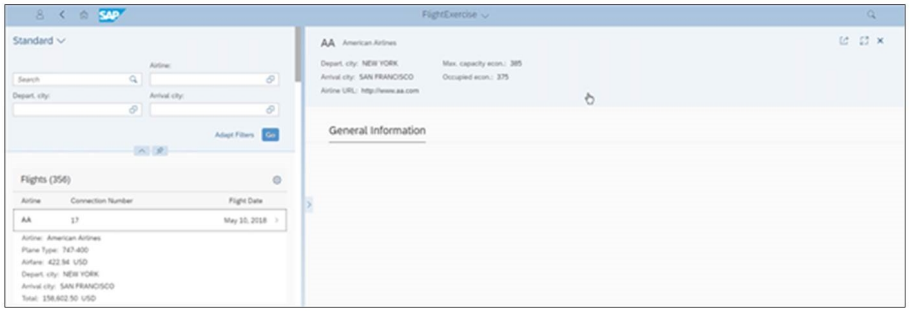


Figure 330: HeaderFacet

Task 2: Adding fields to the standard facet

1. Assign the annotation `@UI.identification` to the listed fields in your *ABAP-CDS* and assign the position as an integer. Test the result in the *SAP Web IDE*.

   Table 86: Assign to the Listed Fields in Your ABAP-CDS

Field	Position
carrname	10
flightdate	20
cityfrom	30
cityto	40
planetype	50

   a) Open the *ABAP CDS ZUX410_C_FLIGHTUI#*, if not already open.

   b) Add `@UI.identification` annotation to the field shown in the table and add the *position* property with the listed integer value.

Figure 331: Add Annotation

   c) Activate your changes  .

d) Start the *SAP Web IDE* as described in previous exercises, if not already started.

e) Close the `annotations.xml` file if open and reopen it again using the *Annotation Modeler*.

f) Expand the `UI.Facets` node in the *Local Annotations* and ensure the `@UI.identification` annotation is shown.

g) Start the application as described in previous exercises.

h) Choose the *Go* button and select a flight from the list.

i) The General Information facet of the object page now shows the annotated fields.

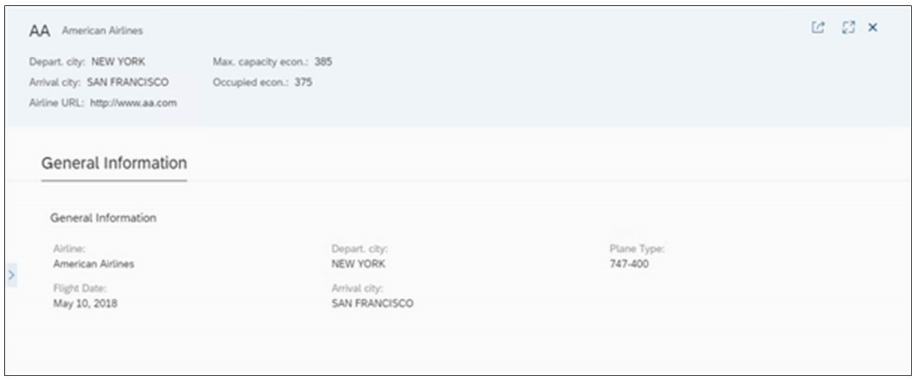


Figure 332: General Information

Task 3: Add Your Own Facets

1. Assign the following fields of the projection list to a *fieldGroup* with a qualifier *seatdata*, and assign the position as listed in the table.

Table 87: Assign Fields

Field	Position
seatsmax_b	10
seatsmax_f	20

a) Start *Eclipse* if not already started.

b) Open *ABAP CDS ZUX410_C_FLIGHTUI##*.

c) Add the *fieldGroup* annotation to the fields of your projection list and assign the positions.

```
89  @UI.fieldGroup : [
90    {qualifier:'seatdata', position:10}
91  ]
92      sf.seatsmax_b,
93      sf.seatsocc_b,
94  @UI.fieldGroup : [
95    {qualifier:'seatdata', position:20}
96  ]
97      sf.seatsmax_f,
98      sf.seatsocc_f
99  }
```

Figure 333: Add Annotation

2. Assign the fields of the projection list to a *fieldGroup* with qualifier *seatdata2*, and assign the positions as listed in the table. Then activate your changes.

Table 88: Assign Fields

Field	Position
seatsocc_b	10
seatsocc_f	20

a) Add the *fieldGroup* annotation to the fields of your projection list and assign the positions.

```
89  @UI.fieldGroup : [
90    {qualifier:'seatdata', position:10}
91  ]
92      sf.seatsmax_b,
93  @UI.fieldGroup : [
94    {qualifier:'seatdata2', position:10}
95  ]
96      sf.seatsocc_b,
97  @UI.fieldGroup : [
98    {qualifier:'seatdata', position:20}
99  ]
100     sf.seatsmax_f,
101 @UI.fieldGroup : [
102   {qualifier:'seatdata2', position:10}
103 ]
104     sf.seatsocc_f
105 }
```

Figure 334: Assign Position

b) Activate your changes  .

3. Add two facets of purpose #STANDARD and type #FIELDGROUP_REFERENCE to your @UI.facet-implementation. The first should reference the targetQualifier seatdata and the second the targetQualifier seatdata2.

a) Extend the @UI.facet-implementation as shown in the next figure.

```
@UI.facet: [
  { purpose: #HEADER,type: #FIELDGROUP_REFERENCE, targetQualifier: 'flightinfo', position:10 },
  { purpose: #HEADER,type: #FIELDGROUP_REFERENCE, targetQualifier: 'seatinfo', position:20 },
  { purpose: #STANDARD, type: #FIELDGROUP_REFERENCE, targetQualifier: 'seatdata', position:10},
  { purpose: #STANDARD, type: #FIELDGROUP_REFERENCE, targetQualifier: 'seatdata2', position:20},
  {purpose:#STANDARD, id:'seat1', label:'Info1', type:#COLLECTION,position:30, label: 'Seatsinfo 1'} ,
   {purpose:#STANDARD, id:'seat2', label:'Info2', type:#COLLECTION,position:40, label:'Seatsinfo 2'}
]
```

Figure 335: Extend the @UI.facet-implementation

b) Save and activate your changes.

4. In the local annotation file of your SAP Web IDE project, add the property label with the following values to the two new UI.ReferenceFacet-entry.

Property	Value
ReferenceFacet for qualifier seatdata	Seatinfo1
ReferenceFacet for qualifier seatdata	Seatinfo2

Note:
You can also add the text to the resource files but, for the sake of simplicity, we are not working with i18n resources.

a) Open the SAP Web IDE if not already opened.

b) Open the *annotation.xml* file located in the annotations folder of your project.

c) Expand the UI.Facets entry of your local annotations.

d) Click the +-button on the level of the UI.ReferenceFacet that points to the target *@UI.FieldGroup#seatdata*.

e) Choose *Label* in the upcoming dialog and choose *OK*.

f) Insert the value from the table into the value field of the label.

g) Repeat the steps d to f for the *ReferenceFacet* with target *@UI.FieldGroup#seatdata2*.

h) Save your changes.

i) Start the application as described in previous exercises.

j) Choose the *Go* button of the *FacetFilter* and choose a flight.

k) You will now see the newly added facets.

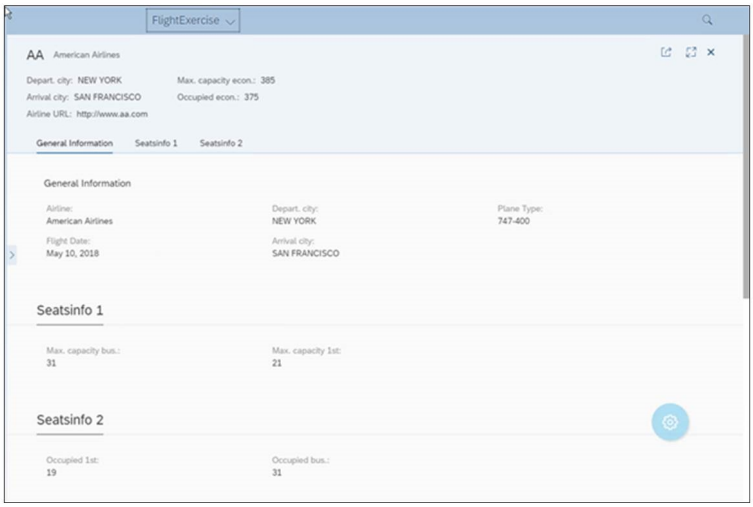


Figure 336: New Facet

Unit 14
Exercise 18

Display Dependent Entities as SAP Fiori Elements

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, step procedures and naming of fields and buttons may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

Note:
Sometimes after the annotations are changed in the ABAP CDS, they are not reflected in the *SAP Fiori Elements Application*, or in the *SAP Web IDE*. When this happens, clean the browser cache and restart the application.

Implement a *CDS* view for the selection of bookings.

1. Implement a *CDS* view for the selection of bookings.

   Table 89: ABAP CDS Configuration

Property	Value
Name	ZUX410_C_BOOKINGUI00## (replace ## with your group number)
Description	Flight bookings
Package	ZTRAIN_## (replace ## with your group number)
Transport request	Use the request assigned to your group by your instructor
Template for creating New Data Definition	No template

2. Copy the content of the file `S:\Courses\UX410_20\Templates\ZUX410_C_BookingUI##.txt` to the newly created *ABAP CDS* view and replace the ## in the copied content with your group number.

Task 1: Add an association to `ZUX410_C_FLIGHTUI##`.

1. Extend the `ZUX410_C_FLIGHT##-ABAP CDS` as shown in the figure and expose the `_Bookings` alias. Remember to use your own group number.

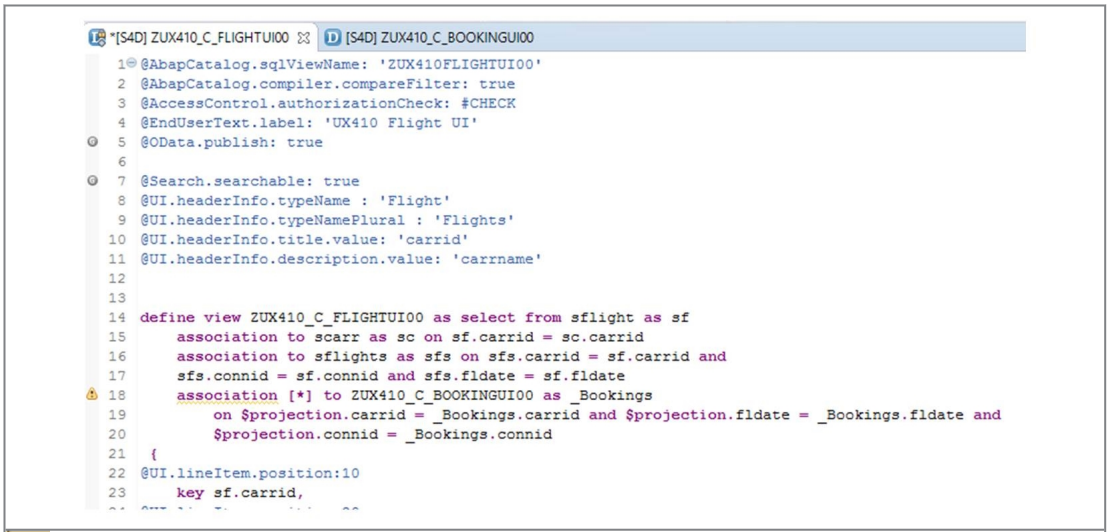


Figure 339: Extend ZUX410_C_FLIGHT##-ABAP CDS

Task 2: Add the bookings to the object page as a facet.

1. Add the `to_bookings` navigation property generated by SADL to your local annotations. Add `to_bookings` as a *ReferenceFacet* to the existing *UI.Facets* annotation. Add a property label to the newly created *ReferenceFacet* and assign the text **`Bookings`** to the *Label* property.
2. Test your application.

Unit 14
Solution 18

Display Dependent Entities as SAP Fiori Elements

Note:
SAP Cloud Platform and its tools are cloud services. Due to the nature of cloud software, step procedures and naming of fields and buttons may differ from the exercise solution. The following figures show the application of student ## and group 00. Use your own student and group number assigned by your instructor for the exercises.

Note:
Sometimes after the annotations are changed in the ABAP CDS, they are not reflected in the *SAP Fiori Elements Application*, or in the *SAP Web IDE*. When this happens, clean the browser cache and restart the application.

Implement a *CDS* view for the selection of bookings.

1. Implement a *CDS* view for the selection of bookings.

   Table 89: ABAP CDS Configuration

Property	Value
Name	ZUX410_C_BOOKINGUI00## (replace ## with your group number)
Description	Flight bookings
Package	ZTRAIN_## (replace ## with your group number)
Transport request	Use the request assigned to your group by your instructor
Template for creating New Data Definition	No template

   a) Start *Eclipse*, as described in the exercise , if not already started.

   b) Select your favorite package `ZTRAIN_##` in the *Project Explorer of Eclipse* and from the context menu, choose *New* → *Other ABAP Repository Object*.

   c) Expand the Core Data Service folder in the new dialog and choose the entry Data Definition and then *Next*.

   d) Enter the listed attributes from the table in the following dialog and choose *Next*.

e) Select a transport request and choose *Next*.

f) In the next dialog, ensure the checkbox *Use the selected template* is deselected and choose *Finish*.

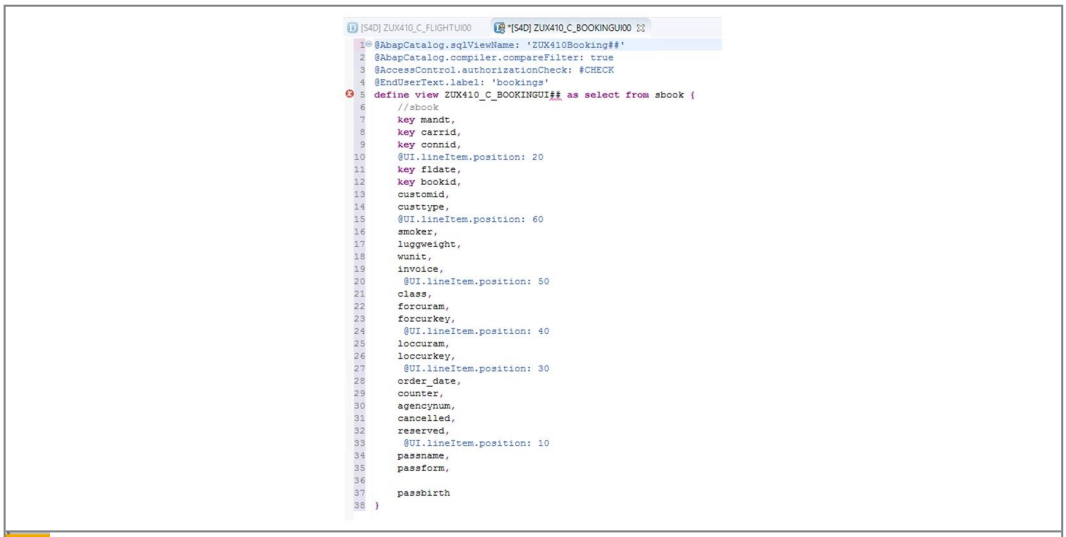
2. Copy the content of the file `S:\Courses\UX410_20\Templates\ZUX410_C_BookingUI##.txt` to the newly created *ABAP CDS* view and replace the ## in the copied content with your group number.

a) Open *Windows file explorer* and navigate to the file `S:\Courses\UX410_20\Templates\ZUX410_C_BookingUI##.txt`.

b) Open the file `ZUX410_C_BookingUI##.txt` and copy the content.

c) Go back to *Eclipse* and paste the copied content to the newly created *ABAP CDS*.

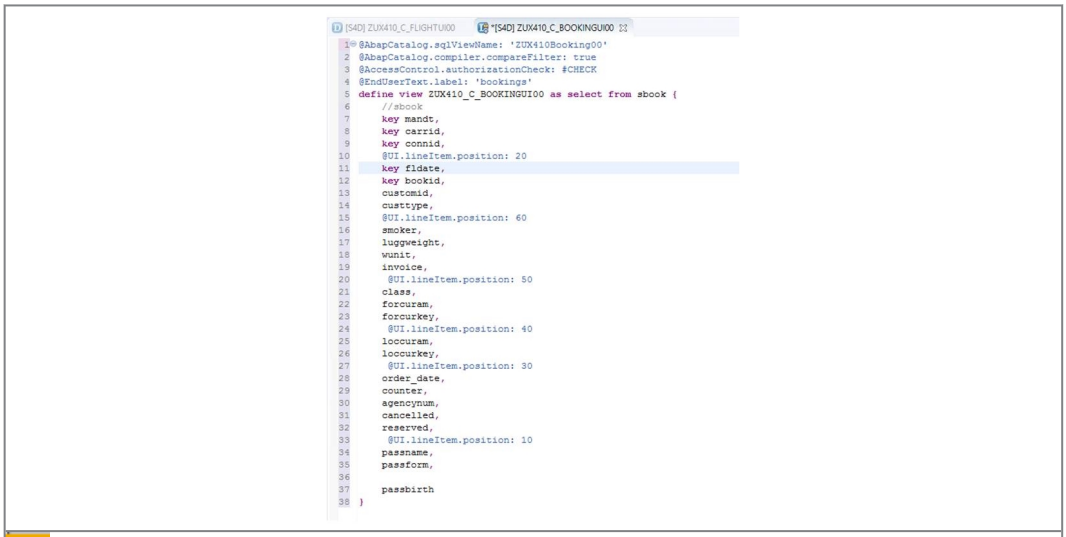
d) Your *ABAP CDS* will look as shown.



```
@AbapCatalog.sqlViewName: 'ZUX410Booking##'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'bookings'
define view ZUX410_C_BOOKINGUI## as select from sbook {
    //sbook
    key mandt,
    key carrid,
    key connid,
    @UI.lineItem.position: 20
    key fldate,
    key bookid,
    customid,
    custtype,
    @UI.lineItem.position: 60
    smoker,
    luggweight,
    wunit,
    invoice,
     @UI.lineItem.position: 50
    class,
    forcuram,
    forcurkey,
     @UI.lineItem.position: 40
    loccuram,
    loccurkey,
     @UI.lineItem.position: 30
    order_date,
    counter,
    agencynum,
    cancelled,
    reserved,
     @UI.lineItem.position: 10
    passname,
    passform,

    passbirth
}
```

Figure 337: ABAP CDS

e) Replace the ## with your group number. For group 00 the result will look as shown.



```
@AbapCatalog.sqlViewName: 'ZUX410Booking00'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'bookings'
define view ZUX410_C_BOOKINGUI00 as select from sbook {
    //sbook
    key mandt,
    key carrid,
    key connid,
    @UI.lineItem.position: 20
    key fldate,
    key bookid,
    customid,
    custtype,
    @UI.lineItem.position: 60
    smoker,
    luggweight,
    wunit,
    invoice,
     @UI.lineItem.position: 50
    class,
    forcuram,
    forcurkey,
     @UI.lineItem.position: 40
    loccuram,
    loccurkey,
     @UI.lineItem.position: 30
    order_date,
    counter,
    agencynum,
    cancelled,
    reserved,
     @UI.lineItem.position: 10
    passname,
    passform,

    passbirth
}
```

Figure 338: Result for Group 00

f) Activate your changes.

Task 1: Add an association to `ZUX410_C_FLIGHTUI##`.

1. Extend the `ZUX410_C_FLIGHT##-ABAP CDS` as shown in the figure and expose the `_Bookings` alias. Remember to use your own group number.

```
*[S4D] ZUX410_C_FLIGHTUI00   [S4D] ZUX410_C_BOOKINGUI00
1  @AbapCatalog.sqlViewName: 'ZUX410FLIGHTUI00'
2  @AbapCatalog.compiler.compareFilter: true
3  @AccessControl.authorizationCheck: #CHECK
4  @EndUserText.label: 'UX410 Flight UI'
5  @OData.publish: true
6
7  @Search.searchable: true
8  @UI.headerInfo.typeName : 'Flight'
9  @UI.headerInfo.typeNamePlural : 'Flights'
10 @UI.headerInfo.title.value: 'carrid'
11 @UI.headerInfo.description.value: 'carrname'
12
13
14 define view ZUX410_C_FLIGHTUI00 as select from sflight as sf
15     association to scarr as sc on sf.carrid = sc.carrid
16     association to sflights as sfs on sfs.carrid = sf.carrid and
17     sfs.connid = sf.connid and sfs.fldate = sf.fldate
18     association [*] to ZUX410_C_BOOKINGUI00 as _Bookings
19         on $projection.carrid = _Bookings.carrid and $projection.fldate = _Bookings.fldate and
20         $projection.connid = _Bookings.connid
21  {
22 @UI.lineItem.position:10
23     key sf.carrid,
```

Figure 339: Extend ZUX410_C_FLIGHT##-ABAP CDS

a) Open the implementation of `ZUX410_C_FLIGHTUI##` and add the implementation as shown in the figure.

b) Expose the `_Bookings` alias.

```
100 @UI.fieldGroup : [
101   {qualifier:'seatdata', position:20}
102 ]
103     sf.seatsmax_f,
104 @UI.fieldGroup : [
105   {qualifier:'seatdata2', position:10}
106 ]
107     sf.seatsocc_f,
108     _Bookings
109 }
```

Figure 340: Expose the Alias

c) Activate your changes.

Task 2: Add the bookings to the object page as a facet.

1. Add the `to_bookings` navigation property generated by SADL to your local annotations. Add `to_bookings` as a *ReferenceFacet* to the existing *UI.Facets* annotation. Add a property label to the newly created *ReferenceFacet* and assign the text **`Bookings`** to the *Label* property.

   a) Start the *SAP Web IDE Full-Stack* if not already started, as described in previous exercises.

   b) Close the `annotations.xml` file If already open, and open it again to ensure the new annotations from the *ABAP CDS* are fetched.

c) Navigate to *Local Annotations* and click on the + symbol on the level of the already existing *UI.Facets* entry.

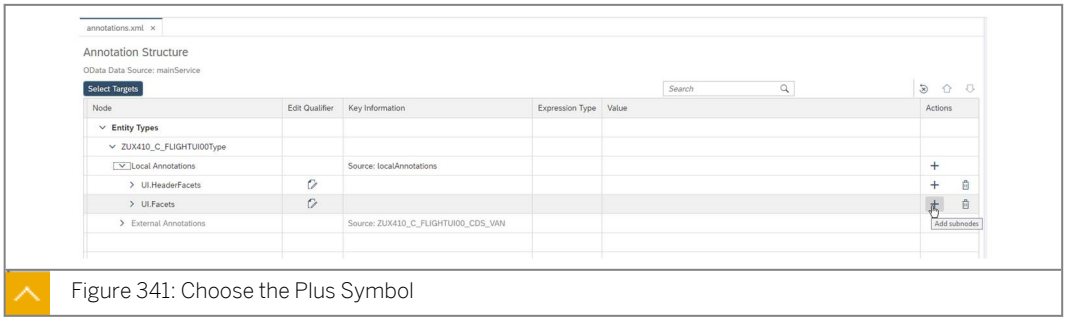


Figure 341: Choose the Plus Symbol

d) Check the checkbox *ReferenceFacet* in the new dialog and choose *OK*.

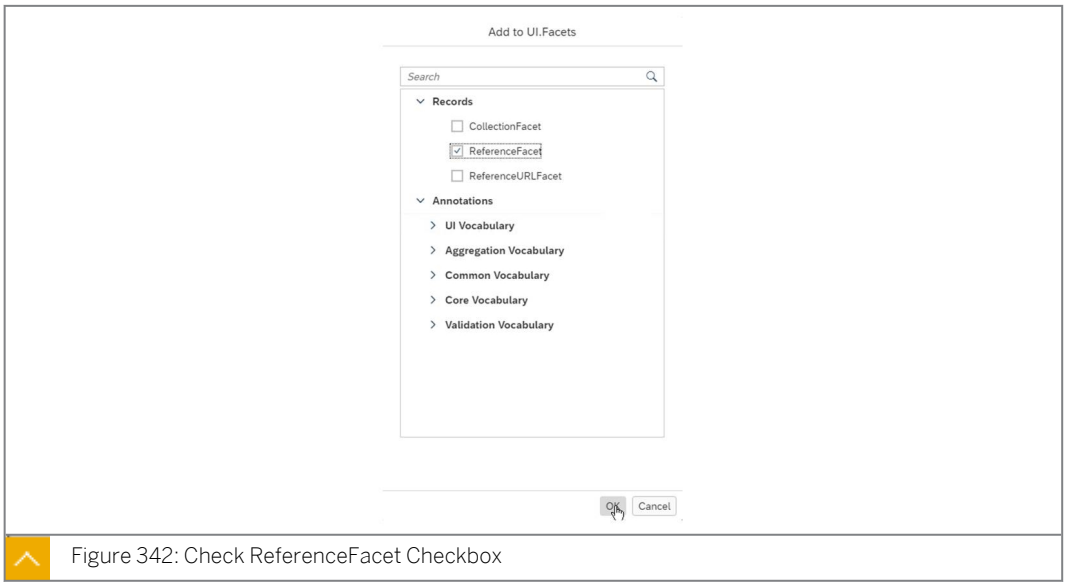


Figure 342: Check ReferenceFacet Checkbox

e) Choose the *F4* help symbol on the *Target* entry.

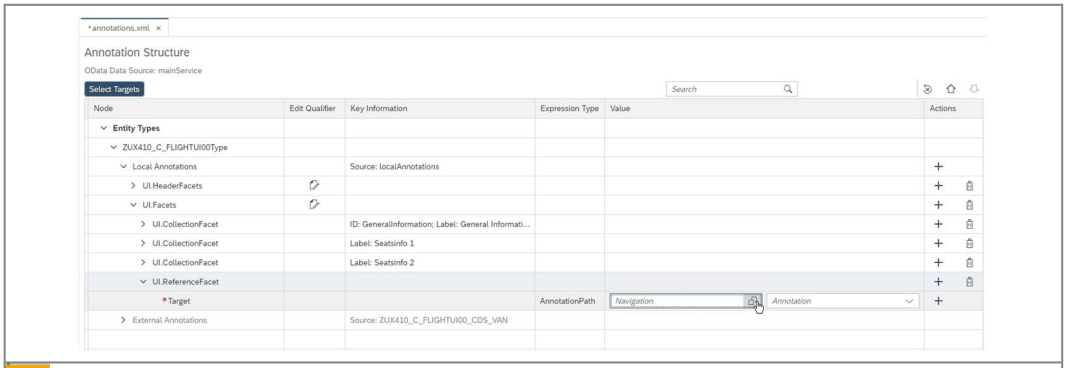


Figure 343: Choose Help Symbol

f) Select the `to_Bookings` entry in the new dialog and choose *OK*.

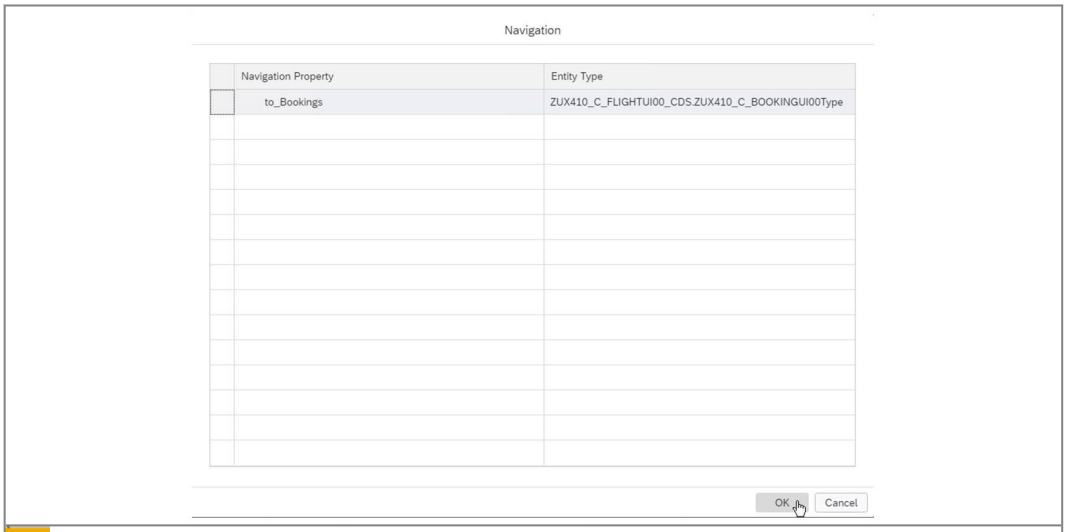


Figure 344: Select Entry and Choose OK

g) Your annotation should look as shown when the dialog is closed.

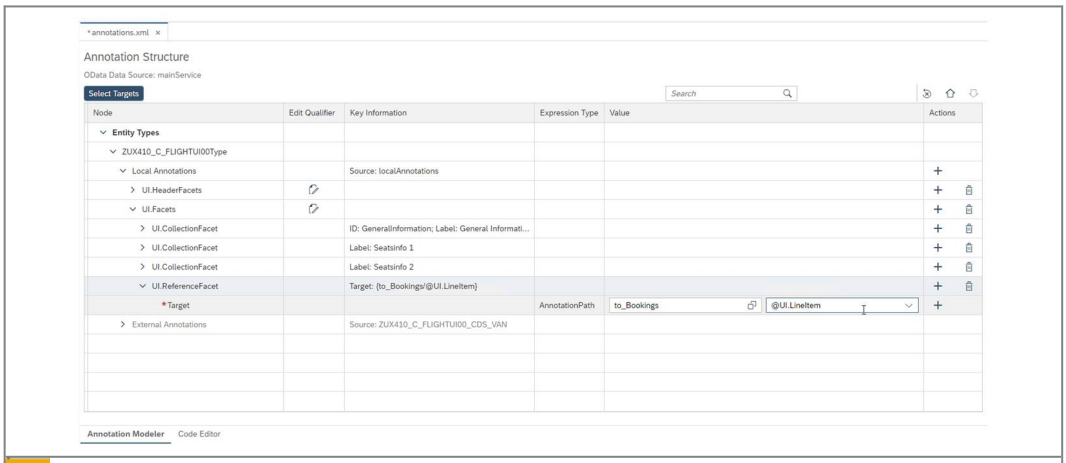


Figure 345: Annotation Displayed

h) Click the + symbol on the level of the currently changed *UI.ReferenceFacet*.

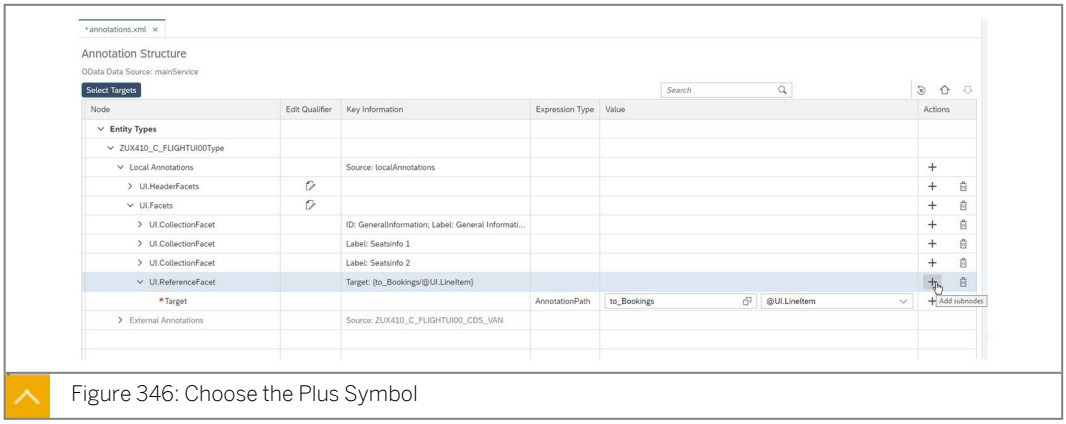


Figure 346: Choose the Plus Symbol

i) Select the checkbox beside the *Label* property in the following dialog and choose *OK*.

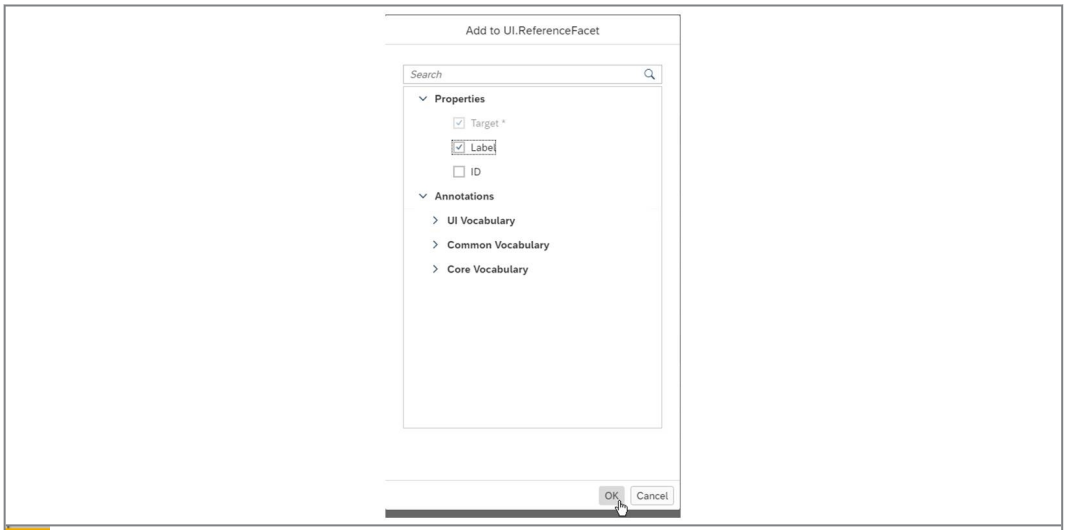


Figure 347: Label Property Checkbox

j) Enter the value **`Bookings`** in the *Value* field of the newly added property.

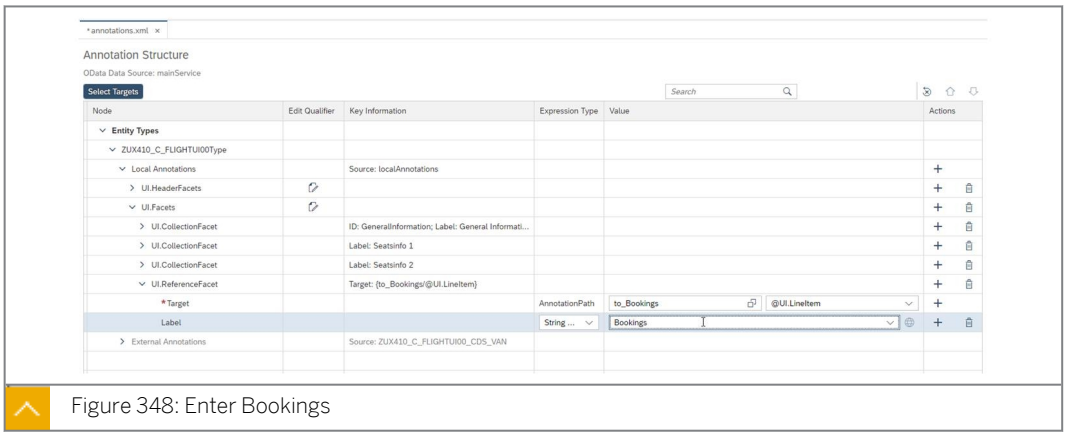


Figure 348: Enter Bookings

k) Save your changes.

2. Test your application.

a) Start your application as described in previous exercises.

b) Choose the *Go* button in the *FilterBar* to trigger a get-request.

c) Select a flight from the list.

d) Select the *Bookings* section to view your result.

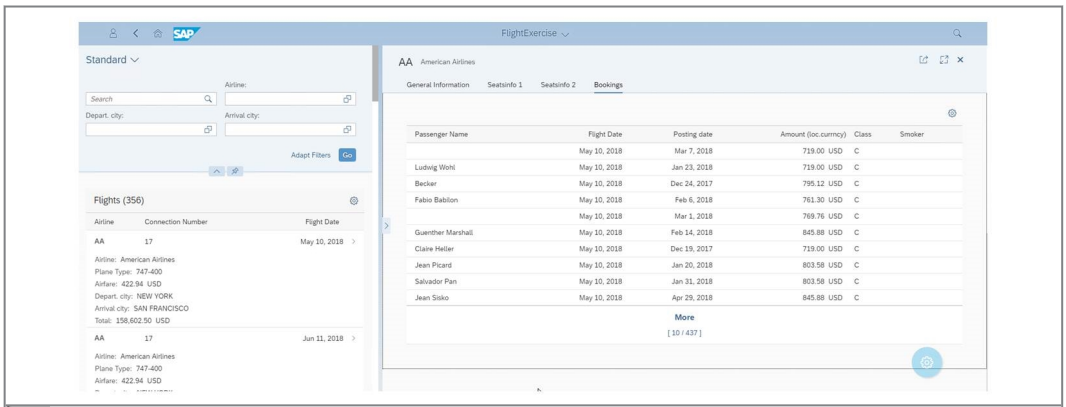


Figure 349: Bookings Section