Freestyle application give you full flexibility in terms of designing and implementing the UI of your application. In Chapter 7: Developing a Freestyle Application, we’ll show you how to develop a freestyle application step by step using the SAP Web IDE full-stack. We’ll show you all the necessary steps required to set up a freestyle application project, connect to an OData service, set up in-app navigation, and develop the UI using smart controls.

“Developing a Freestyle Application”

Contents

Index

The Authors

Stefan Haas, Bince Mathew

ABAP RESTful Programming Model: ABAP Development for SAP S/4HANA

563 pages, 2020, $79.95


www.sap-press.com/4988
Chapter 7
Developing a Freestyle Application

In this chapter, you’ll learn how to develop a freestyle SAPUI5 application using smart controls and the SAP Web IDE full-stack. Developing a freestyle application might be required if no SAP Fiori elements application type fits your specific use case and UI design.

In previous chapters, you learned how to implement several different SAP Fiori element template applications. These template-based applications significantly decrease the amount of required frontend JavaScript and SAPUI5 code and provide several generically configurable and extendable templates for different application scenarios. However, in some scenarios, the templates may not fit the UI design and mockups you developed in the design phase. In this case, you must develop a freestyle SAPUI5 application. Luckily, with the introduction of the SAP Fiori elements application templates, several smart controls were introduced to implement the SAP Fiori elements concept on a lower level while also being used inside the SAP Fiori element templates. Smart controls use OData annotations to reduce the required JavaScript frontend code but still provide the flexibility to use a freestyle application layout. In this chapter, you’ll get to know the most common smart controls. After an introduction to each control, we’ll demonstrate their usage by developing a simple freestyle application for creating and displaying purchase documents. Additionally, you’ll learn how to set up an SAPUI5 freestyle application project in the SAP Web IDE full-stack, how to establish the backend OData service connection, how to configure navigation between different views, and how to write a One-Page Acceptance (OPA5) user interface (UI) integration test to test the whole application scenario.

7.1 Smart Controls

Smart controls are part of the sap.ui.comp SAPUI5 library, which contains composite SAPUI5 controls. As its name suggests, controls in this library are a combination of several SAPUI5 basis controls, for instance, from the sap.m library, and are bundled
together in a new reusable control. Moreover, apart from combining several controls, controls starting with the smart prefix analyze OData metadata or UI annotations to reduce the necessary XML and JavaScript SAPUI5 frontend code. For instance, if a SmartField detects a value help OData annotation in the service metadata document for the property it's bound to, the SmartField will automatically render a value help.

In the following sections, we'll introduce the most common smart controls and their main annotations.

### 7.1 Smart Field

The `sap.ui.comp.smartfield.SmartField` SAPUI5 control is a wrapper around several other controls that are rendered based on the provided OData annotations of the respective bound OData entity type property. The purpose of the SmartField is to display or edit field values of an underlying OData entity. Listing 7.1 shows an example of how a SmartField can be used inside an SAPUI5 XML view definition. In this example, the SmartField is bound to the PurchasingOrganization property via its value property. The smartfield prefix is defined for the "sap.ui.comp.smartfield" XML namespace (`xmlns:smartField="sap.ui.comp.smartfield"`).

```
<smartField:SmartField value="{PurchasingOrganization}"/>
```

Listing 7.1 SmartField Bound to the PurchasingOrganization Property of the Z_C_PurchaseDocumentLrp Entity

At runtime, the SmartField is rendered as an input field with a corresponding label and value help. What controls are rendered exactly depends on the attributes of the bound entity type property of annotations in the OData service metadata document. In our example, the field is rendered as an editable input field because its property element doesn't contain the `sap:creatable="false"` and `sap:updatable="false"` XML attributes, and it has the Entity Data Model (EDM) type Edm.String, as shown in Figure 7.1.

Figure 7.1 OData Service Metadata Document Annotations Interpreted by the SmartField Control

Figure 7.2 shows the required OData annotations for rendering the generic standard value help. The value help is based on the value help definition to the PurchasingOrganization CDS view; it takes the PurchasingOrganization field as an in-and-out argument and additionally displays the Description of the purchasing organization. An in-and-out argument is when the PurchasingOrganization field is taken as an input for filtering the value help data by PurchasingOrganizations and is additionally also displayed in the output table of the value help.

```
<Purchasing Organization:>

Purchasing Organization:>

Figure 7.2 Value Help Annotations for the PurchasingOrganization Field

### 7.1.2 Smart Link

The `sap.ui.comp.navpopover.SmartLink` control renders a link for a field and provides a popup window containing navigation links to other related applications, as shown in Figure 7.3. To generate these links, this control analyzes the SemanticObject OData annotation of the bound OData entity type property and collects all navigation targets of the user containing the respective semantic object—in this case, PurchasingDocument. In our example, the user has access to the PurchasingDocument – create (Create Purchase Document), monitor (Purchase Documents Overview Page), and analyzeKPI-Details (Purchasing Documents) applications. The possible navigation targets are
restricted to the targets contained in the assigned frontend Transaction PFCG business roles of the user. In an SAP Gateway hub scenario, frontend Transaction PFCG roles containing the relevant business catalogs, target mappings, and groups are assigned to the user on the frontend or on the hub system. To generate the Semantic Object OData annotation, the field must be annotated with @Consumption.semanticObject:<SEMANTIC_OBJECT> on the CDS data definition language (DDL) level. Like the UI annotations, the annotation is part of the separate annotation model provided by the SAP Gateway Catalog Service when generating an OData service using Auto-Exposure or the Reference Data Source (RDS) scenario.

Figure 7.3 Smart Link and Its Navigation Popup Window Containing Links to Related Applications with the Annotated Semantic Object

Note
The sap.ui.comp.navpopover.SmartLink control will only work in the SAP Fiori launchpad and not when testing the application locally in the SAP Web IDE. The user will only see navigation links and get access to applications that are part of his assigned frontend Transaction PFCG business roles and whose semantic object defined in their target mappings match the semantic object defined for the field. The available target mappings for a user, containing the semantic objects and actions a user has authorization for, are fetched while SAP Fiori launchpad is starting up.

7.1.3 SmartForm
The sap.ui.comp.smartform.SmartForm control, in combination with SmartField controls, lets you create a form with minimal effort. This control contains a group (sap.ui.comp.smartform.Group) aggregation that contains a GroupElements (sap.ui.comp.smartform.GroupElement) aggregation. A GroupElement is a combination of a label and different controls associated with this label, usually a SmartField. Listing 7.2 shows the general structure of a SmartForm definition, including the Group, GroupElement, and SmartField controls.

Listing 7.2 Element Structure of a SmartForm Consisting of Group, GroupElement, and SmartField Controls

Moreover, the SmartForm contains a layout aggregation that you can use to configure the ResponsiveGridLayout, which is internally used by the SmartForm to align its controls in a responsive way, depending on the available space. The layout organizes its child controls in a 12-column row. Using the properties of the control, you can specify how many columns the controls should consume based on the available screen size. Listing 7.3 shows an example on how the sap.ui.comp.smartform.Layout control can be used with the layout aggregation to configure the layout of the SmartForm.

Listing 7.3 Layout Example for the SmartForm
The `emptySpanL="1"` attribute specifies that, on a large screen, one grid cell at the end of each line will be empty. On a medium size screen, you'll need all the space, so set it to `emptySpanM="0"`. On large and medium size screens, the label control should not take up more than 3 columns of the 12-column grid space (`labelSpanL="3"` and `labelSpanM="3"`).

Another feature of the SmartForm control is the possibility to enable a check button for the form by setting `checkButton="true"`. If you click this button, the values of the SmartFields contained in the form will be checked against their OData metadata annotations. Figure 7.4 shows an example of a validation triggered by the Check button. The `PurchaseDocument` property is marked as mandatory by the `Nullable="false"` attribute in the OData metadata annotations. Therefore, if the check determines that a SmartField is empty, that field will be marked in red. When the user clicks on the field, the corresponding error message is shown. But since we've already made the `PurchaseDocument` field a read-only field (because we're generating its value in the backend), we'll make this field hidden.

Figure 7.4 Example Validation of a Mandatory Field Triggered by Clicking the Check Button

### 7.1.4 Smart Table

The `sap.ui.comp.smarttable.SmartTable` control is a wrapper containing SAPUI5 tables. Like the other smart controls, it analyzes the OData service's metadata annotations (`$metadata`) and the UI annotations provided by a separate annotation model and the Catalog Service of the SAP Gateway hub (`/sap/opu/odata/IWFND/CATALOGSERVICE;v=2/Annotations(TechnicalName='<ANNOTATION_MODEL>',Version='0001')/$value`). UI annotations for Smart Controls and SAP Fiori elements applications are not exposed via the app's standard OData service but via a separate annotation model that is exposed via a separate OData service called `CATALOGSERVICE;v=2`. You will find this service in the set of available services on the SAP Gateway system. The SmartTable control renders a table for a specific OData entity set and can create tables with just a view lines of XML. Because the SmartTable control is also used in the SAP Fiori elements list report template, the same UI annotations can be used for both cases.

Figure 7.5 shows some UI.LineItem annotations and how they are interpreted by the SmartTable control. As in an SAP Fiori elements list application, the `@UI.lineItem` annotation must be used on the CDS DDL level to specify the default visible columns of the table and their sequence. The criticality of the `OverallPrice` and `ApprovalRequired` columns is indicated by the `OverallPriceCriticality` field. This information is also provided by the line item annotation.

Figure 7.5 Example of How the UI.LineItem Annotations Are Interpreted by the SmartTable Control

You can hide certain fields from the UI but not completely from the entity type. In this way, for instance, you can use these fields as a criticality indicator, you must annotate these fields with the `@UI.hidden` annotation. Listing 7.4 shows how to add the annotation to the `OverallPriceCriticality` field since we don't want to make this field available in the table field catalog but instead need it as a technical field to indicate the criticality.
Developing a Freestyle Application

Listing 7.4 Hide a Field from the UI but Not from the OData Entity Type

Defining a smart table in an SAPUI5 XML view is quite simple. Listing 7.5 shows the minimal property values required for rendering the table. The `smartTable` prefix is defined for the `sap.ui.comp.smarttable` namespace (`xmlns:smartTable="sap.ui.comp.smarttable"`). The `entitySet` defines the OData entity set that the table will be bound to. The `header` attribute sets the text of the table header, and the `tableType` attribute specifies the table type to be used—in our case, a `ResponsiveTable`. By setting the `enableAutoBinding` attribute to `true`, data for the specified entity set will be requested immediately from the OData service deployed on the SAP Gateway hub system when the table is rendered.

```
<smartTable:SmartTable entitySet="Z_C_PurchaseDocumentLrp" header="Purchase Documents" enableAutoBinding="true" tableType="ResponsiveTable" />
```

Listing 7.5 SmartTable Definition inside an SAPUI5 XML View

Usually, a `SmartTable` is used in combination with a `SmartFilterBar`. The two controls are connected using the `id` of the `SmartFilterBar`. Therefore, the `SmartTable` must define the `SmartFilterBar`'s `id` using its `smartFilterId` attribute.

7.1.5 Smart Filter Bar

The `sap.ui.comp.smartfilterbar.SmartFilterBar` control analyzes OData metadata and UI annotations and renders a `FilterBar` control that can be used to filter a `SmartTable`. Figure 7.6 shows some of the metadata and UI annotations interpreted by the `SmartFilterBar` control. The initial displayed fields in the expanded filter bar must be annotated with the OData `UI.SelectionField` annotation (`@UI.selectionField` on the CDS DDL level). In the example, the `PurchaseDocument`, `Priority`, `Status`, and `PurchasingOrganization` fields were annotated as selection fields. Which kind of value help is rendered for a selection field depends on the `sap:value-list` attributes of the relevant entity type property; for instance, a dropdown list-style value help will be rendered for the `Priority` field because its `sap:value-list` attribute has a `fixed-values` value. The `PurchasingOrganization` property's `sap:value-list` attribute is set to standard, so a standard value help will be rendered.

```
<smartTable:SmartTable entitySet="Z_C_PurchaseDocumentLrp" header="Purchase Documents" enableAutoBinding="true" tableType="ResponsiveTable" />
```

Listing 7.5 SmartTable Definition inside an SAPUI5 XML View

In an SAPUI5 XML view, the `SmartFilterBar` control can be declared as shown in Listing 7.6. The `smartFilterBar` prefix is defined for the `sap.ui.comp.smartfilterbar` namespace (`xmlns:smartFilterBar="sap.ui.comp.smartfilterbar"`). The `entitySet` attribute defines the OData entity set whose metadata is used to render the smart filter bar. The `persistencyKey` specifies that the key that is used for storing personalization data, for instance, a filter variant that must be used periodically. Although the entity set might support free text search, as indicated by the `sap:searchable="true"` attribute, a free text search field isn't rendered automatically. To be enabled in the smart filter bar, the `enableBasicSearch` attribute must be set to `true` explicitly.

```
<smartFilterBar:SmartFilterBar id="purchaseDocumentsSmartFilterBar" entitySet="PurchaseDocument" persistencyKey="purchaseDocumentsFilterBarKey" enableBasicSearch="true" />
```

Listing 7.6 SmartFilterBar Definition inside an SAPUI5 XML View

To completely remove a field from the smart filter bar's set of selection fields, you'll annotate the field with `@Consumption.filter.hidden: true` on the CDS DDL level, as shown in Listing 7.7. On the OData service level, this configuration will generate the OData UI annotation `UI.HiddenFilter` for the respective field.
7.2 Application Development with the SAP Web IDE Full-Stack

In this section, we’ll develop a small SAPUI5 freestyle application consisting of a SmartForm for creating purchase documents and a smart filter bar and table for displaying purchase documents. We’ll develop the frontend part of the application in the SAP Web IDE full-stack and reuse the OData service developed in Chapter 4 for the SAP Fiori elements list report and object page. We’ll guide you through all of the necessary steps for developing the application, including setting up the OData service connectivity and navigation in the app descriptor (manifest.json) as well as creating an OPAS integration test covering the whole create and display scenario.

7.2.1 Setting Up an OData Service

We’ll start developing our app by first creating the initial project structure and setting up the OData service connection to the backend. To create a new freestyle SAPUI5 project in the SAP Web IDE full-stack, choose File > New > Project from Template from the top-level menu bar. In the initial Template Selection step of the project creation wizard shown in Figure 7.7, select the SAPUI5 Application template, which is contained in the SAP Fiori Application Category. The SAPUI5 Application template is an empty SAPUI5 project skeleton that includes a start view and its corresponding controller. Click Next.

Note

The smart table, as well as the smart filter bar control, support variant management as soon as you provide a unique key via their persistencyKey property under which the settings can be stored. The variants will be stored in the layered repository of the SAP Gateway hub frontend system. To store smart table and filter bar settings in a single variant, you can use the smart variant management control to enable page variant management. You can find further information under Smart Variant Management in the documentation for SAPUI5 Software Development Kit (SDK) at http://s-prs.co/498820.

Figure 7.7 Creating a New SAPUI5 Application in the SAP Web IDE Full-Stack

On the next screen of the project creation wizard shown in Figure 7.8, provide the Project Name and the Namespace with which to prefix the SAPUI5 artifacts. Click Next.

Figure 7.8 Providing a Project Name and Namespace for the Freestyle SAPUI5 Project
In the template customization step shown in Figure 7.9, specify the **View Type** and a **View Name** for the initial view. XML view type is the recommended view type for defining SAPUI5 views. For our example, let’s call the initial view “CreatePurchaseDocument.” Click **Finish**.

**Figure 7.9** Selecting the View Type and Defining a View Name for the Initial View

To set up the connection to the backend OData service we developed in Chapter 4, open the app descriptor `manifest.json` file. The app descriptor is the central repository containing all application-relevant configurations, including the app’s data sources, models, and routing settings. To enable the OData service, including its UI annotation model, for usage within the application, add the service to the data sources of the application, as shown in Figure 7.10. In general, you can edit the app descriptor in two ways: using the form-based descriptor editor or the JavaScript Object Notation (JSON)-based code editor. Use the descriptor editor and click on the OData service’s **Add (+)** button.

A dialog wizard, shown in Figure 7.11, will open, where you’ll configure the **Data Connection**. First, enter the SAP Gateway hub system where the service was registered. Then, choose the service by searching for it using the **Services** search field. When you’ve found and selected the service, click **Next**.

In the **Model Selection** step shown in Figure 7.12, provide the model name for the previously defined OData service connection or set it as the default model of the application. In our example, we want to set the data connection as the default model because it’s the primary OData service of the application and therefore its default data source. Click **Next** and then **Finish**.

**Figure 7.10** Descriptor Editor to Edit the App Descriptor `manifest.json` File’s Data Sources

**Figure 7.11** Selecting the OData Service to Add as a Data Source to the App Descriptor

**Figure 7.12** Model Selection to Choose the Default Model for the OData Service
Now that the OData service has been selected and assigned to the default model, you’ll configure the annotation source to enable the use of smart controls, as shown in Figure 7.13.

![Figure 7.13 Added OData Service and Corresponding Annotation File](image)

Click the Add New Annotation (+) button in the Annotations section of the screen, below the OData Services section. Let’s call the annotation data source “Z_UI_PURCHASEDOCUMENT_V2_VAN” like the actual OData annotation model in the backend and set its URI to “/sap/opu/odata/IWFND/CATALOGSERVICE/v=2/Annotations(TechnicalName='Z_UI_PURCHASEDOCUMENT_V2_VAN',Version='0001')/$value/” because the annotations must be downloaded via the Catalog Service of the SAP Gateway hub system.

To write data back to the OData model and service, you’ll need to set the binding mode of the model to two-way. Select the Models tab of the descriptor editor and then select TwoWay from the Default Binding Mode dropdown list, as shown in Figure 7.14.

![Figure 7.14 Setting the Default Model’s Binding Mode to TwoWay](image)

Figure 7.15 and Figure 7.16 summarize the changes made so far. Figure 7.15 shows the data sources of the application in the JSON code editor of the `manifest.json` app descriptor file. We’ve added the Z_UI_PURCHASEDOCUMENT_V2 OData and Z_UI_PURCHASEDOCUMENT_V2_VAN annotations services as data sources to the app. The OData service references its UI annotations via the annotations property that points to the annotation service.
7 Developing a Freestyle Application

358

Figure 7.15 OData and OData Annotation Data Sources for the App in the App Descriptor Code Editor (manifest.json)

Figure 7.16 shows the models of the applications. The default model of the app references the Z_UI_PURCHASEDOCUMENT_V2 OData service and configures the OData service using additional properties, for instance, the binding and count mode. The model that will be instantiated by the UI component file (Component.js) on app startup is an sap.ui.model.odata.v2.ODataModel model. The i18n model provides language-dependent texts stored locally in i18n-* properties files, for instance, i18n-en.properties for English. Texts stored as properties in i18n-* properties files can be accessed in XML views via the i18n model using relative bindings (\{i18n>PROPERTY\}). For simplicity, we’ll hard-code the UI texts and not fetch them from language dependent i18n-* properties files.

```
{}
```

Figure 7.16 Default OData and i18n Models of the App in the App Descriptor Code Editor (manifest.json)

7.2.2 Object Creation Page Using SmartFields and Forms

Our app is now connected to an OData model, and you can start with the actual application development. Recall that, when we created the project in the previous section, the SAP Web IDE already created a view for us, which we called CreatePurchaseDocument.view.xml. As the next step, we’ll need to insert a SmartForm for creating purchase documents into the currently empty view definition that contains several SmartFields for entering the initial data of a purchase document. Additionally, we’ll add a save button to the footer of the page to trigger the submission of the purchase document to the backend OData service. The final SmartForm definition is shown in Figure 7.17. You’ll need to set the editable attribute of the SmartForm to true to render sap.m.Input fields instead of sap.m.Text fields for the different SmartFields. You’ll also set the checkButton attribute to true to enable the evaluation of the input data against the OData metadata of the relevant bound OData entity type property.

```
<smartform>
  <content>
    <sap.m.Input type="text" value="" />
    <sap.m.Input type="number" value="" />
    <sap.m.Input type="text" value="" />
    <sap.m.Input type="text" value="" />
    <sap.m.Input type="text" value="" />
    <sap.m.Input type="text" value="" />
  </content>

  <checkbutton attribute="true" />
</smartform>
```

Figure 7.17 SmartForm Definition for Creating a Purchase Document

Inside the SmartForm, you can only define one Group without a Label, but you can add several GroupElement controls, including SmartFields to the Group’s default groupElements.
aggregation. More specifically, you’ll add 
SmartFields fields for the PurchaseDocument ID, 
Description, Priority, PurchasingOrganization, and PurchaseDocumentImageURL properties of the Z_C_PurchaseDocumentLRP OData entity and bind their values to the respective value attribute of the SmartField control.

However, to render the UI based on OData metadata and UI annotations, the SmartForm and its child controls require bindings to OData entity properties to obtain their metadata information. The binding to a newly created purchase document entity can be established via the respective view controller. When the initial CreatePurchaseDocument.view.xml file was created by the SAP Web IDE, a corresponding CreatePurchaseDocument.controller.js file was also generated to serve as the controller of this view. As we’ve added the OData service as the default model to the app descriptor, the sap.ui.model.odata.v2.ODataModel will be instantiated automatically by the SAPUI5 framework. Automatically instantiated models can be retrieved in controllers by calling this.getView().getModel(). Having to retrieve a model from a controller happens quite frequently, so moving this functionality to a BaseController.js module from which other view controllers inherit, as shown in Listing 7.8, makes sense. You can then get the OData model inside view controllers by simply calling this.getModel().

Creating a new empty purchase document in the controller and binding it to the view is shown in Listing 7.9. In the onInit method of the CreatePurchaseDocument controller, you’ll attach to the route-matched event, which is fired every time the CreatePurchaseDocument route is displayed (more specifically, whenever the URL hash matches the pattern defined for the route). When the event is fired, the _createPurchaseDocument callback function is called. Inside the function, you’ll wait until the metadataLoaded promise is resolved, and if the model has no pending changes, you’ll create a new purchase document entity by calling the createEntry method of the OData model with the entity set name PurchaseDocument as a parameter. The new purchase document isn’t immediately sent to the backend, but instead a request is stored in a request queue so that we can later manually send it to the backend when we have finished editing the document.

Creating a new empty purchase document in the controller and binding it to the view is shown in Listing 7.9. In the onInit method of the CreatePurchaseDocument controller, you’ll attach to the route-matched event, which is fired every time the CreatePurchaseDocument route is displayed (more specifically, whenever the URL hash matches the pattern defined for the route). When the event is fired, the _createPurchaseDocument callback function is called. Inside the function, you’ll wait until the metadataLoaded promise is resolved, and if the model has no pending changes, you’ll create a new purchase document entity by calling the createEntry method of the OData model with the entity set name PurchaseDocument as a parameter. The new purchase document isn’t immediately sent to the backend, but instead a request is stored in a request queue so that we can later manually send it to the backend when we have finished editing the document.
To run the app from the Files pane in the SAP Web IDE full-stack, simply right-click on the root folder of the app and choose Run > Run As > Web Application from its context menu. At runtime, the Create Purchase Document view will resemble the screen shown in Figure 7.18.

![Create Purchase Document View at Runtime](image)

Finally, to send the newly created purchase document entity to the backend when the Save button is clicked and its onSave event handler function is triggered inside the controller, you must call the submitChanges method of the sap.ui.model.odata.v2.ODataModel instance as shown in Listing 7.10. Its parameter object contains a success callback function that is called on success and an error callback function that is called on error. If submitting the changes to the backend was successful, you'll also unbind the entity from the view so that you can create and bind a new entity.

```javascript
sap.ui.define([
    "...BaseController"
], function (BaseController) {
    "use strict",

    return BaseController.extend("...CreatePurchaseDocument",
    {
        onSave: function () {
            this.getModel().submitChanges({
                success: function () {
                    this.getView().unbindElement();
                    }.bind(this),
                error: function () { }.bind(this)
            });
        }
    });

Listing 7.10 Submit the Collected Changes to the Backend in the onSave Method of the CreatePurchaseDocument.controller.js Module

Before submitting the new purchase document to the backend, you should first evaluate its values against the metadata of the service to detect any obvious client errors by calling the SmartForm's check method. As shown in Listing 7.11, we must first fetch the SmartForm instance by its ID and then call its check method. The check method then returns an array of fields having errors in relation to the fields' OData metadata, for instance, the field is too long (MaxLength) or is an empty mandatory field (Nulllable=“false”).

```javascript
onSave: function () {
    var oSmartForm = this.getView().byId("createPurchaseDocumentSmartForm");
    var aSmartFieldsWithErrors = oSmartForm.check();
    if (aSmartFieldsWithErrors.length !== 0) {
        return;
    }
}
```

Listing 7.11 Call the SmartForm's Check Method to Check for Client Errors before Submitting Changes to the Backend

After submitting a newly created purchase document to the backend, you'll need to navigate to a new view that will display all purchase documents created so far. To
display and filter the purchase documents, you’ll use a smart table and smart filter bar control. For the moment, let’s create a new empty view called `DisplayPurchaseDocuments.view.xml`. You’ll first need to set up the navigation between the `CreatePurchaseDocument.view.xml` and the `DisplayPurchaseDocuments.view.xml` view before implementing the view definition. Additionally, you’ll create another view called `App.view.xml` that will serve as the root view of the application, as shown in Listing 7.12. Inside the view, you’ll only put a single `sap.m.App` control with the ID `app` as the root container for all the views.

Other views will then be dynamically inserted into the pages aggregation of the instantiated `sap.m.App` control at runtime. The `sap.m.App` control inherits its navigation capabilities from the `sap.m.NavContainer` and additionally adds some header tags to the HTML page that are considered useful for mobile applications.

```xml
<mvc:View xmlns="sap.m" xmlns:mvc="sap.ui.core.mvc" displayBlock="true" >
  <App id="app" busy="{app>/isBusy}" busyIndicatorDelay="0" />
</mvc:View>
```

**Listing 7.12 Root View of the Application**

The navigation routes and targets of the application are defined in the `sap.ui5` namespace of the app descriptor `manifest.json` file. However, the first thing you must do now is to make the newly created `App.view.xml` the root view of the application. Figure 7.19 shows how the `App.view.xml` can be defined as the root view of the application inside the `manifest.json` app descriptor file.

```json
"sap.ui5": { "rootView": { "viewName": "fiori.create.purchase.documentsPurchaseDocumentsFreestyle.view.App", "type": "XML", "async": true, "id": "app" } }
```

**Figure 7.19 Defining the Root View of the Application in the App Descriptor manifest.json File**

The navigation configuration for our application is shown in Figure 7.20. In the config object of the routing configuration, define some default configuration relevant for routes and targets, for instance, the `viewType` or `transition`. The `controllerId` and `controllerAggregation` properties are relevant for `sap.m.routing.Targets` and define that new views accessed using navigation will be inserted into the `pages` aggregation of the `sap.m.App` container control inside the root view (`App.view.xml`) with the ID `app`. The `CreatePurchaseDocument` route is mapped to the `CreatePurchaseDocument` view via the `CreatePurchaseDocument.target`. The pattern of the route is empty because it’s the start page of the application. The `DisplayPurchaseDocuments` route is mapped to the `DisplayPurchaseDocuments` view via the `DisplayPurchaseDocuments.target`.

```json
"routing": { "config": { "routerClass": "sap.m.routing.Router", "viewType": "XML", "controller": "app", "controllerAggregation": "pages", "transition": "slide", "async": true } } "route": { "name": "CreatePurchaseDocument", "pattern": "/CreatePurchaseDocument\{\}" target": "CreatePurchaseDocument" }, { "name": "DisplayPurchaseDocuments", "pattern": "/DisplayPurchaseDocuments\{\}" target": "DisplayPurchaseDocuments" }, { "targets": { "CreatePurchaseDocument": "v1View", "DisplayPurchaseDocuments": "v2View" } }
```

**Figure 7.20 Mapping from Routes to Target Views in the App Descriptor manifest.json File**

You can directly navigate to the `DisplayPurchaseDocuments` view by appending the fragment identifier `#/DisplayPurchaseDocuments` to the application URL as its route pattern is `DisplayPurchaseDocuments`.

To programmatically navigate in controllers from one view to another, you first must retrieve an instance of the `sap.m.routing.Router` defined in the app descriptor `manifest.json` file, which can be achieved inside a controller, as shown in Listing 7.13. We’ll define a new convenience function in the base controller from which all other controllers inherit since most controllers will need this functionality.
sap.ui.define([
    "sap/ui/core/mvc/Controller"
], function (Controller) {
    "use strict";
    return Controller.extend("./BaseController", {
        getRouter: function () {
            return sap.ui.core.UIComponent.getRouterFor(this);
        }
    });
});

Listing 7.13 Function Definition for Obtaining an Automatically Instantiated sap.m.routing.Router Instance via the sap.ui.core.UIComponent

To navigate from the create view to the display view after the new purchase document has been submitted to the backend, you’ll simply execute the code shown in Listing 7.14. The route name of the view you want to navigate to (DisplayPurchaseDocuments) is simply passed as a parameter to the navTo function of the sap.m.routing.Router object.

this.getRouter().navTo("DisplayPurchaseDocuments");

Listing 7.14 Navigate from the Create View to the Display View

This navigation is triggered inside the success callback of the submitChanges call shown earlier in Listing 7.10.

7.2.3 List Report Page Using Smart Table and Filter Bar

So far, we haven’t defined the content of the DisplayPurchaseDocuments.view.xml, which is our next step. As mentioned earlier, our goal is to display all available purchase documents in a table and enable filtering. Thus, we’ll use a smart table to display the purchase documents and a smart filter bar to filter the table data. Figure 7.21 shows the view definition containing a smart filter bar and smart table. Both controls are bound to the PurchaseDocument entity set, and the smart table and filter bar are connected by the smart filter bar ID purchaseDocumentsSmartFilterBar.

If you run the app now and create a purchase document, you should automatically be forwarded to the Display Purchase Documents view, which will display all available purchase documents in a smart table, as shown in Figure 7.22. To create purchase document items, you could simply navigate to the created object page (discussed throughout Chapter 4) by clicking on the Purchase Documents Manage link. From the object page, you could add items to the purchase document.

Figure 7.21 Definition of the DisplayPurchaseDocuments.view.xml View Containing a Smart Table and Filter Bar Control

Figure 7.22 Display Purchase Documents View at Runtime

7.2.4 Add a One-Page Acceptance Integration Test

To test and safeguard the creation and display of purchase documents with the application, including the navigation between the create and display views, you should
write a UI integration test, with the help of the One-Page Acceptance (OPAS) test for the SAPUI5 framework, which is part of the SAPUI5 SDK. This framework provides an API for SAPUI5 controls and eases testing user interactions and navigations by hiding asynchronicity. OPAS can be used with any JavaScript unit test framework, but, in our case, we’ll use QUnit as the basic test runner and unit test framework. You can easily set up a default test structure for an SAPUI5 application project in the SAP Web IDE full-stack by right-clicking on the project root folder and choosing New • Test Structure, as shown in Figure 7.23.

Figure 7.23 Creating a Test Structure for an SAPUI5 Application Project in the SAP Web IDE Full-Stack

A test folder with unit and integration test subfolders will be generated. The unit subfolder will contain QUnit test skeletons, and the integration subfolder will contain OPAS integration test skeletons. The generated structure is shown in Figure 7.24. We’ll ignore unit testing for now and focus on UI integration testing in the following section. The SAP Web IDE has also generated a default OPAS journey (navigationJourney.js), which we’ll also ignore and delete because we’ll create our own journey.

Nevertheless, getting this test structure generated is quite convenient; for instance, we now have a test runner (opaTests.qunit.html) for running the OPAS journeys we’ll develop and a module collecting all our journeys (AllJourneys.js). We’ll later adapt the files contained in the pages folder to our specific needs. OPAS tests are organized in journeys and pages. A journey contains all test cases for a specific use case and structures its test cases according to high-level given-when-then sentences hiding the actual test case implementation. Journeys use pages that implement the necessary arrangements, actions, and assertions used within the high-level journey definition and are usually related to a single view.

Figure 7.24 Generated Test Structure for the SAPUI5 Application Project

Setting Up a Mock Server

When testing the purchase document creation, navigation, and display process in the UI, you should use a mock server because you don’t want to depend on any backend OData service or data being available. The mock server will intercept the OData requests the application sends to the backend and return local mock data instead. The mock server supports create, read, update, and delete (CRUD) requests and stores changes locally during a session. To set up a mock server, create a mockserver.js module in the localService folder of the project where you’ll set up and configure the server. Listing 7.15 shows how you can instantiate the mock server with the root URI of the OData service and simulate the service by specifying the local service metadata document (metadata.xml) and the base folder for mock data (mockdata). By setting the bGenerateMissingMockData property to true, the mock server will automatically
generate mock data for entities that don’t have dedicated mock data in the mockdata
folder.

```javascript
sap.ui.define([
    "sap/ui/core/util/MockServer"
], function (MockServer) {
    "use strict";
    return {
       init: function () {
          var oMockServer = new MockServer({
             rootUri: "/sap/opu/odata/sap/ Z_UI_PURCHASEDOCUMENT_V2/"
          });

          oMockServer.simulate("../localService/metadata.xml", {
             sMockdataBaseUrl: "/localService/metadata.xml",
             bGenerateMissingMockData: true
          });

          oMockServer.start();
        }
    }
});
```

Listing 7.15 Instantiating the sap.ui.core.util.MockServer with the OData Service’s Root URI and Simulating the OData Service by Specifying the Metadata XML File and the Base Folder of the Mock Data

Create Mock Data
You can easily create proper mock data for entities by right-clicking on the local metadata.xml file in the localService folder and choosing Edit Mock Data. The Edit Mock Data dialog box, shown in Figure 7.25, will open where you’ll select entities in the Entity Sets pane and create mock data by clicking on the Add Row button in the upper-right corner.

After you click OK, mock data JSON files will be created in the mockdata folder containing the previously defined data, as shown in Figure 7.26.

Because you’re using smart controls, the application also heavily depends on the UI annotation model for rendering the controls. To completely isolate the application from the backend, you also must mock the Catalog Service, which is providing the UI annotation model of the OData service, as shown in Listing 7.16.
mock server, you’ll define an additional request, matching all GET requests sent to the service that will simply return the local annotation file Z_C_PURCHASEDOCUMENTLRP_CDS_VAN.xml.

```javascript
... var oAnnotationMockServer = new MockServer({
  rootUri: "/sap/opu/odata/IWFND/CATALOGSERVICE;v=2/Annotations(TechnicalName='Z_C_PURCHASEDOCUMENTLRP_CDS_VAN',Version='0001')/$value/
});

oAnnotationMockServer.setRequest({
  method: "GET",
  path: new RegExp(".*"),
  response: function (oXhr) {
    var oLocalAnnotations = jQuery.sap.sjax({
      url: sAnnotationURL,
      datatype: "XML"
    }).data;
    oXhr.respondXML(200, {}, jQuery.sap.serializeXML(oLocalAnnotations));
    return true;
  }
});

oAnnotationMockServer.start();
```

Note
To update the local metadata file in the SAP Web IDE full-stack, simply right-click on the file and choose Sync Metadata.

Creating an OPAS UI Integration Test
The journey we want to implement and test is shown in Listing 7.17 and should almost read like a manual test case description or user story. This approach is called behavior-driven development (BDD). The general idea behind BDD is that requirements are defined in the form of high-level given-when-then sentences and later executed as automated tests, which eases the definition of requirements and test scenarios by nontechnical stakeholders. With our journey, we’ll test the creation, saving, and correct display in the smart table of the purchase document after the automatic navigation to the display view took place.

```javascript
sap.ui.define([
  "sap/ui/test/opaQunit"
], function (opatTest) {
  "use strict";

  QUnit.module("Create Purchase Document Journey");

  opatTest("Should see the newly created Purchase Document in the Smart Table", function (Given, When, Then) {
    Given.iStartTheApp();
    var oPurchaseDocument = {
      Description: "OPAS Test Description",
      Priority: "1",
      PurchasingOrganization: "ORG1",
      PurchaseDocumentImageURL: "../images/book.jpg"
    };

    When.onTheCreatePurchaseDocumentPage.iEnterPurchaseDocumentData(oPurchaseDocument)
      .and.iSavePurchaseDocument();

    Then.onTheDisplayPurchaseDocumentsPage.iCheckPurchaseDocumentIsInTable(oPurchaseDocument)
      .and.iTeardownMyUIComponent();
  });

  function (Given, When, Then) {
    Given.iStartTheApp();
    var oPurchaseDocument = {
      Description: "OPAS Test Description",
      Priority: "1",
      PurchasingOrganization: "ORG1",
      PurchaseDocumentImageURL: "../images/book.jpg"
    };

    When.onTheCreatePurchaseDocumentPage.iEnterPurchaseDocumentData(oPurchaseDocument)
      .and.iSavePurchaseDocument();

    Then.onTheDisplayPurchaseDocumentsPage.iCheckPurchaseDocumentIsInTable(oPurchaseDocument)
      .and.iTeardownMyUIComponent();
  });
```

Listing 7.17 An OPAS Journey for Creating and Displaying a Purchase Document (createPurchaseDocumentJourney.js)
The `Given` object provided as a parameter to the `opaTest` function contains the arrangements of all loaded pages, the `When` object in turn contains all actions, and the `Then` object contains all assertions. In our journey, we’re calling one arrangement method (`iStartTheApp`), which will start the app’s UI component (`Component.js`) and the mock servers defined in the `mockserver.js` module by calling its `init` method. The arrangement is implemented in the `Common.js` module, which serves as the base class for all the page objects we’ll implement, as shown in Figure 7.27.

### Figure 7.27 Common.js Base Object for OPA5 Pages Containing the iStartApp Arrangement

The `iEnterPurchaseDocumentData` and `iSavePurchaseDocument` actions are implemented in the `CreatePurchaseDocument.js` page. Figure 7.28 shows some of the implementation of the `iEnterPurchaseDocumentData` action. We’re using the `waitFor` method of `sap.ui.test.Opa5` to wait until the control we want to select using its view and SmartField ID has been rendered. As soon as the control is selected, we’re calling the pre-defined `sap.ui.test.actions.EnterText` action to enter text into the input fields, which will enable us to fill out the purchase document creation form.

`function` shown in Figure 7.28 also implemented in the `CreatePurchaseDocument.js` OPA5 Page Object

After you’ve filled out the form, the `iSavePurchaseDocument` action will be called from the `createPurchaseDocumentJourney.js` journey. In this action, shown in Listing 7.18, simply click the `Save` button using the predefined `sap.ui.test.actions.Press` action as soon as the button has been found by its ID inside the `CreatePurchaseDocument` view.

### Listing 7.18 iSavePurchaseDocument Action also Implemented in the CreatePurchaseDocument.js OPAP Page Object

```javascript
CreatePurchaseDocument.js

createPurchaseDocumentJourney.js

Listing 7.18 iSavePurchaseDocument Action also Implemented in the CreatePurchaseDocument.js OPAP Page Object

7.2 Application Development with the SAP Web IDE Full-Stack
For the iCheckPurchaseDocumentIsInTable assertion, checking the presence of the newly created purchase document in the smart table, we’ll create a new page called DisplayPurchaseDocuments.js as the assertion logically belongs to the DisplayPurchaseDocuments.view.xml. In the assertion shown in Figure 7.29, the smart table is first selected by its view and ID. On top of that, two custom matchers are implemented to find the newly created purchase document inside the smart table.

Figure 7.29 Assertion (iCheckPurchaseDocumentIsInTable) Checking the Display of the Newly Created Purchase Document in the Smart Table on the DisplayPurchaseDocuments View

```javascript
DisplayPurchaseDocuments.js

1 - sap.ui.define()
2  "use strict";
3  "lib\create\purchase\documents\PurchaseDocumentsFreestyle\test\integration\pages\Common"
4 + function (Opas, Common) {
5    "use strict";
6    "lib\create\purchase\documents\PurchaseDocumentsFreestyle\test\integration\pages\Common"
7    function purchaseDocumentsAreEqual(expectedPurchDoc, actualPurchDoc) {
8      return expectedPurchDoc.Description === actualPurchDoc.Description &&
9          expectedPurchDoc.Priority === actualPurchDoc.Priority &&
12        }
13
14 + Opas.createPageObject();
15      onTheDisplayPurchaseDocumentsPage: () => {
16          baseClass: Common,
17          actions: () => {
18            iCheckPurchaseDocumentsIsInTable: function (PurchaseDocument) {
19              return this.waitFor(
20                  viewName: 'DisplayPurchaseDocuments',
21                  id: 'purchasedocumentsSmartTable',
22                  matchers: [function (oSmartTable) {
23                      return oSmartTable.getItems().some(function (oItem) {
24                          return oItem.getItems().some(function (oTableEntry) {
25                              return purchaseDocumentsAreEqual(PurchaseDocument, oPurchaseDocumentTableEntry);
26                          });
27                      });
28                  }]
29              })
30          })
31        }
32 + success: function () {
33            Opas.assert.ok(true, "New Purchase Document is Displayed in the Smart Table");
34        }
35    },
36    errorMessage: 'Could not find the new Purchase Document in the Smart Table'
37  });
38
39 })
```

Figure 7.30 AllJourneys.js Module: Central Module for Loading All OPAS Journeys and Pages Needed for Testing the App
Similar to the QUnit unit tests, the `opaTests.qunit.html` file is a QUnit runner and executes all integration tests defined in the `AllJourneys.js` module file, as shown in Figure 7.31. Additionally, this file displays its result in a readable format as a web page.

![Figure 7.31 Running the OPA5 Test Using the opaTests.qunit.html File as a QUnit Test Runner](image)

### 7.3 Summary

In this chapter, you learned about the most common smart controls: `SmartField`, `smart link`, `SmartForm`, `smart table`, and `smart filter bar`. We discussed the annotations they analyze to adapt their appearance and how they significantly reduce the amount of required frontend SAPUI5 XML and JavaScript code by still providing flexibility in terms of the overall layout and design of the application. We demonstrated all the smart controls described in this chapter by developing a small freestyle SAPUI5 application for creating and displaying purchase documents. Additionally, you learned how to set up a freestyle SAPUI5 project in SAP Web IDE full-stack, configure data sources and models, set up navigation between views, and create a UI integration test, including a mock server for the whole create and display scenario using the OPA5 framework included with the SAPUI5 SDK. In the next chapter you’ll learn how to deploy all applications developed so far to the SAP Fiori launchpad.
Contents

Preface ........................................................................................................................ 15

PART I SAP S/4HANA and SAP Fiori

1 SAP S/4HANA Architecture ........................................................................ 23
   1.1 Core Architecture ......................................................................................... 23
       1.1.1 Overview ................................................................................................. 24
       1.1.2 Open Data Protocol (OData) ................................................................. 26
       1.1.3 SAP HANA .............................................................................................. 26
       1.1.4 SAP NetWeaver Application Server for ABAP ............................. 27
       1.1.5 SAP Fiori Frontend Server Based on SAP NetWeaver Gateway ....... 29
       1.1.6 SAPUI5 ..................................................................................................... 30
       1.1.7 SAP Fiori Launchpad ............................................................................. 30
   1.2 Backend .......................................................................................................... 31
       1.2.1 Virtual Data Model .................................................................................. 31
       1.2.2 ABAP Programming Model for SAP Fiori ........................................ 39
       1.2.3 ABAP RESTful Programming Model .................................................. 42
       1.2.4 Analytics ................................................................................................ 45
   1.3 User Experience ............................................................................................... 47
       1.3.1 SAP Fiori .................................................................................................. 47
       1.3.2 SAP Fiori Launchpad ............................................................................. 47
   1.4 SAP S/4HANA Editions .................................................................................. 48
       1.4.1 On-Premise Architecture ....................................................................... 48
       1.4.2 Cloud Architecture ................................................................................ 48
       1.4.3 Release Strategies .................................................................................. 50
   1.5 SAP Cloud Platform ....................................................................................... 51
   1.6 Summary .......................................................................................................... 53
2 SAP Fiori and the Design-Led Development Process

2.1 What is SAP Fiori? .................................................. 55
   2.1.1 SAP Fiori 1.0 ............................................. 56
   2.1.2 SAP Fiori 2.0 ............................................. 57
   2.1.3 SAP Fiori 3.0 ............................................. 58
   2.1.4 Design Principles ........................................ 64
   2.1.5 Responsiveness and Adaptiveness ..................... 66
   2.1.6 SAP Fiori Launchpad .................................. 67

2.2 Design-Led Development Process ......................... 74
   2.2.1 Discover ................................................. 75
   2.2.2 Design .................................................... 76
   2.2.3 Develop .................................................. 76

2.3 Different SAP Fiori App Types ............................... 76
   2.3.1 Freestyle ................................................. 76
   2.3.2 SAP Fiori Elements .................................... 77

2.4 Prototyping Tools .................................................. 90
   2.4.1 Axure .................................................... 90
   2.4.2 SAP Build ............................................... 90
   2.4.3 Building a Prototype Using the SAP Build Tool .......... 92

2.5 Summary ......................................................... 105

3 ABAP RESTful Programming Model ......................... 107

3.1 Core Data Services ............................................. 108

3.2 SAP Gateway .................................................. 110

3.3 OData ........................................................... 111

3.4 ABAP RESTful Programming Model ................. 115
   3.4.1 Introduction .......................................... 115
   3.4.2 Query .................................................... 120
   3.4.3 Business Objects ..................................... 123

3.4.4 Business Service ........................................ 132
3.4.5 Service Consumption ................................... 134

3.5 Summary ......................................................... 135

PART II Developing Applications for SAP S/4HANA

4 Developing an SAP Fiori Elements List Report and Object Page .................................................. 139

4.1 Core Data Services Views ..................................... 141
   4.1.1 ABAP Development Tools in Eclipse .................... 142
   4.1.2 Introduction to the Data Model ........................ 145
   4.1.3 Creating Basic Interface Core Data Services Views .... 150
   4.1.4 Adding Calculated Fields ............................... 156
   4.1.5 Adding Data Control Language Files ................. 167

4.2 Developing an Unmanaged Transactional Application Using the ABAP RESTful Programming Model .................. 170
   4.2.1 Generating an Unmanaged Transaction Business Object from a CDS Model ........................................ 171
   4.2.2 Service Definition ......................................... 173
   4.2.3 Service Binding .......................................... 175
   4.2.4 Entity Manipulation Language (EML) ................. 179
   4.2.5 Behavior Definitions to Add Transactional Features ....................... 181
   4.2.6 Behavior Implementation Language (BIL) ............... 186
   4.2.7 Behavior Handler Local Class ........................ 187
   4.2.8 Behavior Saver Local Class ........................... 198
   4.2.9 Behavior Implementation ................................. 199
   4.2.10 Creating Projection Views ............................. 231
   4.2.11 Creating Projection Behavior Definition ............... 236

4.3 Virtual Elements in Core Data Services ............... 236
   4.3.1 Adding a Virtual Element to a Core Data Services View ....................... 237
   4.3.2 Implementing an ABAP Code Exit to Populate the Virtual Element .... 237

4.4 Adding User Interface Annotations to Projection Views .................................................. 238
   4.4.1 Creating a Metadata Extension File .................... 238
5 Developing an Overview Page

5.1 Core Data Services Views
   5.1.1 Creating a Simple Core Data Services View .............................................. 269
   5.1.2 Adding a Data Control File ........................................................................ 279
5.2 Adding User Interface Annotations
   5.2.1 Creating Annotations for an Analytical Card ............................................. 280
   5.2.2 Creating Annotations for a List Card ......................................................... 282
5.3 Creating an OData Service Using the ABAP RESTful Programming Model
   5.3.1 Creating a New Service Definition ............................................................... 286
   5.3.2 Creating a New Service Binding ................................................................. 288
5.4 Generating an Overview Page Template Project in SAP Web IDE
   5.4.1 Generating the Basic Overview Page Layout .............................................. 291
   5.4.2 Adding the List Analytical Card ................................................................. 297
   5.4.3 Adding the Standard List Card ................................................................... 300
   5.4.4 Adding the Bar List Card ............................................................................. 303
   5.4.5 Adding the Table Card ................................................................................ 304
   5.4.6 Overview Page Output ................................................................................ 306
5.5 Summary ......................................................................................................... 307

6 Developing an Analytical List Page

6.1 Introduction .................................................................................................... 309
6.2 Building the Required CDS Views
   6.2.1 Building Dimension Views .......................................................................... 309
   6.2.2 Building Cube Views .................................................................................. 311
   6.2.3 Building the Main Query View ................................................................... 314
6.3 Configuring the Title Area ............................................................................. 315
6.4 Configuring the Filter Area ............................................................................ 318
6.5 Configuring the Content Area
   6.5.1 Configuring the Default Chart ..................................................................... 321
   6.5.2 Configuring the Table .................................................................................. 323
6.6 Combining All the UI Annotations in the Metadata Extension View ............. 325
6.7 Generating an Analytical List Page from SAP Web IDE
   6.7.1 Adding Key Performance Indicators to the Project ..................................... 329
   6.7.2 Adding Visual Filters to the Project ............................................................. 333
6.8 Summary ......................................................................................................... 341

7 Developing a Freestyle Application

7.1 Smart Controls
   7.1.1 SmartField ................................................................................................. 343
   7.1.2 SmartLink ................................................................................................... 345
   7.1.3 SmartForm .................................................................................................. 347
   7.1.4 SmartTable ................................................................................................ 348
   7.1.5 SmartFilterBar .......................................................................................... 350
7.2 Application Development with the SAP Web IDE Full-Stack
   7.2.1 Setting Up an OData Service ................................................................. 352
   7.2.2 Object Creation Page Using SmartFields and Forms ............................... 352
   7.2.3 List Report Page Using Smart Table and Filter Bar ............................... 359
   7.2.4 Add a One-Page Acceptance Integration Test .......................................... 366
7.3 Summary ......................................................................................................... 378
PART III  Operating Applications

9  Version Control in SAP Web IDE Using Git 419

9.1 Git Introduction ................................................................. 419
9.2 Git Basics ............................................................................ 419
  9.2.1 Creating Initial Project Repositories Using GitHub .................. 421
  9.2.2 Initializing the Local Repository for the Projects in SAP Web IDE .... 423
  9.2.3 Linking the Local Repository with the Remote Repository in GitHub ......................................................... 424
  9.2.4 Submitting Code to Repository (Stage, Commit, Push) ............... 425
  9.2.5 Cloning the Project into SAP Web IDE .................................. 431
  9.2.6 Getting Code from the Remote Branch (Fetch, Merge, Pull) ........ 434
  9.2.7 Working with Branches .................................................. 439
9.3 Summary ............................................................................. 444

10  Automated Testing 445

10.1 Backend Test Automation .................................................. 448
  10.1.1 Unit Testing (ABAP Unit) ..................................................... 448
  10.1.2 Unit Testing (Core Data Services Test Double Framework) .... 461
10.2 Frontend Test Automation .................................................. 465
  10.2.1 Unit Testing (QUnit) ............................................................ 465
  10.2.2 Integration Testing (OPAS) .................................................. 468
10.3 End-to-End Test Automation Tools ...................................... 471
  10.3.1 Setting Up Nightwatch.js .................................................... 472
  10.3.2 Creating the Create Purchase Document End-to-End Nightwatch.js Test ......................................................... 473
  10.3.3 Running the Create Purchase Document End-to-End Nightwatch.js Test ......................................................... 474
10.4 Summary ............................................................................. 475

11  Continuous Integration 477

11.1 Introduction ......................................................................... 478
  11.1.1 Continuous Integration ....................................................... 478
  11.1.2 Continuous Delivery .......................................................... 479
  11.1.3 Continuous Deployment ..................................................... 480
11.2 Setting Up a Continuous Integration Pipeline for SAPUI5 on the ABAP Server ................................................................. 481
  11.2.1 Setting Up a Local Jenkins Automation Server ....................... 482
  11.2.2 Creating an Initial Jenkinsfile .............................................. 483
  11.2.3 Creating the Continuous Deployment Pipeline ....................... 484
  11.2.4 SAPUI5 Grunt Plug-ins ....................................................... 487
  11.2.5 Implementing the Build Stage ............................................. 488
  11.2.6 Automatically Triggering Builds on Git Push ......................... 491
  11.2.7 Implementing the Test Stage ............................................... 492
  11.2.8 Implementing the Deploy Stage ........................................... 495
11.3 Continuous Integration on the ABAP Server ......................... 498
  11.3.1 Quality Checking Using the ABAP Test Cockpit ..................... 499
11.3.2 Scheduling ABAP Unit Tests Using the ABAP Unit Runner .......... 502
11.4 Summary ............................................................................................... 505

12 ABAP on Git .......................................................................................... 507

12.1 Creating a Git Repository ...................................................................... 507
12.2 Installation and Setup ........................................................................... 509
12.3 Create New ABAP Objects in Eclipse .................................................... 512
12.4 Staging and Committing ABAP Code to Git ........................................... 513
12.5 Setting Up the abapGit Plugin in Eclipse ............................................. 516
12.6 Connecting to abapGit Repositories ...................................................... 516
12.7 Cloning Git Repository into Eclipse ..................................................... 517
12.8 Exporting Existing Packages as ZIP Files from a System Using abapGit .. 519
12.9 Importing Package ZIP Files into Another ABAP System Using abapGit ... 520
12.10 Summary ............................................................................................... 522

Appendices ................................................................................................. 523

A Developing Applications on the SAP Cloud Platform ......................... 523
B The Authors .............................................................................................. 553

Index ............................................................................................................. 555
Index

A

ABAP ................................................................. 478, 507, 507
commit ................................................................. 513
development tools in Eclipse ......................... 142
stage .................................................................. 513
ABAP code exit ................................................... 451
ABAP data model .................................................. 145
ABAP development tools ..................................... 48, 449
ABAP Development Workbench ......................... 142
ABAP Dictionary ............................................ 146, 148
ABAP in Eclipse .................................................. 48
ABAP objects ...................................................... 512
ABAP on Git ....................................................... 43, 507
ABAP programming model .................................. 42, 107
SAP Fiori .......................................................... 49
ABAP repository ................................................. 382
ABAP RESTful programming model .................... 42, 115
architecture ....................................................... 44
ABAP testcockpit ............................................... 478, 499
ABAP test double framework .............................. 447
ABAP unit ......................................................... 142, 444, 448
ABAP unit runner .............................................. 448, 478, 502
ABAP unit test ................................................... 449
abapGit ............................................................. 43, 507
Eclipse .............................................................. 516
plug-in .............................................................. 516
report program ............................................... 510
repository ......................................................... 516
Access control .................................................. 167
Analytical card ............................................... 297
add ................................................................. 297
annotations ...................................................... 280
Analytical Engine .............................................. 45
Analytical list page ........................................... 45, 86, 309, 329
annotation selection ...................................... 332
data connection .............................................. 331
page content .................................................... 86
page header ...................................................... 86
page tile .......................................................... 86
template ........................................................ 329
Analytical view ................................................ 309
cube ............................................................... 310
dimension ........................................................ 310
Analytical view (Cont.) ........................................ 310
text ................................................................. 310
Analytics .......................................................... 45
Annotation ......................................................... 357
@AbapCatalog.enhancementCategory: ........................................... 148
@AccessControl.authorizationCheck: ................................. 148
@Analytics.query-true ........................................ 309
@Consumption.valueHelpDefinition: ........................... 277
@DefaultAggregation:#NONE .................................. 154
@EndUserText.label ............................................. 147
@ObjectModel.compositionRoot:true .................... 172
@ObjectModel.dataCategory: ................................ 278
@ObjectModel.foreignKey. association ...................... 277
@ObjectModel.representativeKey ......................... 152
@ObjectModel.semanticKey ................................. 152
@ObjectModel.text.element: ['Description'] .............. 152
@OData.publish:true .......................................... 331
@selectionVariant ............................................. 315
@Semantics.amount.currencyCode ......................... 149
@Semantics.imageUri.true .................................. 241
@Semantics.quantity.unitOfMeasure ..................... 149
@UI.chart ...................................................... 280, 320, 327
@UI.datapoint.criticalityCalculation ................... 283
@UI.datapoint.visualization: #NUMBER ................. 280
@UI.facet.parentId ............................................ 249
@UI.facet.type:#COLLECTION ............................ 249
@UI.headerInfo ............................................... 241
@UI.hidden ..................................................... 349
@UI.identification: .......................................... 281
@UI.lineItem .................................................. 240–241, 245, 249, 349
@UI.presentationVariant .................................. 326
@UI.selectionField .......................................... 243
@UI.selectionVariant ....................................... 326
@VDM.viewType:#BASIC ...................................... 152
#ALLOWED ................................................... 148
semanticsObjectAction: ..................................... 281
Z_C_PurchaseDocumentLRP ................................ 239

555
### Index

| Annotation modeler | 336 |
| App launcher | 393 |
| dynamic | 393 |
| static | 393 |
| Application development | 352 |
| Application programming interface | 467 |
| Application tiles | 392, 405 |
| Apply stash | 427 |
| Artificial intelligence | 523 |
| Assertions | 469 |
| Authorization field | 167 |
| ACTVT | 167 |
| ZPurchOrga | 167 |
| Authorization fields | 169 |
| Authorization object | 169 |
| Automated testing | 445 |
| Axure | 55, 90 |

### B

| Bamboo | 482 |
| Bar list card | 303 |
| Behavior | 116 |
| Behavior definition | 28, 126, 172, 181 |
| Behavior Definition Language | 181 |
| Behavior handler | 172 |
| Behavior handler local class | 187 |
| lock | 195 |
| modify | 187 |
| read | 196 |
| Behavior implementation | 116, 128, 199 |
| class coding | 205 |
| classes | 171 |
| language | 186 |
| Behavior pools | 116–117, 171 |
| Behavior saver classes | 172 |
| Behavior saver local class | 198 |
| cleanup | 199 |
| save | 199 |
| Behavior-driven development | 372, 469 |
| Blue Ocean | 482 |
| Branches | 439 |
| Breakout | 260 |
| Build task | 76 |
| Business intelligence consumer services | 45 |

### C

| Business logic | 42 |
| action | 42 |
| determinations | 42 |
| validations | 42 |
| managed | 124 |
| unmanaged | 123 |
| Business Object Processing Framework | 27, 39, 498 |
| Business object generation | 171 |
| Business service | 28, 132 |
| provisioning | 116 |

| Calculated fields | 156 |
| Catalog | 387 |
| business | 388, 391 |
| link to user | 407 |
| technical | 387, 391 |
| CDS test double frameworks | 447 |
| CDS views | 36, 139 |
| basic view | 36, 139 |
| composite view | 36, 139 |
| consumption | 139 |
| create | 151 |
| interface view | 36 |
| Change and Transport System | 108 |
| Change and transport system | 478 |
| Classic online transaction processing | 45 |
| Cloud Foundry | 52, 539 |
| Code pushdown | 31 |
| Code under test | 448 |
| Code-to-Data paradigm | 31 |
| Collection path | 339 |
| Commit | 425 |
| Component.js | 396 |
| Composite interface view | 156 |
| Concours CI | 482 |
| Constructor injection | 456 |
| Consumption view | 38, 271 |
| Z_C_PurchaseDocumentOVP | 271 |
| Contact quick view | 242 |
| Content area | 321 |
| Content delivery network | 494 |

### D

| Data control file | 279 |
| Data control language | 32, 34, 174 |
| Data control language files | 167 |
| Data definition language | 26, 32, 108 |
| analytical annotations | 108 |
| search annotations | 108 |
| UI annotations | 108 |
| Data modeling | 116 |
| Default chart | 321 |
| Delete | 193 |
| export parameters | 194 |
| import parameters | 193 |
| Depended-on components | 448 |
| Dependency injection | 456 |
| Dependency inversion principle | 457 |
| Deploy stage | 495 |
| Deployment | 379 |
| Design-led development | 74 |
| design | 76 |
| develop | 76 |
| discover | 76 |

### E

| Eclipse | 142, 449 |
| Eclipse IDE | 509 |
| Eclipse Oxygen | 142 |
| Embedded analytics | 45 |
| End-to-end tests | 447 |
| Enterprise search | 27, 74 |
| Entity Data Model | 117 |
| Entity Manipulation Language | 115, 119, 179 |
| commit entity | 180 |
| modify entity | 179 |
| read entity | 180 |
| Entity relationship model | 145 |
| ESLint | 478 |
| ETag | 126 |
| External API views | 38 |
| Extreme programming | 478 |

### F

| Filter | 51 |
| Feature package locks | 50 |
| Fetch | 426 |
| Filter document items by priority | 318 |
| document items by purchasing organization | 318 |
| document items by vendor | 318 |
| document items by vendor type | 318 |
| overall item price by purchase documents | 318 |
| Filter area | 318 |
| Filter bar | 366 |
| consumption view | 276 |
| Filters visual | 335 |
| Freestyle app | 400 |
| Freestyle application | 343 |
| development | 352 |
Index

Pages expression ............................................. 369
Path expression .................................................. 369

Purchasedocumentitem .............................................. 163
OverallItemPrice .................................................. 163
PostgreSQL ........................................................... 329
Project repositories .............................................. 423
Project scope .......................................................... 93
deliverables ............................................................ 93
milestones .............................................................. 93
organization ........................................................... 93
overall goal ............................................................. 93
stakeholders ............................................................ 93
Projection behavior definition ................................... 236
Projection views ...................................................... 231
Prototyping ............................................................ 90
Pull ................................................................. 434, 441
Purchase document .............................................. 315, 361, 364, 474
ZPURCHDOCITEM .................................................. 146
ZPURCHDOCUENT .................................................. 146
Push ................................................................. 425

Q
Quality checking ..................................................... 499
Query ................................................................. 120
managed query ....................................................... 120
Query runtime ........................................................ 44
Quick actions ........................................................ 245
QUnit ................................................................. 265, 378, 445, 465, 492

R
RabbitMQ ............................................................. 52
Read ................................................................. 434, 441
export parameters .................................................. 434
import parameters .................................................. 434
Read by association .................................................. 197
export parameters .................................................. 197
import parameters .................................................. 197
Read by key ........................................................... 196
Rebase ................................................................. 426
Redis ................................................................. 52
Reference data source .................................................. 26
Remote branch ....................................................... 29
Remote function call .................................................. 29
Remote repository .................................................. 242
Reset ................................................................. 428
REST ... ................................................................. 26
Roles ................................................................. 389, 407
Runtime .............................................................. 118
S
SAP Analysis for Microsoft Excel .................................. 45
SAP API Business Hub ............................................... 38
SAP Ariba .............................................................. 57
SAP Basis .............................................................. 27
SAP Build ............................................................ 55, 90, 99
create prototype ..................................................... 92
design phase .......................................................... 95
discover phase ......................................................... 92
create phase .......................................................... 95
list report .............................................................. 99
new report ............................................................ 99
persona ............................................................... 99
prototype phase ...................................................... 97
research phase ....................................................... 94
scope phase .......................................................... 92
synthesize ........................................................... 94
validate phase ......................................................... 102
SAP Business By Design .............................................. 28
SAP Business Warehouse .......................................... 27
SAP Cloud for Customer ............................................. 57
SAP Cloud Platform .................................................. 51, 523
application programming model .................................. 524
DevOps ............................................................... 52
setup SAP Web IDE ................................................. 526
SAP Cloud Platform Integration ..................................... 52
SAP CoPilot ............................................................ 61, 71
business objects ..................................................... 73
chat ................................................................. 73
notes ................................................................. 72
quick actions ......................................................... 73
screenshots ........................................................... 72
SAP Development Tools .............................................. 143
SAP ERP .............................................................. 24
SAP Fiori .............................................................. 23, 29–30, 55–56
adaptive design ..................................................... 66
design principles ..................................................... 64
develop ............................................................... 139
elements ............................................................. 77
elements list report .................................................. 139
SAP Fiori (Cont.)

extending the UI .................................................... 260
frontend server ...................................................... 29
marquee .............................................................. 525
object page .......................................................... 139
responsive design .................................................. 66
UI ................................................................. 260
UX ................................................................. 41, 47, 65
SAP Fiori 2.0 ........................................................... 58
SAP Fiori 3.0 ........................................................... 58
consistency ........................................................... 59
integration ........................................................... 60
intelligence .......................................................... 59
situation .............................................................. 59
SAP Fiori apps .......................................................... 76
analytical list page .................................................. 85
elements ............................................................. 77
freestyle .............................................................. 76
list report ............................................................. 78
object page .......................................................... 80
overview page ....................................................... 83
worklist page ........................................................ 87
SAP Fiori Elements

application in service binding ..................................... 230
SAP Fiori elements .................................................... 477
SAP Fiori launchpad .................................................... 413
admin pane .......................................................... 386
catalog ............................................................... 387
components ........................................................ 74
content manager .................................................... 389
groups ............................................................... 388
home page ........................................................... 69
Me Area .............................................................. 69
notifications ......................................................... 70
roles ................................................................. 389
UX ................................................................. 47
viewpoint ............................................................ 67
SAP Fiori search ....................................................... 30
SAP Fiori UI ........................................................... 29
SAP Gateway .......................................................... 24
embedded deployment .............................................. 354
hub deployment ..................................................... 29
service builder ...................................................... 344
SAP QUI .............................................................. 24, 30, 55
SAP HANA ............................................................ 26
architecture .......................................................... 26
database table ....................................................... 146
SAP HANA (Cont.)

to enterprise cloud .................................................. 50
SAP Leonardo ........................................................ 52
SAP Lumira ............................................................ 45
SAP NetWeaver ....................................................... 24, 48, 110
ABAP ................................................................. 27
ABAP backend server .............................................. 144
application server for ABAP ........................................ 49
backend ............................................................. 48
SAP NetWeaver Application Server for
ABAP ................................................................. 24
SAP Notes ............................................................ 48
SAP 5/4HANA ......................................................... 23, 445
applications .......................................................... 477
architecture .......................................................... 23
backend ............................................................. 425
backend system ...................................................... 31
basic interface view layer ......................................... 154
cloud ................................................................. 48
on-premise ........................................................... 48
on-premise architecture ............................................ 48
release strategies ..................................................... 50
UX ................................................................. 45, 47
virtual data model ................................................... 141
SAP 5/4HANA Cloud .................................................. 23, 48, 523
architecture .......................................................... 48
private cloud ........................................................ 49
public cloud ........................................................ 49
release strategies ..................................................... 50
SDK ................................................................. 52
SAP SuccessFactors Employee Central ................................ 57
SAP Tax Service ..................................................... 52
SAP Web IDE ......................................................... 52, 256, 280, 291, 329, 419
business application .................................................. 529
cloning project ....................................................... 431
deployment .......................................................... 379
export application .................................................. 382
interface extensions ................................................ 260
overview page template project .................................. 291
template ............................................................ 329
SAP Web IDE Full-Stack ............................................ 352
SAP Cloud Platform development .................................. 526
SAPUI5 ............................................................... 15, 30, 47, 58, 348, 445
application .......................................................... 353
freestyle .............................................................. 477
grunt ................................................................. 487
Stefan Haas is currently working as an IT Specialist at BMW Group’s IT center in Munich, Germany. Before that, he was a senior developer at SAP working on enterprise asset management applications for SAP S/4HANA within the digital supply chain and manufacturing lines of business. He is an experienced full stack developer on all layers of SAP S/4HANA, ranging from SAP HANA, core data services, and ABAP to OData, SAP Gateway, SAP Fiori, SAPUI5, and JavaScript. Additionally, he was a fellow at the machine learning engine team at SAP’s predictive maintenance and service cloud solution. He holds a master’s degree in computer science, with a minor in business, from Ludwig Maximilian University of Munich and is a certified Cloud Foundry developer.

Bince Mathew is a development architect at SAP with more than ten years of experience in providing solutions on various SAP mobility platforms. He is an expert in areas such as SAP Fiori, SAP S/4HANA, SAP Syco, SAP Mobile Platform (formerly Sybase Unwired Platform), SAP Afaria, and the Internet of Things. He has worked with industry verticals such as manufacturing and retail. He is an SAP Certified Technology Associate for SAP Fiori and a published author with a focus on SAP Fiori developments. He was also given the best speaker award for SAP Fiori at the SAP Inside Track event held by SAP. He is active in the SAP Community on topics related to SAP mobility.