# Глава 7. Разработка freestyle приложения

*В этой главе вы узнаете, как разработать приложение SAPUI5 в свободном стиле с использованием интеллектуальных элементов управления и полного стека SAP Web IDE. Разработка приложения свободного стиля может потребоваться, если тип приложения элементов SAP Fiori не подходит для вашего конкретного варианта использования и дизайна пользовательского интерфейса.*

В предыдущих главах вы узнали, как реализовать несколько различных приложений-шаблонов элементов SAP Fiori. Эти приложения на основе шаблонов значительно сокращают объем необходимого внешнего кода JavaScript и SAPUI5 и предоставляют несколько универсальных настраиваемых и расширяемых шаблонов для различных сценариев приложений. Однако в некоторых сценариях шаблоны могут не соответствовать дизайну пользовательского интерфейса и макетам, которые вы разработали на этапе проектирования. В этом случае вы должны разработать приложение SAPUI5 свободного стиля. К счастью, с введением шаблонов приложений элементов SAP Fiori было введено несколько интеллектуальных элементов управления для реализации концепции элементов SAP Fiori на более низком уровне, которые также используются внутри шаблонов элементов SAP Fiori. Интеллектуальные элементы управления используют аннотации OData, чтобы сократить требуемый интерфейсный код JavaScript, но при этом обеспечить гибкость использования произвольного макета приложения. В этой главе вы познакомитесь с наиболее распространенными интеллектуальными элементами управления. После знакомства с каждым элементом управления мы продемонстрируем их использование, разработав простое произвольное приложение для создания и отображения документов о покупке. Кроме того, вы узнаете, как настроить проект приложения свободного стиля SAPUI5 в полном стеке SAP Web IDE, как установить соединение с серверной службой OData, как настроить навигацию между различными представлениями и как написать одностраничный (OPA5) интеграционный тест пользовательского интерфейса (UI) для проверки всего сценария приложения.

## 7.1 Smart Controls

Интеллектуальные элементы управления являются частью библиотеки *sap.ui.comp* SAPUI5, которая содержит составные элементы управления SAPUI5. Как следует из названия, элементы управления в этой библиотеке представляют собой комбинацию нескольких базовых элементов управления SAPUI5, например, из библиотеки *sap.m*, и объединены в новый повторно используемый элемент управления. Более того, помимо объединения нескольких элементов управления, элементы управления, начинающиеся с префикса *smart*, анализируют метаданные OData или аннотации пользовательского интерфейса, чтобы сократить необходимый внешний код XML и JavaScript SAPUI5. Например, если *SmartField* обнаруживает аннотацию OData справки по значению в документе метаданных службы для свойства, к которому оно привязано, *SmartField* автоматически отображает справку по значению. В следующих разделах мы представим наиболее распространенные интеллектуальные элементы управления и их основные аннотации.

### 7.1.1 Смарт-поле (SmartField)

Элемент управления *sap.ui.comp.smartfield.SmartField* SAPUI5 является оболочкой для нескольких других элементов управления, которые отображаются на основе предоставленных аннотаций OData соответствующего связанного свойства типа сущности OData. Целью *SmartField* является отображение или изменение значений поля базовой сущности OData. В листинге 7.1 показан пример того, как можно использовать *SmartField* внутри определения XML-представления SAPUI5. В этом примере *SmartField* привязан к свойству *PurchasingOrganization* через свойство *value*. Префикс *smartField* определен для пространства имен XML *«sap.ui.comp.smartfield»* *(xmlns:smartField="sap.ui.comp.smartfield")*.

<smartField:SmartFieldvalue="{PurchasingOrganization}"/>

Листинг 7.1. SmartField, привязанный к свойству PurchasingOrganization объекта Z\_C\_PurchaseDocumentLrp

Во время выполнения *SmartField* отображается как поле ввода с соответствующей меткой и справкой по значению. Какие именно элементы управления отображаются, зависит от атрибутов свойства связанного типа сущности аннотаций в документе метаданных службы OData.

В нашем примере поле отображается как редактируемое поле ввода, поскольку его элемент свойства не содержит XML-атрибутов *sap:creable="false"* и *sap:updatable="false"* и имеет модель данных объекта (EDM) *Edm.String*, как показано на рис. 7.1. Кроме того, метка поля автоматически отображается без необходимости явного определения элемента управления меткой и берется из атрибута *sap:label* соответствующего элемента свойства. Источником значения атрибута *sap:label* обычно является метка поля его элемента данных или аннотация *@EndUser.textLabel* на уровне основных служб данных (CDS). Кроме того, также можно перезаписать метки полей в проекте построителя сервисов SAP Gateway (транзакция SEGW). Атрибут *sap:value-list* указывает, что стандартная справка по значениям доступна и будет отображаться.

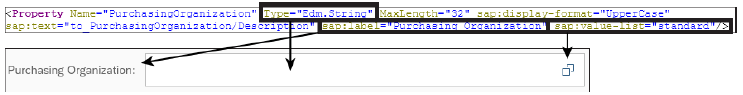


Рис. 7.1 Аннотации документа метаданных службы OData, интерпретируемые элементом управления SmartField

На рис. 7.2 показаны необходимые аннотации OData для отображения справки по стандартным значениям. Справка по значениям основана на определении справки по значениям для представления *Z\_I\_PurchasingOrganization* CDS; он принимает поле *PurchasingOrganization* в качестве входного и исходящего аргумента и дополнительно отображает описание закупочной организации.

Входящий и исходящий аргумент — это когда поле *PurchasingOrganization* берется в качестве входных данных для фильтрации данных справки по значениям по *PurchasingOrganizations* и дополнительно также отображается в выходной таблице справки по значениям.



Рис. 7.2 Аннотации справки по значению для поля PurchasingOrganization

### 7.1.2 Смарт-ссылка (Smart Link)

Элемент управления *sap.ui.comp.navpopover.SmartLink* отображает ссылку для поля и предоставляет всплывающее окно, содержащее навигационные ссылки на другие связанные приложения, как показано на рис. 7.3. Для создания таких ссылок этот элемент управления анализирует аннотацию OData *SemanticObject* связанного свойства типа сущности OData и собирает все цели навигации пользователя, содержащие соответствующий семантический объект — в данном случае *PurchasingDocument*. В нашем примере у пользователя есть доступ к *PurchasingDocument* — создание (создание документа покупки), мониторинг (страница обзора документов покупки) и анализ приложений KPIDetails (документы покупки). Возможные цели навигации ограничены целями, содержащимися в назначенных пользователю бизнес-ролях интерфейса транзакции PFCG. В сценарии концентратора SAP Gateway роли транзакций внешнего интерфейса PFCG, содержащие соответствующие бизнес-каталоги, целевые сопоставления и группы, назначаются пользователю во внешнем интерфейсе или в системе-концентраторе. Чтобы сгенерировать аннотацию *SemanticObject* OData, поле должно быть аннотировано ***@Consumption.semanticObject:<SEMANTIC\_OBJECT>*** на уровне языка определения данных CDS (DDL). Как и аннотации пользовательского интерфейса, аннотации являются частью отдельной модели аннотаций, предоставляемой службой каталогов SAP Gateway при создании службы OData с использованием сценария Auto-Exposure или Reference Data Source (RDS).

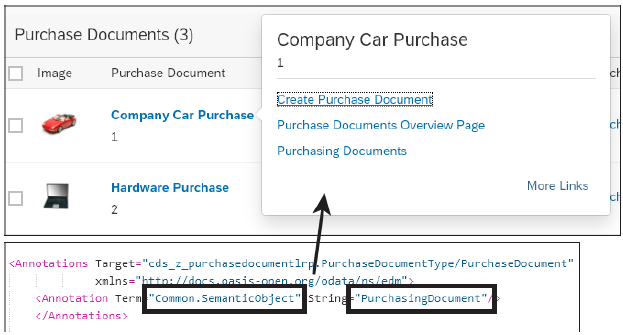


Рис. 7.3 Смарт-ссылка и ее всплывающее окно навигации, содержащее ссылки на связанные приложения с аннотированным семантическим объектом

*Примечание*

*Элемент управления sap.ui.comp.navpopover.SmartLink будет работать только на панели запуска SAP Fiori, но не при локальном тестировании приложения в SAP Web IDE. Пользователь увидит только навигационные ссылки и получит доступ к приложениям, которые являются частью назначенных ему бизнес-ролей PFCG внешней транзакции и чей семантический объект, определенный в их целевых сопоставлениях, соответствует семантическому объекту, определенному для поля. Доступные целевые сопоставления для пользователя, содержащие семантические объекты и действия, для которых у пользователя есть авторизация, извлекаются при запуске панели запуска SAP Fiori.*

### 7.1.3 Смарт-форма (SmartForm)

Элемент управления *sap.ui.comp.smartform.SmartForm* в сочетании с элементами управления *SmartField* позволяет создавать формы с минимальными усилиями. Этот элемент управления содержит агрегацию группы (*sap.ui.comp.smartform.Group*), которая содержит агрегацию *GroupElements (sap.ui.comp.smartform.GroupElement)*. *GroupElement* представляет собой комбинацию метки и различных элементов управления, связанных с этой меткой, обычно *SmartField*. В листинге 7.2 показана общая структура определения *SmartForm*, включая элементы управления *Group, GroupElement* и *SmartField*.

<SmartFormtitle=”SmartFormTitle”>

<Grouplabel=”GroupLabel”>

<GroupElement>

<SmartFieldvalue=”{ODATA\_ENTITY\_TYPE\_PROPERTY}”/>

</GroupElement>

</Group>

<Group>

…

</Group>

</SmartForm>

Листинг 7.2. Структура элементов SmartForm, состоящая из элементов управления Group, GroupElement и SmartField

Кроме того, *SmartForm* содержит агрегацию макета, которую можно использовать для настройки *ResponsiveGridLayout*, которая используется внутри *SmartForm* для выравнивания элементов управления в зависимости от доступного пространства. Макет организует свои дочерние элементы управления в ряд из 12 столбцов. Используя свойства элемента управления, вы можете указать, сколько столбцов должны занимать элементы управления в зависимости от доступного размера экрана.

В листинге 7.3 показан пример использования элемента управления *sap.ui.comp.smartform.Layout* с агрегацией макета для настройки макета *SmartForm*.

<SmartForm>

<layout>

<LayoutemptySpanL="1"emptySpanM="0"labelSpanL="3"labelSpanM="3"/>

</ layout>

</SmartForm>

Листинг 7.3. Пример макета для SmartForm

Атрибут ***emptySpanL="1"*** указывает, что на большом экране одна ячейка сетки в конце каждой строки будет пустой. На экране среднего размера вам понадобится все пространство, поэтому установите для него значение ***emptySpanM="0"***. На экранах большого и среднего размера элемент управления label не должен занимать более 3 столбцов из 12-столбцового пространства сетки (***labelSpanL="3"*** *и* ***labelSpanM="3"***).

Еще одной особенностью элемента управления *SmartForm* является возможность включить кнопку проверки формы, установив ***checkButton="true"***. Если вы нажмете эту кнопку, значения *SmartField*, содержащиеся в форме, будут проверены на соответствие их аннотациям метаданных OData. На рис. 7.4 показан пример проверки, запускаемой кнопкой «Проверить».

Свойство *PurchaseDocument* помечено как обязательное с помощью атрибута ***Nullable="false"*** в аннотациях метаданных OData. Поэтому, если проверка определяет, что *SmartField* пуст, это поле будет отмечено красным. Когда пользователь щелкает поле, отображается соответствующее сообщение об ошибке. Но поскольку мы уже сделали поле *PurchaseDocument* доступным только для чтения (поскольку мы генерируем его значение в бэкэнде), мы сделаем это поле скрытым.

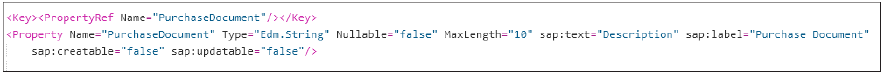


Рис. 7.4 Пример проверки обязательного поля, запускаемой нажатием кнопки «Проверить»

### 7.1.4 Смарт-таблица (Smart Table)

Элемент управления *sap.ui.comp.smarttable.SmartTable* представляет собой оболочку, содержащую таблицы SAPUI5. Как и другие интеллектуальные элементы управления, он анализирует аннотации метаданных службы OData (*$metadata*) и аннотации пользовательского интерфейса, предоставляемые отдельной моделью аннотаций и службой каталогов концентратора SAP Gateway.

/sap/opu/odata/IWFND/CATALOGSERVICE;v=2/Annotations(TechnicalName=’<ANNOTATION\_MODEL>’,Version='0001')/$value/

Аннотации пользовательского интерфейса для приложений Smart Controls и элементов SAP Fiori предоставляются не через стандартную службу OData приложения, а через отдельную модель аннотаций, которая предоставляется через отдельную службу OData под названием *CATALOGSERVICE; v=2*. Вы найдете эту услугу в наборе доступных услуг в системе SAP Gateway. Элемент управления *SmartTable* отображает таблицу для определенного набора сущностей OData и может создавать таблицы только со строками представления XML. Поскольку элемент управления *SmartTable* также используется в шаблоне отчета со списком элементов SAP Fiori, одни и те же аннотации пользовательского интерфейса можно использовать для обоих случаев.

На рис. 7.5 показаны некоторые аннотации *UI.LineItem* и то, как они интерпретируются элементом управления *SmartTable*. Как и в приложении списка элементов SAP Fiori, аннотацию *@UI.lineItem* необходимо использовать на уровне CDS DDL, чтобы указать видимые столбцы таблицы по умолчанию и их последовательность. Критичность столбцов *OverallPrice* и *ApprovalRequired* указывается полем *OverallPriceCriticality*. Эта информация также предоставляется аннотацией line item.

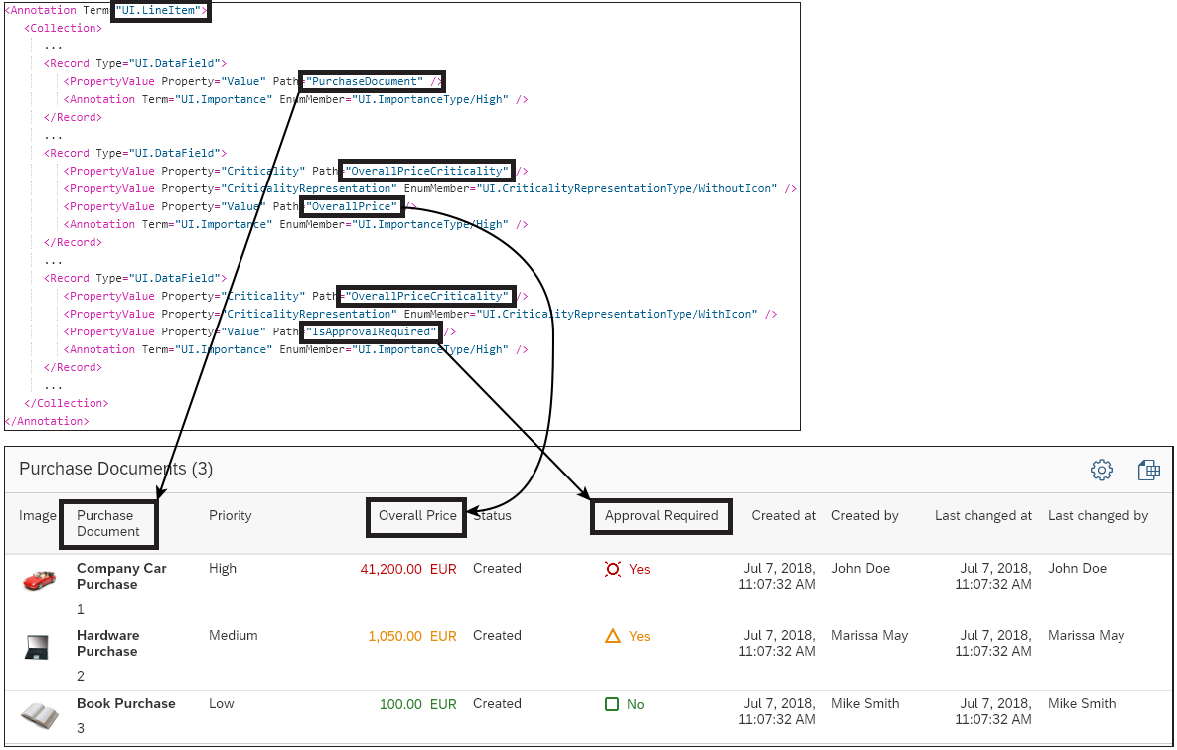


Рис. 7.5 Пример интерпретации аннотаций UI.LineItem элементом управления SmartTable

Вы можете скрыть определенные поля от пользовательского интерфейса, но не полностью от типа сущности. Таким образом, например, вы можете использовать эти поля в качестве индикатора критичности, вы должны аннотировать эти поля аннотацией *@UI.hidden*. В листинге 7.4 показано, как добавить аннотацию к полю *OverallPriceCriticality*, поскольку мы не хотим делать это поле доступным в каталоге полей таблицы, а вместо этого нуждаемся в нем как в техническом поле для указания критичности.

@UI.hidden:true

OverallPriceCriticality;

Листинг 7.4. Скрытие поля из пользовательского интерфейса, но не из типа сущности OData

Определить смарт-таблицу в XML-представлении SAPUI5 довольно просто. В листинге 7.5 показаны минимальные значения свойств, необходимые для отображения таблицы. Префикс *smartTable* определяется для пространства имен *sap.ui.comp.smarttable (xmlns:smartTable="sap.ui.comp.smarttable")*. *entitySet* определяет набор сущностей OData, к которому будет привязана таблица. Атрибут *header* задает текст заголовка таблицы, а атрибут *tableType* указывает используемый тип таблицы — в нашем случае это *ResponsiveTable*. Если для атрибута *enableAutoBinding* задано значение *true*, данные для указанного набора сущностей будут немедленно запрашиваться из службы OData, развернутой в системе-концентраторе SAP Gateway, при отображении таблицы.

<smartTable:SmartTableentitySet="Z\_C\_PurchaseDocumentLrp"header=

"PurchaseDocuments"enableAutoBinding="true"tableType="ResponsiveTable"/>

Листинг 7.5. Определение SmartTable в XML-представлении SAPUI5

Обычно *SmartTable* используется в сочетании со *SmartFilterBar*. Два элемента управления связаны с использованием идентификатора *SmartFilterBar*. Следовательно, *SmartTable* должен определить идентификатор *SmartFilterBar*, используя его атрибут *smartFilterId*.

### 7.1.5 Smart Filter Bar – Панель интеллектуальных фильтров

Элемент управления *sap.ui.comp.smartfilterbar.SmartFilterBar* анализирует метаданные OData и аннотации пользовательского интерфейса и отображает элемент управления *FilterBar*, который можно использовать для фильтрации *SmartTable*. На рис. 7.6 показаны некоторые метаданные и аннотации пользовательского интерфейса, интерпретируемые элементом управления *SmartFilterBar*. Начальные отображаемые поля в расширенной панели фильтров должны быть снабжены аннотацией OData *UI.SelectionField* *(@UI.selectionField* на уровне CDS DDL). В этом примере поля *PurchaseDocument, Priority, Status* и *PurchasingOrganization* были аннотированы как поля выбора. Какой тип справки по значениям отображается для поля выбора, зависит от атрибутов ***sap:value-list*** соответствующего свойства типа объекта; например, для поля «Приоритет» будет отображаться справка в виде раскрывающегося списка, поскольку его атрибут *sap:value-list* имеет фиксированное значение. Для атрибута *sap:value-list* свойства *PurchasingOrganization* задано значение ***standard***, поэтому будет отображаться справка по стандартному значению.

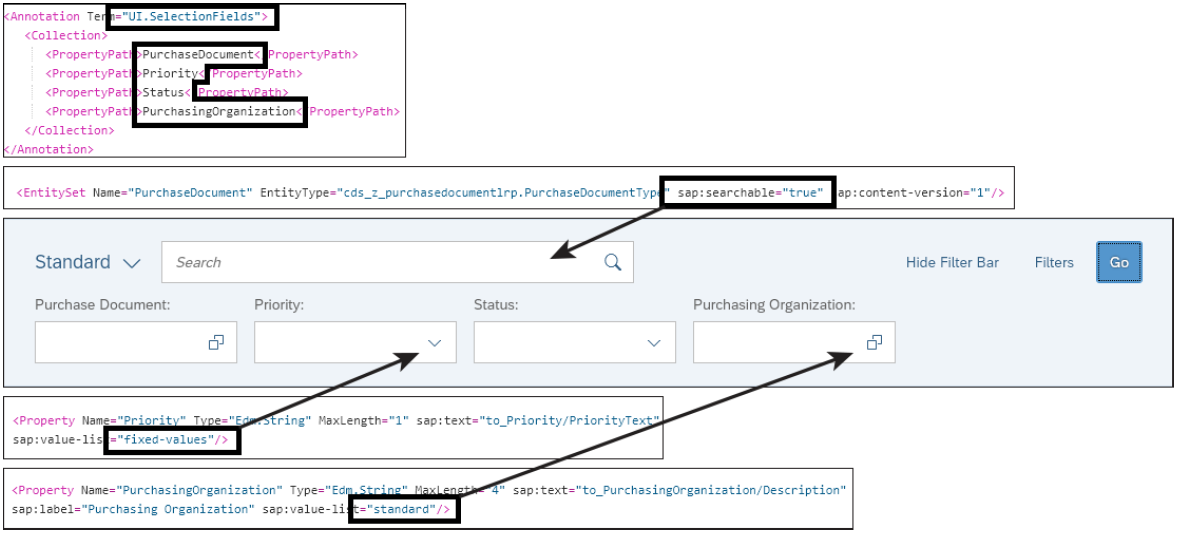


Рис. 7.6. Некоторые метаданные и аннотации пользовательского интерфейса, проанализированные панелью смарт-фильтров

В XML-представлении SAPUI5 элемент управления *SmartFilterBar* может быть объявлен, как показано в листинге 7.6. Префикс *smartFilterBar* определен для пространства имен *sap.ui.comp.smartfilterbar* (*xmlns:smartFilterBar=" sap.ui.comp.smartfilterbar"*). Атрибут ***entitySet*** определяет набор сущностей OData, чьи метаданные используются для отображения панели смарт-фильтров. Ключ постоянства указывает, что ключ, который используется для хранения данных персонализации, например, вариант фильтра, который необходимо использовать периодически. Хотя набор сущностей может поддерживать поиск с произвольным текстом, на что указывает атрибут ***sap:searchable="true"***, поле поиска с произвольным текстом не отображается автоматически. Чтобы включить его на панели интеллектуального фильтра, атрибут ***enableBasicSearch*** должен быть явно установлен в значение ***true***.

<smartFilterBar:SmartFilterBarid="purchaseDocumentsSmartFilterBar"entitySet=

"PurchaseDocument"persistencyKey=

"purchaseDocumentsFilterBarKey"enableBasicSearch="true"/>

Листинг 7.6. Определение SmartFilterBar в XML-представлении SAPUI5

Чтобы полностью удалить поле из набора полей выбора панели интеллектуальных фильтров, вы аннотируете поле ***@Consumment.filter.hidden: true*** на уровне CDS DDL, как показано в листинге 7.7. На уровне службы OData эта конфигурация создаст аннотацию пользовательского интерфейса OData ***UI.HiddenFilter*** для соответствующего поля.

@Consumption.filter.hidden:true

PurchaseDocumentImageURL

Листинг 7.7. Скрытие поля из набора доступных полей выбора на панели смарт-фильтров

*Примечание*

*Смарт-таблица, а также панель смарт-фильтров поддерживают управление вариантами, как только вы предоставляете уникальный ключ через их свойство* ***persistencyKey****, в котором могут храниться настройки. Варианты будут храниться в многоуровневом репозитории интерфейсной системы концентратора SAP Gateway. Чтобы сохранить настройки смарт-таблицы и панели фильтров в одном варианте, вы можете использовать элемент управления смарт-вариантом, чтобы включить управление вариантами страницы. Дополнительную информацию можно найти в разделе Smart Variant Management в документации для SAPUI5 Software Development Kit (SDK) по адресу* [*http://s-prs.co/498820*](http://s-prs.co/498820)*.*

## 7.2 Разработка приложений с помощью SAP Web IDE Full-Stack

В этом разделе мы разработаем небольшое приложение SAPUI5, состоящее из *SmartForm* для создания документов о покупке, а также интеллектуальной панели фильтров и таблицы для отображения документов о покупке. Мы разработаем интерфейсную часть приложения в полном стеке SAP Web IDE и повторно используем службу OData, разработанную в главе 4, для отчета со списком элементов SAP Fiori и страницы объектов. Мы проведем вас через все необходимые шаги для разработки приложения, включая настройку подключения к службе OData и навигацию по дескриптору приложения (*manifest.json*), а также создание интеграционного теста OPA5, охватывающего весь сценарий создания и отображения.

### 7.2.1 Настройка службы OData

Мы начнем разработку нашего приложения, сначала создав начальную структуру проекта и настроив подключение службы OData к серверной части. Чтобы создать новый проект SAPUI5 произвольного стиля в полном стеке SAP Web IDE, выберите **File - New - Project from Template** в строке меню верхнего уровня. На начальном этапе выбора шаблона мастера создания проекта, показанного на рис. 7.7, выберите шаблон приложения SAPUI5, который содержится в категории приложений SAP Fiori. Шаблон приложения SAPUI5 представляет собой пустой скелет проекта SAPUI5, который включает начальное представление и соответствующий ему контроллер. Нажмите «Далее».

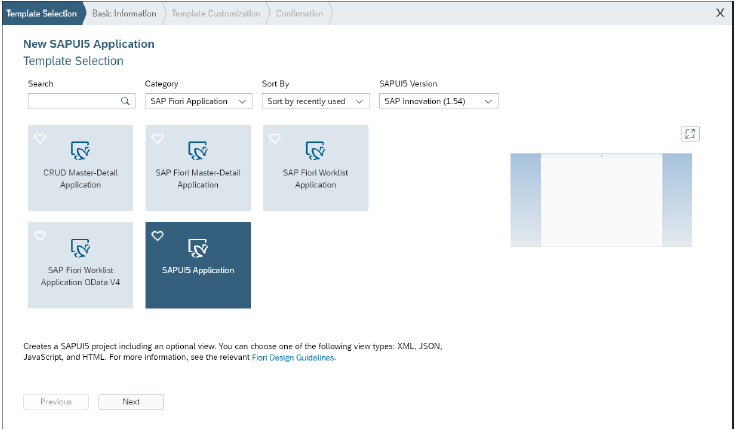


Рисунок 7.7 Создание нового приложения SAPUI5 в SAP Web IDE Full-Stack

На следующем экране мастера создания проекта, показанного на рис. 7.8, укажите имя проекта и пространство имен, с помощью которых можно добавить префикс к артефактам SAPUI5. Нажмите «Далее».

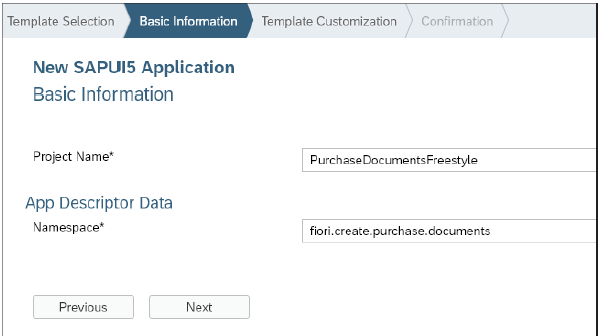


Рисунок 7.8 Предоставление имени проекта и пространства имен для проекта Freestyle SAPUI5

На шаге настройки шаблона, показанном на рис. 7.9, укажите View Type и View Name для Вида. View Type *XML* является рекомендуемым для определения вида в SAPUI5. В нашем примере давайте назовем Вид «CreatePurchaseDocument». Нажмите «Готово».

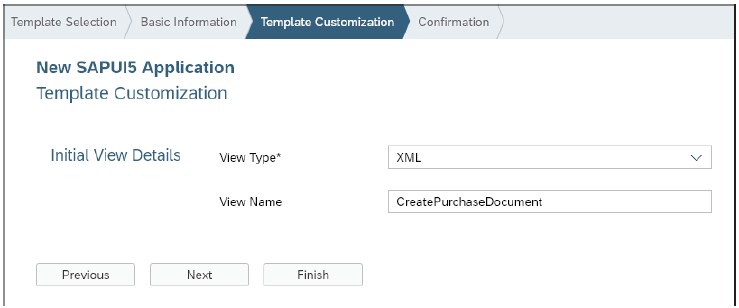


Рис. 7.9 Выбор типа Вида и определение имени Вида

Чтобы настроить подключение к серверной службе OData, которую мы разработали в главе 4, откройте файл ***manifest.json*** дескриптора приложения. ***Дескриптор приложения*** *— это центральный репозиторий, содержащий все конфигурации, относящиеся к приложению, включая источники данных приложения, модели и параметры маршрутизации.* Чтобы включить службу OData, включая ее модель аннотаций пользовательского интерфейса, для использования в приложении, добавьте службу в источники данных приложения, как показано на рис. 7.10. Как правило, дескриптор приложения можно редактировать двумя способами: с помощью редактора дескрипторов на основе форм или редактора кода на основе нотации объектов JavaScript (JSON). Используйте редактор дескрипторов и нажмите кнопку «Добавить» службы OData (+).

Откроется диалоговое окно мастера, показанное на рис. 7.11, в котором вы настроите соединение данных. Сначала войдите в хаб-систему SAP Gateway, в которой была зарегистрирована служба. Затем выберите службу, выполнив поиск в поле поиска Службы. Когда вы нашли и выбрали службу, нажмите «Далее».

На шаге выбора модели, показанном на рис. 7.12, укажите имя модели для ранее определенного подключения службы OData или установите его в качестве модели приложения по умолчанию. В нашем примере мы хотим установить подключение к данным в качестве модели по умолчанию, поскольку это основная служба OData приложения и, следовательно, его источник данных по умолчанию. Нажмите «Далее», а затем «Готово».

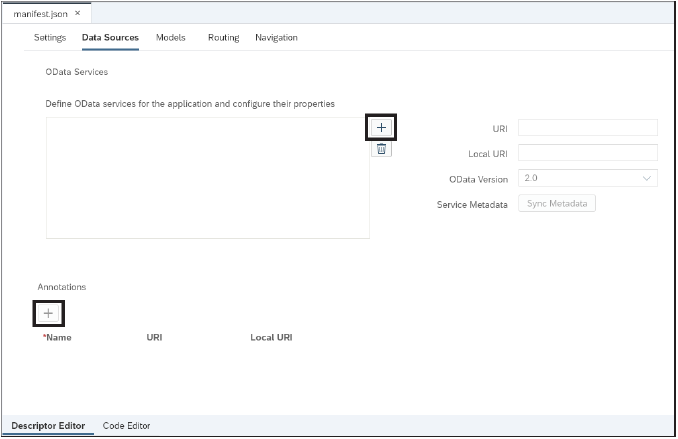


Рисунок 7.10 Редактор дескрипторов для редактирования источников данных файла manifest.json дескриптора приложения

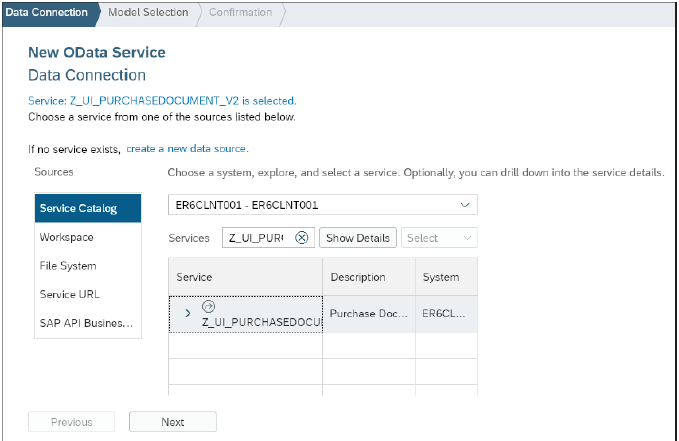


Рис. 7.11. Выбор службы OData для добавления в качестве источника данных в дескриптор приложения

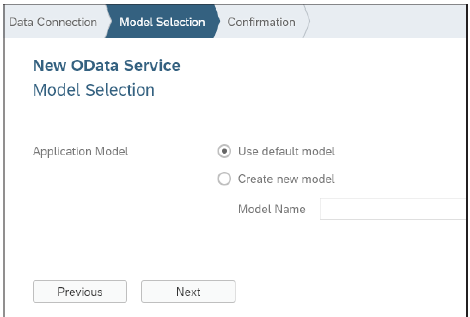


Рис. 7.12 Установка ранее созданного подключения к данным в качестве модели приложения по умолчанию

Теперь, когда служба OData выбрана и назначена модели по умолчанию, настройте источник аннотаций, чтобы разрешить использование интеллектуальных элементов управления, как показано на рис. 7.13.

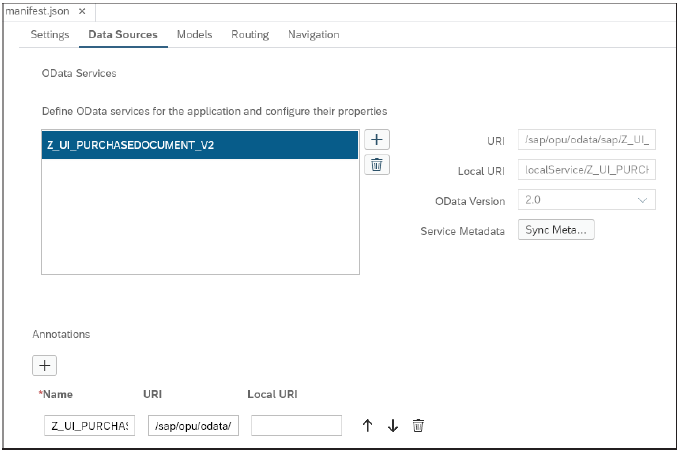


Рис. 7.13 Добавленная служба OData и соответствующий файл аннотаций

Нажмите кнопку Add New Annotation (+) в разделе «*Annotations*» экрана под разделом «*OData Services*». Давайте назовем источник данных аннотаций «*Z\_UI\_PURCHASEDOCUMENT\_V2\_VAN*», как фактическую модель аннотаций OData в бэкэнде, и установим для его URI значение

*«/sap/opu/odata/IWFND/CATALOGSERVICE;v=2/Annotations(TechnicalName='Z\_UI\_PURCHASEDOCUMENT\_V2\_VAN',Version='0001')/$value/»,*

поскольку аннотации необходимо загружать через службу каталогов системы-хаба SAP Gateway.

Чтобы записать данные обратно в модель и службу OData, вам необходимо установить режим привязки модели на двусторонний. Выберите вкладку «*Models»* в редакторе дескрипторов, а затем выберите *TwoWay* в раскрывающемся списке Default Binding Mode, как показано на рис. 7.14.

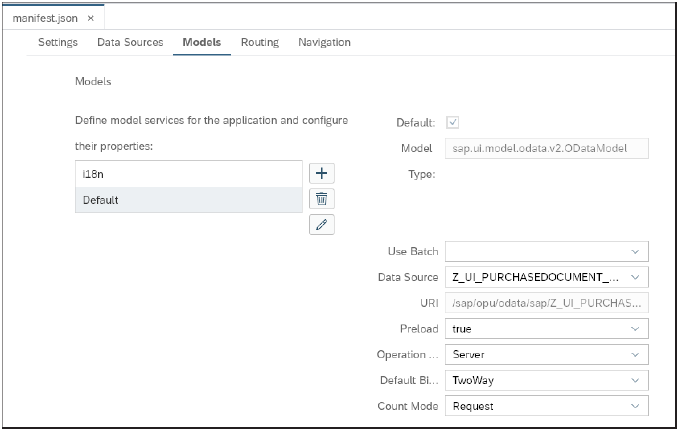


Рисунок 7.14 Установка режима привязки модели по умолчанию на TwoWay

Рисунок 7.15 и Рисунок 7.16 обобщают сделанные до сих пор изменения. На рис. 7.15 показаны источники данных приложения в редакторе кода JSON файла дескриптора приложения *manifest.json*. Мы добавили службы аннотаций Z\_UI\_PURCHASEDOCUMENT\_V2 OData и Z\_UI\_PURCHASEDOCUMENT\_V2\_VAN в качестве источников данных в приложение. Служба OData ссылается на свои аннотации пользовательского интерфейса через свойство *annotations*, указывающее на службу аннотаций.



Рис. 7.15 Источники данных OData и OData Annotation для приложения в редакторе кода дескриптора приложения (manifest.json)

На рис. 7.16 показаны модели приложений. Модель приложения по умолчанию ссылается на службу OData Z\_UI\_PURCHASEDOCUMENT\_V2 и настраивает службу OData, используя дополнительные свойства, например режим привязки и подсчета. Модель, которая будет создана файлом компонента пользовательского интерфейса (*Component.js*) при запуске приложения, представляет собой модель *sap.ui.model.odata.v2.ODataModel*. Модель *i18n* предоставляет тексты, зависящие от языка, которые хранятся локально в файлах *i18n-\*.properties*, например, *i18n-en.properties* для английского языка. Доступ к текстам, хранящимся как свойства в файлах *i18n-\*.properties*, можно получить в представлениях XML через модель *i18n* с использованием относительных привязок *({i18n>PROPERTY})*. Для простоты мы будем жестко кодировать тексты пользовательского интерфейса, а не извлекать их из зависимых от языка файлов *i18n-\*.properties*.



Рис. 7.16. Модели OData и i18n приложения по умолчанию в редакторе кода дескриптора приложения (manifest.json)

### 7.2.2 Страница создания объекта с использованием смарт-полей и форм

Наше приложение теперь подключено к модели OData, и вы можете начать собственно разработку приложения. Напомним, что когда мы создавали проект в предыдущем разделе, SAP Web IDE уже создала для нас Вид, который мы назвали *CreatePurchaseDocument.view.xml*. В качестве следующего шага нам нужно будет вставить *SmartForm* для создания документов покупки в пустое в настоящее время определение представления, которое содержит несколько *SmartField* для ввода исходных данных документа покупки. Кроме того, мы добавим кнопку сохранения в нижний колонтитул страницы, чтобы инициировать отправку документа о покупке в серверную службу OData. Окончательное определение *SmartForm* показано на рис. 7.17. Вам потребуется установить для редактируемого атрибута *SmartForm* значение *true*, чтобы отобразить поля *sap.m.Input* вместо полей *sap.m.Text* для различных полей *SmartField*. Вы также установите для атрибута *checkButton* значение *true*, чтобы включить оценку входных данных по сравнению с метаданными OData соответствующего связанного свойства типа сущности OData.



Рисунок 7.17 Определение смарт-формы для создания документа покупки

Внутри *SmartForm* вы можете определить только одну группу без метки, но вы можете добавить несколько элементов управления *GroupElement*, включая *SmartField*, в агрегацию *groupElements* группы по умолчанию. В частности, вы добавите поля *SmartFields* для свойств *PurchaseDocument ID, Description, Priority, PurchasingOrganization* и *PurchaseDocumentImageURL* объекта OData *Z\_C\_PurchaseDocumentLRP* и привяжете их значения к соответствующему атрибуту *value* элемента управления *SmartField*.

Однако для визуализации пользовательского интерфейса на основе метаданных OData и аннотаций пользовательского интерфейса *SmartForm* его дочерние элементы управления требуют привязок к свойствам объекта OData для получения информации о своих метаданных. Привязка к вновь созданному объекту документа покупки может быть установлена ​​через соответствующий Контроллер. Когда исходный файл Вида *CreatePurchaseDocument.view.xml* был создан SAP Web IDE, также был создан соответствующий файл *CreatePurchaseDocument.controller.js*, который служит контроллером. Поскольку мы добавили службу OData в качестве модели по умолчанию в дескриптор приложения, экземпляр *sap.ui.model.odata.v2.ODataModel* будет автоматически создан платформой SAPUI5. Автоматически созданные Модели можно получить в Контроллерах, вызвав *this.getView().getModel()*. Извлечение модели из контроллера происходит довольно часто, поэтому имеет смысл перенести эту функциональность в модуль *BaseController.js*, от которого наследуются другие контроллеры представлений, как показано в листинге 7.8. Затем вы можете получить модель OData внутри контроллеров представления, просто вызвав *this.getModel()*.

sap.ui.define([

"sap/ui/core/mvc/Controller" ],

function(Controller){

"usestrict";

returnController.extend("…/BaseController", {

…

getModel:function(sName){

var oModel=this.getView().getModel(sName);

if(!oModel){

oModel = this.getOwnerComponent().getModel(sName);

}

return oModel;

}

…

)};

)};

Листинг 7.8. Модуль BaseController.js, содержащий удобный метод getModel для получения Модели внутри контроллера

Создание нового пустого документа покупки в Контроллере и привязка его к Виду показано в листинге 7.9. В методе *onInit* Контроллера *CreatePurchaseDocument* вы присоединяетесь к событию *route-matched*, которое запускается каждый раз, когда отображается маршрут *CreatePurchaseDocument* (точнее, всякий раз, когда хэш URL-адреса соответствует шаблону, определенному для маршрута). Когда событие запускается, вызывается функция обратного вызова *\_createPurchaseDocument*. Внутри функции вы будете ждать, пока promise *metdataLoaded* будет разрешен, и, если в модели нет ожидающих изменений, вы создадите новую сущность документа покупки, вызвав метод *createEntry* Модели OData с именем набора сущностей *PurchaseDocument* в качестве параметра. Новый документ о покупке не сразу отправляется на серверную часть, а вместо этого запрос сохраняется в очереди запросов, чтобы мы могли позже вручную отправить его на серверную часть, когда закончим редактирование документа.

sap.ui.define([

"…/BaseController" ],

function(BaseController){

"usestrict";

returnBaseController.extend("…/CreatePurchaseDocument",

{

onInit:function(){

this.getRouter().getRoute("CreatePurchaseDocument").

attachMatched(this.\_createPurchaseDocument,this);

},

\_createPurchaseDocument:function(){

this.getModel().metadataLoaded().then(function(){

if (!this.getModel().hasPendingChanges()){

var oPurchaseDocumentBindingContext=

this.getModel().createEntry("/PurchaseDocument");

this.getView().bindElement(oPurchaseDocumentBindingContext.getPath());

}

}.bind(this));

}

});

});

Листинг 7.9. Создание новой записи документа покупки и привязка ее к Виду

Чтобы запустить приложение из панели «Файлы» в полнофункциональной среде SAP Web IDE, просто щелкните правой кнопкой мыши корневую папку приложения и выберите **Run - Run As - Web Application** в его контекстном меню. Во время выполнения представление «Создание документа покупки» будет похоже на экран, показанный на рис. 7.18.

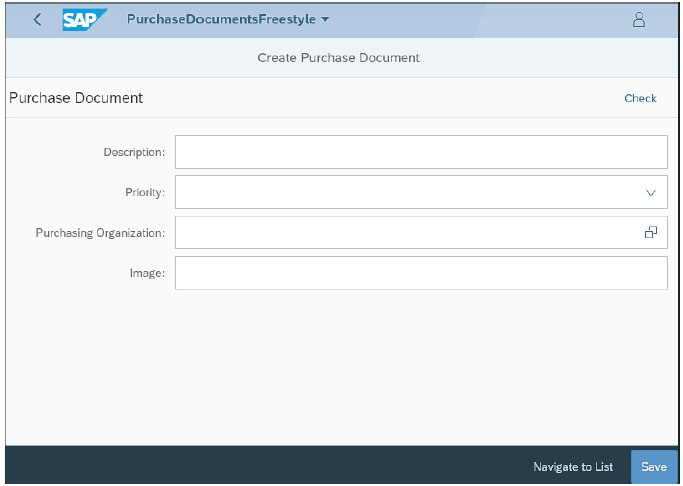


Рис. 7.18 Создание представления документа покупки во время выполнения

Наконец, чтобы отправить вновь созданный объект документа покупки в серверную часть при нажатии кнопки «Сохранить» и вызове функции обработчика события *onSave* внутри Контроллера, необходимо вызвать метод *submitChanges* объекта *sap.ui.model.odata.v2.ODataModel*. как показано в листинге 7.10. Его объект параметра содержит функцию обратного вызова успеха, которая вызывается в случае успеха, и функцию обратного вызова ошибки, которая вызывается в случае ошибки. Если отправка изменений в серверную часть прошла успешно, вы также отмените привязку объекта к представлению, чтобы можно было создать и привязать новый объект.

sap.ui.define([

"…BaseController" ],

function(BaseController){

"usestrict";

returnBaseController.extend("…CreatePurchaseDocument",

{

onSave:function(){

…

this.getModel().submitChanges({

success:function(){

this.getView().unbindElement();

}.bind(this),

error: function(){…}.bind(this)

});

…

}

});

});

Листинг 7.10. Отправка собранных изменений на серверную часть в методе onSave модуля CreatePurchaseDocument.controller.js

Прежде чем отправлять новый документ о покупке в серверную часть, вы должны сначала сравнить его значения с метаданными службы, чтобы обнаружить любые очевидные ошибки клиента, вызвав метод проверки *SmartForm*. Как показано в листинге 7.11, мы должны сначала получить экземпляр *SmartForm* по его идентификатору, а затем вызвать его метод проверки. Затем метод проверки возвращает массив полей с ошибками по отношению к метаданным OData полей, например, поле слишком длинное (*MaxLength*) или является пустым обязательным полем (*Nullable="false"*).

onSave:function(){

var oSmartForm=this.getView().byId("createPurchaseDocumentSmartForm");

var aSmartFieldsWithErrors=oSmartForm.check();

if (aSmartFieldsWithErrors.length!==0){

return;

}

…

}

Листинг 7.11. Вызов метода проверки SmartForm для проверки ошибок клиента перед отправкой изменений на сервер

После отправки вновь созданного документа о покупке в серверную часть вам нужно будет перейти к новому представлению, в котором будут отображаться все документы о покупке, созданные на данный момент. Чтобы отобразить и отфильтровать документы о покупке, вы будете использовать смарт-таблицу и панель смарт-фильтров. На данный момент давайте создадим новое пустой Вид с именем *DisplayPurchaseDocuments.view.xml*. Сначала вам нужно настроить навигацию между Видами *CreatePurchaseDocument.view.xml* и *DisplayPurchaseDocuments.view.xml* перед реализацией определения последнего Вида. Кроме того, вы создадите еще один Вид с именем *App.view.xml*, который будет служить корневым Видом приложения, как показано в листинге 7.12. Внутри Вида вы поместите только один элемент управления *sap.m.App* с ID *app* в качестве корневого контейнера для всех Видов.

Затем другие Виды будут динамически вставлены в агрегацию страниц экземпляра элемента управления *sap.m.App* во время выполнения. Элемент управления *sap.m.App* наследует свои возможности навигации от *sap.m.NavContainer* и дополнительно добавляет на HTML-страницу некоторые теги заголовков, которые считаются полезными для мобильных приложений.

<mvc:Viewxmlns="sap.m" xmlns:mvc="sap.ui.core.mvc" displayBlock="true">

<Appid="app" busy="{app>/isBusy}" busyIndicatorDelay="0"/>

</mvc:View>

Листинг 7.12. Корневой вид приложения

Маршруты навигации и цели приложения определяются в пространстве имен *sap.ui5* файла *manifest.json* дескриптора приложения. Однако первое, что вы должны сделать сейчас, это сделать только что созданный файл *App.view.xml* корневым Видом приложения. На рис. 7.19 показано, как *App.view.xml* можно определить как корневое представление приложения внутри файла дескриптора приложения *manifest.json*.

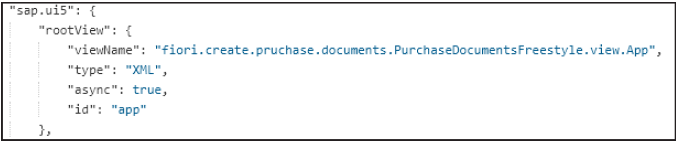


Рис. 7.19. Определение корневого представления приложения в файле дескриптора приложения manifest.json

Конфигурация навигации для нашего приложения показана на рис. 7.20. В объекте *config* конфигурации маршрутизации определите некоторую конфигурацию по умолчанию, относящуюся к маршрутам и целям, например, тип Вида или переход. Свойства *controlId* и *controllAggregation* относятся к *sap.m.routing.Targets* и определяют, что новые Виды, доступ к которым осуществляется с помощью навигации, будут вставлены в агрегацию страниц элемента управления контейнера *sap.m.App* внутри корневого представления (*App.view.xml*) с ID *app*.

Маршрут *CreatePurchaseDocument* сопоставляется с Видом *CreatePurchaseDocument* через цель *CreatePurchaseDocument*. Шаблон маршрута пуст, потому что это стартовая страница приложения. Маршрут *DisplayPurchaseDocuments* сопоставляется с Видом *DisplayPurchaseDocuments* через цель *DisplayPurchaseDocuments*.



Рис. 7.20 Сопоставление маршрутов с целевыми Видами в файле manifest.json дескриптора приложения

Вы можете напрямую перейти к Виду *DisplayPurchaseDocuments*, добавив идентификатор фрагмента *#/DisplayPurchaseDocuments* к URL-адресу приложения, поскольку его шаблон (*pattern*) маршрута — *DisplayPurchaseDocuments*.

Для программного перехода в Контроллерах от одного Вида к другому сначала необходимо получить экземпляр *sap.m.routing.Router*, определенный в файле *manifest.json* дескриптора приложения, что можно получить внутри Контроллера, как показано в листинге 7.13.

Мы определим новую вспомогательную функцию в базовом Контроллере, от которой наследуются все остальные Контроллеры, поскольку большинству Контроллеров потребуется эта функциональность.

sap.ui.define([

"sap/ui/core/mvc/Controller" ],

function(Controller){

"usestrict";

returnController.extend("…/BaseController",{

…

getRouter:function(){

return sap.ui.core.UIComponent.getRouterFor(this);

}

…

)};

)};

Листинг 7.13. Определение функции для получения автоматически созданного экземпляра sap.m.routing.Router через компонент sap.ui.core.UI.

Чтобы перейти от Вида создания к Виду отображения после того, как новый документ о покупке был отправлен в серверную часть, вы просто выполните код, показанный в листинге 7.14. Имя маршрута Вида, к которому вы хотите перейти (*DisplayPurchaseDocuments*), просто передается в качестве параметра функции *navTo* объекта *sap.m.routing.Router*.

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

Листинг 7.14. Переход от Вида «Создать» к Виду «Отображение»

Эта навигация запускается внутри успешного обратного вызова *submitChanges*, показанного ранее в листинге 7.10.

### 7.2.3 Страница List Report с использованием смарт-таблицы и панели фильтров

Пока что мы не определили содержимое *DisplayPurchaseDocuments.view.xml*, что является нашим следующим шагом. Как упоминалось ранее, наша цель — отобразить все доступные документы о покупке в таблице и включить фильтрацию. Таким образом, мы будем использовать смарт-таблицу для отображения документов о покупке и смарт-панель фильтров для фильтрации данных таблицы. На рис. 7.21 показано определение Вида, содержащее полосу смарт-фильтров и смарт-таблицу. Оба элемента управления привязаны к набору сущностей *PurchaseDocument*, а смарт-таблица и панель фильтров связаны с помощью идентификатора панели смарт-фильтров *PurchaseDocumentsSmartFilterBar*.



Рис. 7.21. Определение Вида DisplayPurchaseDocuments.view.xml, содержащего смарт-таблицу и панель фильтров

Если вы сейчас запустите приложение и создадите документ о покупке, вы автоматически будете перенаправлены в Вид «Показать документы о покупке», в котором будут отображаться все доступные документы о покупке в смарт-таблице, как показано на рис. 7.22. Чтобы создать элементы документа покупки, вы можете просто перейти на страницу созданного объекта (обсуждается в главе 4), щелкнув ссылку «Управление документами покупки». Со страницы объекта вы можете добавлять товары в документ покупки.

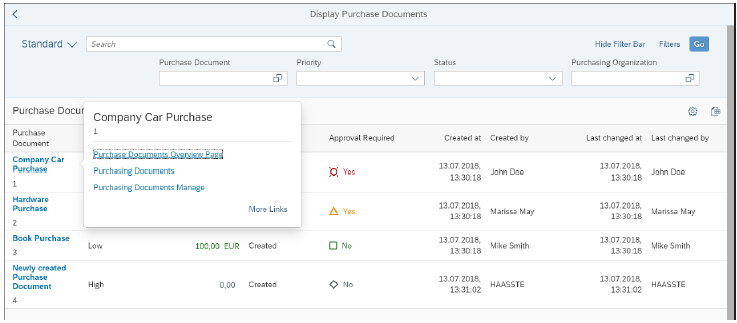


Рис. 7.22 Отображение документов покупки во время выполнения

### 7.2.4 Добавление одностраничного приемочного интеграционного теста

Чтобы протестировать и защитить создание и отображение документов о покупке с помощью приложения, включая навигацию между Видами создания и отображения, вы должны написать тест интеграции пользовательского интерфейса с помощью теста одностраничной приемки (OPA5) для платформы SAPUI5, который является частью SAPUI5 SDK. Эта структура предоставляет API для элементов управления SAPUI5 и упрощает тестирование взаимодействия с пользователем и навигации, скрывая асинхронность. OPA5 можно использовать с любой средой модульного тестирования JavaScript, но в нашем случае мы будем использовать QUnit в качестве базового средства запуска тестов и среды модульного тестирования. Вы можете легко настроить тестовую структуру по умолчанию для проекта приложения SAPUI5 в полном стеке SAP Web IDE, щелкнув правой кнопкой мыши корневую папку проекта и выбрав «**New - Test Structure**», как показано на рис. 7.23.

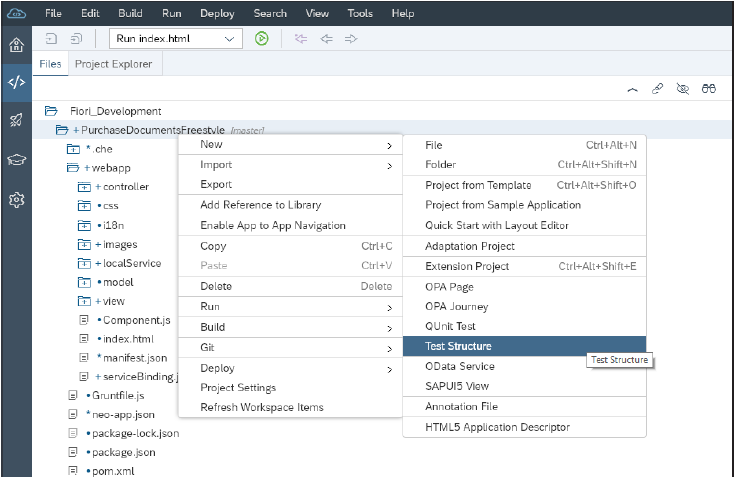


Рис. 7.23 Создание тестовой структуры для проекта приложения SAPUI5 в SAP Web IDE Full-Stack

Будет создана тестовая папка с вложенными папками модульного и интеграционного тестов. Подпапка модуля будет содержать скелеты тестов QUnit, а подпапка интеграции будет содержать скелеты тестов интеграции OPA5. Сгенерированная структура показана на рис. 7.24. Мы пока проигнорируем модульное тестирование и сосредоточимся на тестировании интеграции пользовательского интерфейса в следующем разделе. SAP Web IDE также сгенерировал путешествие OPA5 по умолчанию (*navigationJourney.js*), которое мы также проигнорируем и удалим, поскольку у нас свой путь.

Тем не менее, сгенерировать эту тестовую структуру довольно удобно; например, теперь у нас есть средство запуска тестов (*opaTests.qunit.html*) для запуска разрабатываемых нами путей (***journeys***) OPA5 и модуль, собирающий все наши пути (*AllJourneys.js*). Позже мы адаптируем файлы, содержащиеся в папке *pages*, к нашим конкретным потребностям. Тесты OPA5 организованы в виде путей и страниц. Путь содержит все тестовые примеры для конкретного варианта использования и структурирует свои тестовые примеры в соответствии с высокоуровневыми предложениями «дано-когда-тогда», скрывающими реальную реализацию тестового примера. Пути используют страницы, которые реализуют необходимые механизмы, действия и утверждения, используемые в высокоуровневом определении пути, и обычно связаны с одним Видом.

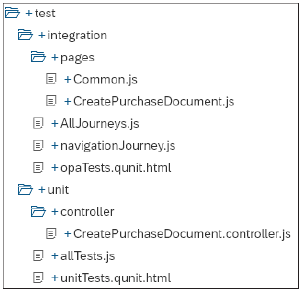


Рис. 7.24 Сгенерированная тестовая структура для проекта приложения SAPUI5

#### Настройка фиктивного (mock) сервера

При тестировании процесса создания, навигации и отображения документа о покупке в пользовательском интерфейсе следует использовать фиктивный сервер, поскольку вы не хотите зависеть от какой-либо серверной службы OData или доступных данных. Мок-сервер будет перехватывать запросы OData, которые приложение отправляет на серверную часть, и вместо этого возвращать локальные фиктивные данные.

Мок-сервер поддерживает запросы на создание, чтение, обновление и удаление (CRUD) и сохраняет изменения локально во время сеанса. Чтобы настроить фиктивный сервер, создайте модуль *mockserver.js* в папке *localService* проекта, в котором вы будете устанавливать и настраивать сервер. В листинге 7.15 показано, как можно создать экземпляр фиктивного сервера с корневым URI службы OData и смоделировать службу, указав документ метаданных локальной службы (*metadata.xml*) и базовую папку для фиктивных данных (*mockdata*). Установив для свойства ***bGenerateMissingMockData*** значение ***true***, фиктивный сервер будет автоматически генерировать фиктивные данные для сущностей, у которых нет выделенных фиктивных данных в папке фиктивных данных.

sap.ui.define([

"sap/ui/core/util/MockServer"

], function(MockServer){

"usestrict";

return{

init:function(){

var oMockServer=newMockServer({

rootUri:"/sap/opu/odata/sap/Z\_UI\_PURCHASEDOCUMENT\_V2/"

});

oMockServer.simulate("../localService/metadata.xml",{

sMockdataBaseUrl:"../localService/mockdata",

bGenerateMissingMockData:true

});

oMockServer.start();

}

};

});

Листинг 7.15. Создание экземпляра sap.ui.core.util.MockServer с корневым URI службы OData и имитация службы OData путем указания XML-файла метаданных и базовой папки фиктивных данных

#### Создать фиктивные данные

Вы можете легко создать правильные фиктивные данные для сущностей, щелкнув правой кнопкой мыши локальный файл *metadata.xml* в папке *localService* и выбрав «**Edit Mock Data**». Откроется диалоговое окно «Edit Mock Data», показанное на рис. 7.25, где вы выберете сущности на панели «Entity Sets» и создадите фиктивные данные, нажав кнопку «Add Row» в правом верхнем углу.

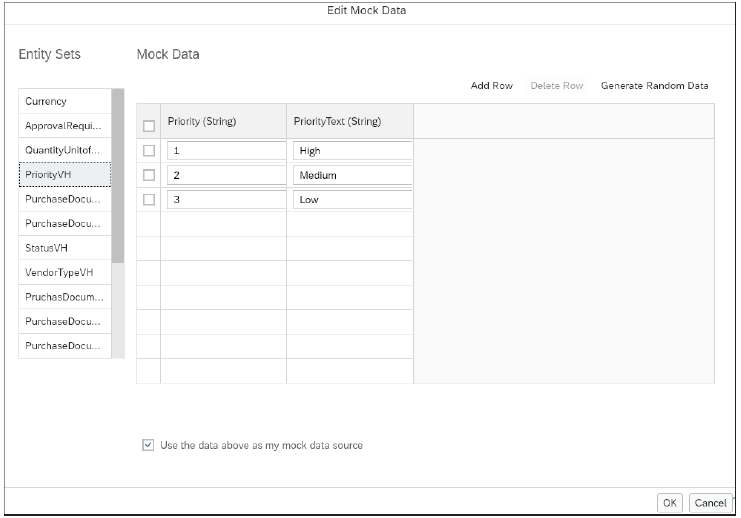


Рис. 7.25. Создание фиктивных данных для службы OData, определенной в документе метаданных локальной службы (metadata.xml)

После того, как вы нажмете OK, файлы JSON с фиктивными данными будут созданы в папке фиктивных данных, содержащей ранее определенные данные, как показано на рис. 7.26.

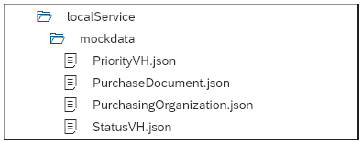


Рис. 7.26 Созданные JSON-файлы фиктивных данных

Поскольку вы используете смарт-элементы управления, приложение также сильно зависит от модели аннотаций пользовательского интерфейса для отображения элементов управления. Чтобы полностью изолировать приложение от серверной части, вы также должны имитировать службу каталогов, которая предоставляет модель аннотаций пользовательского интерфейса службы OData, как показано в листинге 7.16. Для фиктивного сервера аннотаций пользовательского интерфейса вы определите дополнительный запрос, соответствующий всем запросам GET, отправленным в службу, которая просто вернет локальный файл аннотаций *Z\_C\_PURCHASEDOCUMENTLRP\_CDS\_VAN.xml*.

…

var oAnnotationMockServer=newMockServer({

rootUri:"/sap/opu/odata/IWFND/CATALOGSERVICE;v=2/Annotations(TechnicalName=

'Z\_C\_PURCHASEDOCUMENTLRP\_CDS\_VAN',Version='0001')/$value/"

});

oAnnotationMockServer.setRequests([{

method: "GET",

path: newRegExp(".\*"),

response:function(oXhr){

var oLocalAnnotations=jQuery.sap.sjax({

url: sAnnotationURL,

datatype:"XML"

}).data;

oXhr.respondXML(200,{},jQuery.sap.serializeXML

(../localService/Z\_UI\_PURCHASEDOCUMENT\_V2\_VAN.xml));

return true;

}

}]);

oAnnotationMockServer.start();

…

Листинг 7.16. Создание локального фиктивного сервера для аннотаций пользовательского интерфейса

*Примечание*

*Чтобы обновить локальный файл метаданных в полном стеке SAP Web IDE, просто щелкните файл правой кнопкой мыши и выберите «Sync Metadata».*

#### Создание теста интеграции пользовательского интерфейса OPA5

Путь, который мы хотим реализовать и протестировать, показан в листинге 7.17 и должен выглядеть почти как описание тестового примера вручную или пользовательская история. Такой подход называется разработкой, управляемой поведением (***BDD***). Общая идея BDD заключается в том, что требования определяются в виде высокоуровневых предложений «дано-когда-тогда», а затем выполняются в виде автоматизированных тестов, что упрощает определение требований и тестовых сценариев нетехническими заинтересованными сторонами. В нашем пути мы проверим создание, сохранение и правильное отображение в смарт-таблице документа покупки после того, как произошел автоматический переход к Виду отображения.

sap.ui.define([

"sap/ui/test/opaQunit" ],

function(opaTest){

"usestrict";

QUnit.module("CreatePurchaseDocumentJourney");

opaTest("Should see the newly created PurchaseDocument in the SmartTable",

function(Given,When,Then){

Given.iStartTheApp();

var oPurchaseDocument={

Description:"OPA5TestDescription",

Priority:"1",

PurchasingOrganization:"ORG1",

PurchaseDocumentImageURL:"../images/book.jpg"

};

When.onTheCreatePurchaseDocumentPage

.iEnterPurchaseDocumentData(oPurchaseDocument)

.and

.iSavePurchaseDocument();

Then.onTheDisplayPurchaseDocumentsPage

.iCheckPurchaseDocumentIsInTable(oPurchaseDocument)

.and.iTeardownMyUIComponent();

});

});

Листинг 7.17. Путь OPA5 для создания и отображения документа покупки (createPurchaseDocumentJourney.js)

Объект *Given*, предоставляемый в качестве параметра функции *opaTest*, содержит расположение всех загруженных страниц, объект *When*, в свою очередь, содержит все действия, а объект *Then* содержит все утверждения. В нашем пути мы вызываем один метод размещения (*iStartTheApp*), который запускает компонент пользовательского интерфейса приложения (*Component.js*) и фиктивные серверы, определенные в модуле *mockserver.js*, вызывая его метод *init*. Схема реализована в модуле *Common.js*, который служит базовым классом для всех объектов страницы, которые мы будем реализовывать, как показано на рис. 7.27.



Рисунок 7.27 Базовый объект Common.js для страниц OPA5, содержащих расположение iStartApp

Действия *iEnterPurchaseDocumentData* и *iSavePurchaseDocument* реализованы на странице *CreatePurchaseDocument.js*. На рис. 7.28 показана часть реализации действия *iEnterPurchaseDocumentData*. Мы используем метод *waitFor* из *sap.ui.test.Opa5*, чтобы дождаться, пока элемент управления, который мы хотим выбрать, используя его представление и идентификатор *SmartField*, не будет обработан. Как только элемент управления выбран, мы вызываем предопределенное действие *sap.ui.test.actions.EnterText* для ввода текста в поля ввода, что позволит нам заполнить форму создания документа покупки.

После заполнения формы действие *iSavePurchaseDocument* будет вызвано из цикла *createPurchaseDocumentJourney.js*. В этом действии, показанном в листинге 7.18, просто нажмите кнопку «Сохранить», используя предопределенное действие *sap.ui.test.actions.Press*, как только кнопка будет найдена по ее идентификатору в представлении *CreatePurchaseDocument*.

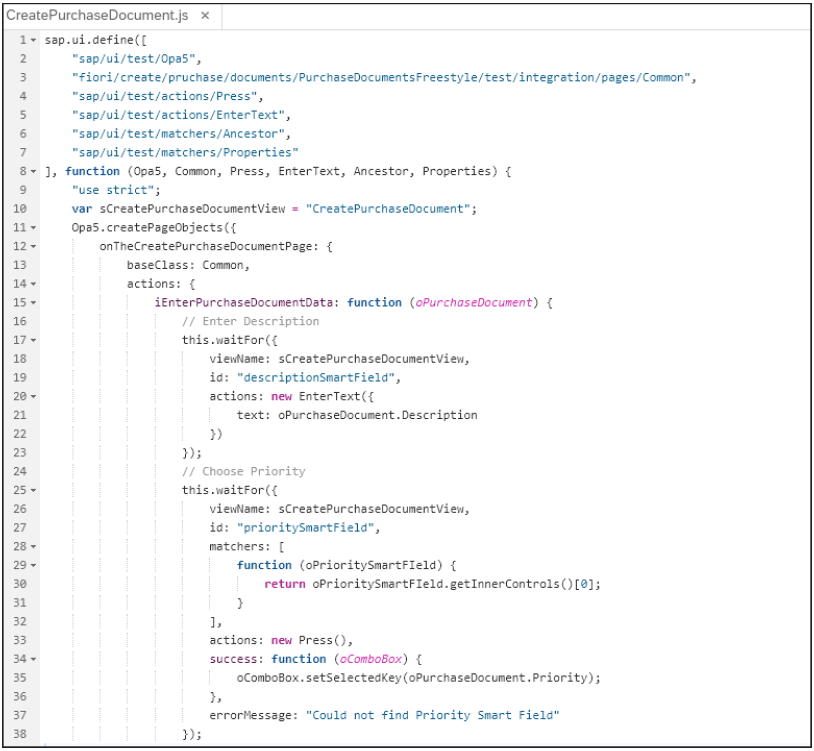


Рис. 7.28. Объект страницы OPA5 CreatePurchaseDocument.js, содержащий действие iEnterPurchaseDocumentData

iSavePurchaseDocument:function(){

// PressSaveButton

returnthis.waitFor({

viewName:sCreatePurchaseDocumentView,

id:"savePurchaseDocumentButton",

actions:newPress()

});

}

Листинг 7.18. Действие iSavePurchaseDocument также реализовано в объекте страницы OPA5 CreatePurchaseDocument.js

Для утверждения *iCheckPurchaseDocumentIsInTable*, проверяющего наличие вновь созданного документа покупки в смарт-таблице, мы создадим новую страницу с именем *DisplayPurchaseDocuments.js*, поскольку утверждение логически принадлежит *DisplayPurchaseDocuments.view.xml*. В утверждении, показанном на рис. 7.29, смарт-таблица сначала выбирается по ее представлению и идентификатору. Кроме того, реализованы два пользовательских сопоставителя для поиска вновь созданного документа покупки внутри смарт-таблицы.

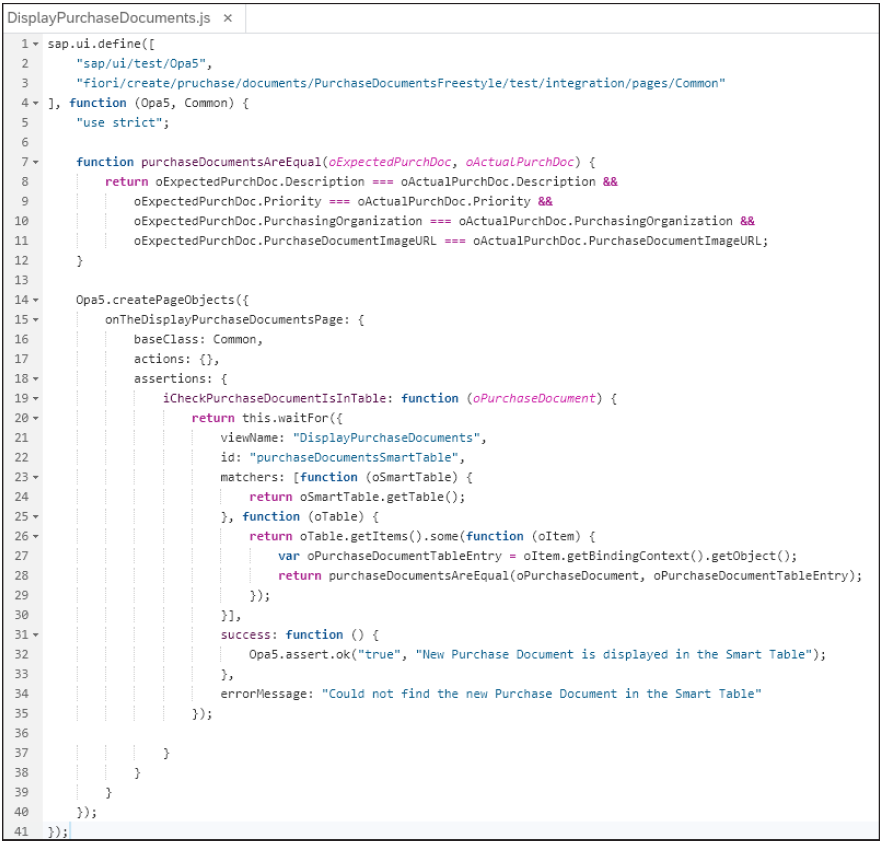


Рис. 7.29 Утверждение (iCheckPurchaseDocumentIsInTable) Проверка отображения вновь созданного документа покупки в смарт-таблице Вида DisplayPurchaseDocuments

Первая функция сопоставления извлекает таблицу, содержащуюся внутри смарт-таблицы; в нашем случае эта извлеченная таблица будет *sap.m.Table*. Вторая функция сопоставления извлекает элементы таблицы и перебирает их с помощью функции JavaScript *some*, которая проверяет, соответствует ли какой-либо элемент в массиве определенному условию. Мы вернем *true* в функции обратного вызова, если один элемент таблицы документа покупки равен ожидаемому документу покупки. Мы можем проверить равенство фактического и ожидаемого документа покупки, используя наивную функцию сравнения (*purchaseDocumentsAreEqual*), которая просто сравнивает свойства документа покупки, используемые для создания двух объектов документа покупки. Если второй сопоставитель возвращает *true*, будет вызвана функция успеха, и тест будет выполнен успешно. Если тайм-аут истекает из-за того, что документ о покупке не найден, будет отображаться свойство *errorMessage*, и тест завершится ошибкой.

Когда мы создали тестовую структуру для нашего проекта, полный стек SAP Web IDE создал файл модуля *AllJourneys.js* и файл *opaTests.qunit.html* для загрузки и выполнения интеграционных тестов OPA5. Чтобы запустить новые интеграционные тесты, мы должны загрузить наш путь (*createPurchaseDocumentJourney*) и соответствующие страницы (*CreatePurchaseDocument* и *DisplayPurchaseDocuments*) внутри файла модуля *AllJourneys.js*, показанного на рис. 7.30. Используя файл *opaTests.qunit.html*, мы можем затем напрямую выполнить путь, потому что оно уже загружает файл модуля *AllJourneys.js*. Запустите тесты, щелкнув правой кнопкой мыши файл opaTests.qunit.html и выбрав «**Run - Run As - Web Application**».



Рисунок 7.30. Модуль AllJourneys.js: центральный модуль для загрузки всех переходов и страниц OPA5, необходимых для тестирования приложения

Подобно модульным тестам QUnit, файл *opaTests.qunit.html* является средством запуска QUnit и выполняет все интеграционные тесты, определенные в файле модуля *AllJourneys.js*, как показано на рис. 7.31. Кроме того, этот файл отображает результат в удобном для чтения формате в виде веб-страницы.

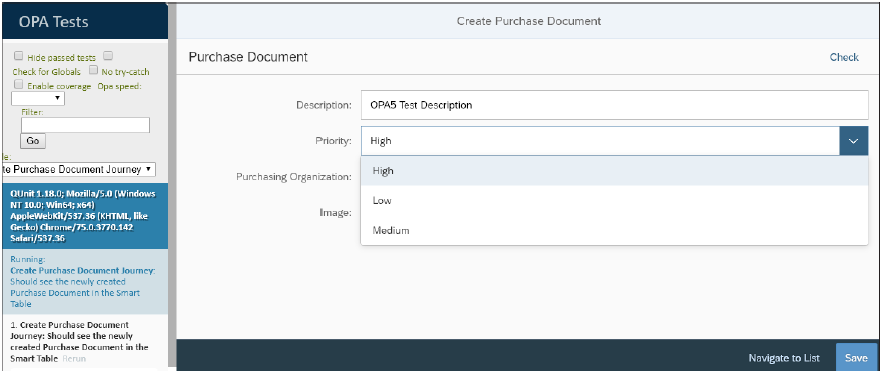


Рис. 7.31 Запуск теста OPA5 с использованием файла opaTests.qunit.html в качестве средства выполнения тестов QUnit

## 7.3 Резюме

В этой главе вы узнали о наиболее распространенных смарт-элементах управления: SmartField, SmartLink, SmartForm, смарт-таблице и смарт-панели фильтров. Мы обсудили аннотации, которые они анализируют, чтобы адаптировать их внешний вид, и то, как они значительно сокращают объем необходимого интерфейса SAPUI5 XML и кода JavaScript, сохраняя при этом гибкость с точки зрения общей компоновки и дизайна приложения. Мы продемонстрировали все смарт-элементы управления, описанные в этой главе, разработав небольшое приложение SAPUI5 для создания и отображения документов о покупке. Кроме того, вы узнали, как настроить проект SAPUI5 в свободном стиле в полном стеке SAP Web IDE, настроить источники данных и модели, настроить навигацию между представлениями и создать тест интеграции пользовательского интерфейса, включая фиктивный сервер для всего сценария создания и отображения, используя инфраструктуру OPA5, включенную в SAPUI5 SDK. В следующей главе вы узнаете, как развернуть все разработанные до сих пор приложения на панели запуска SAP Fiori.