openSAP Evolved Web Apps with SAPUI5

Week 2 Unit 4: Adding Views and Configuring Navigation

Exercises

PUBLIC

TABLE OF CONTENTS

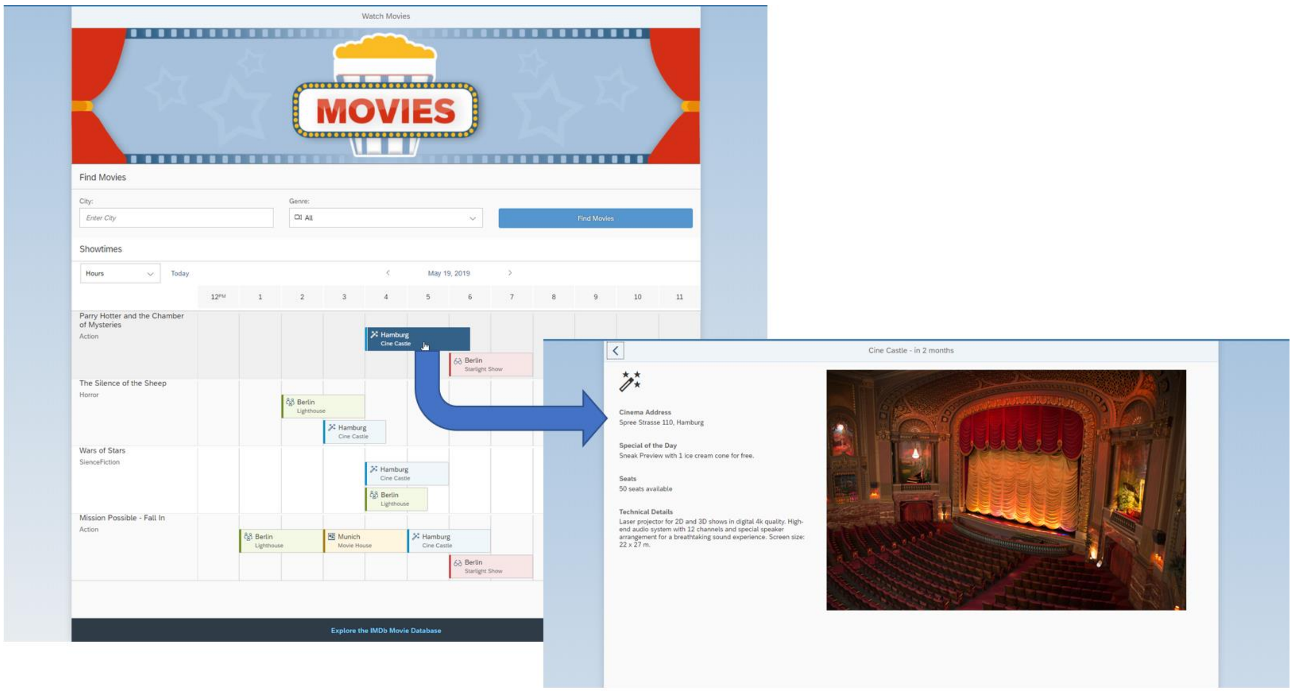
ADDING VIEWS AND CONFIGURING NAVIGATION

Summary

In this unit you will explore routing and navigation. When you have an application with more than one view, you need to define how and when to navigate to the other view(s) and how to handle the state of the application in the URL.
You'll start by adding another view to your app offering more details about the showtimes. A click on any showtime appointment should display the respective detail information:

- An icon to convey the cinema's corporate identity
- The cinema's address
- The number of seats
- Special information
- Technical information about the equipment
- A picture of the movie theater

Preview



Cinema Address
Spree Strasse 110, Hamburg

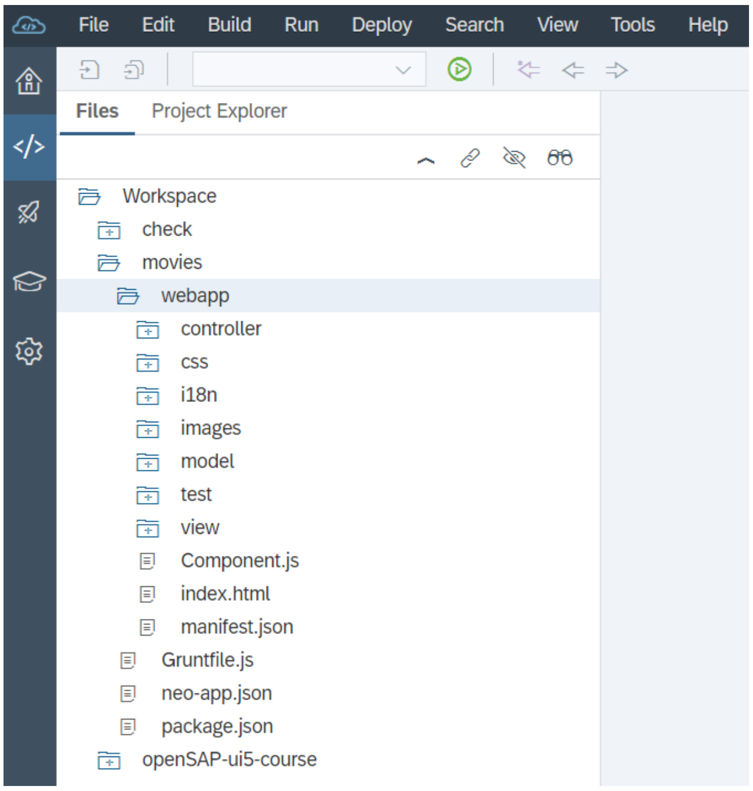
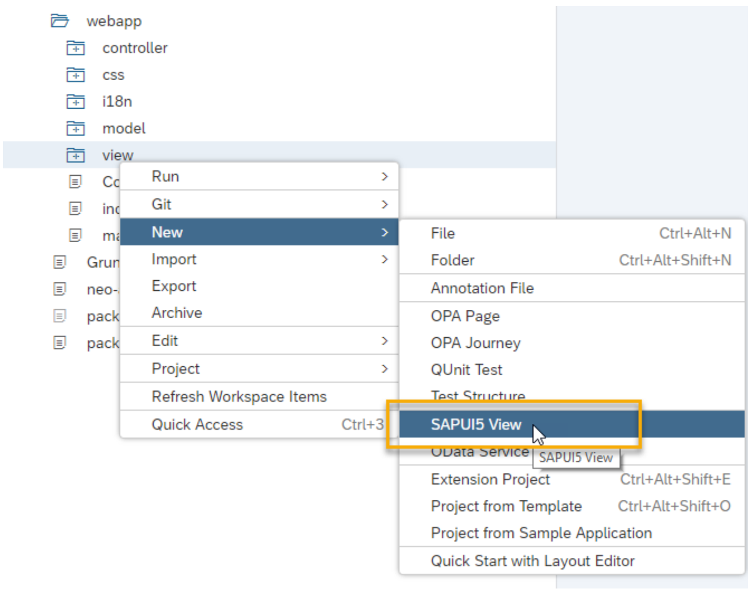
Special of the Day
Sneak Preview with 1 ice cream cone for free.

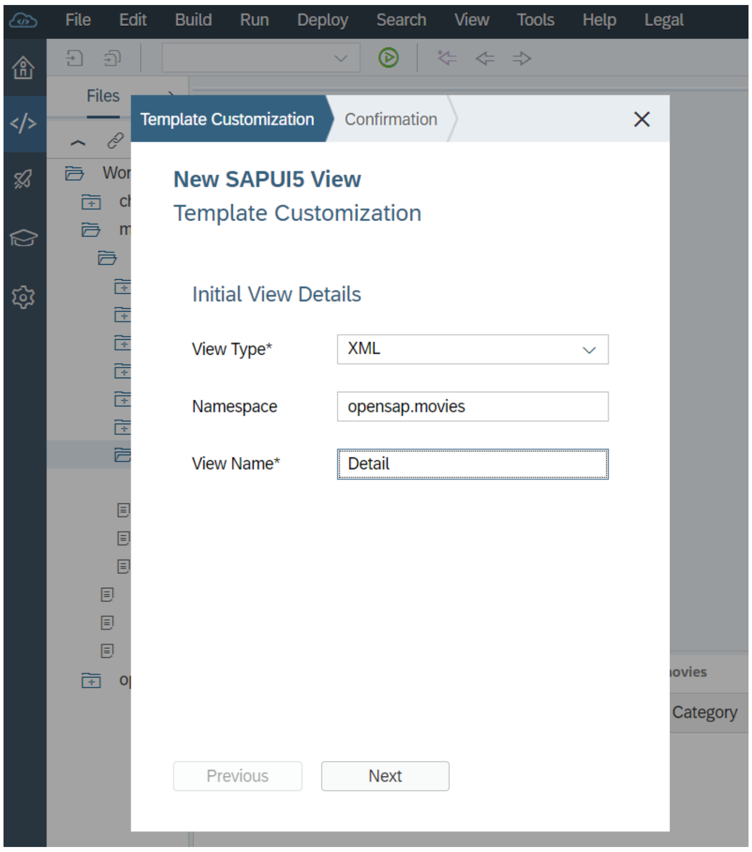
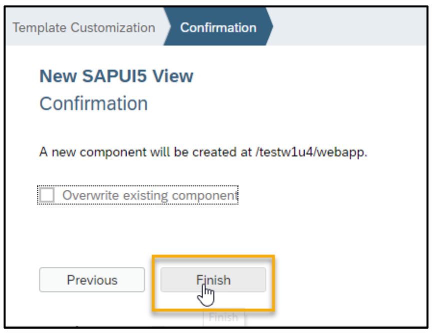
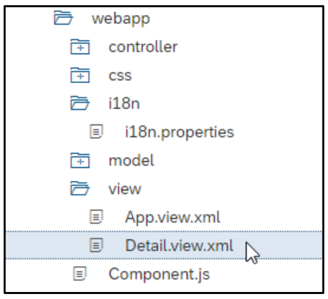
Seats
50 seats available

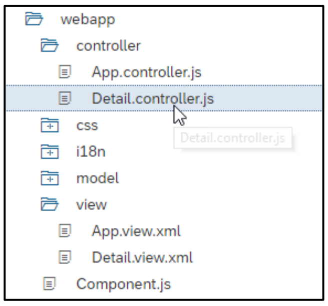
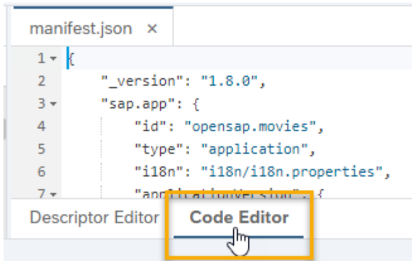
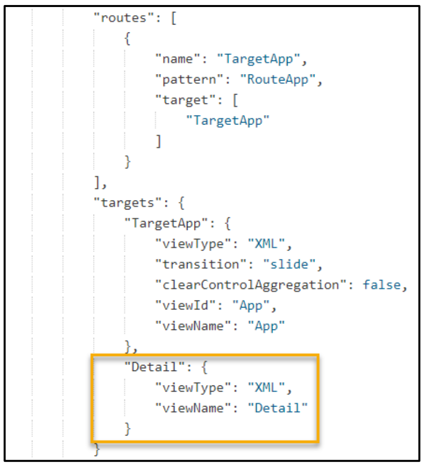
Technical Details
Laser projector for 2D and 3D shows in digital 4k quality. High-end audio system with 12 channels and special speaker arrangement for a breathtaking sound experience. Screen size: 22 x 27 m.

Figure 1 – Movies app with detail view and navigation

Create a new Detail view

Explanation	Screenshot
1. Go to your workspace on SAP Web IDE. Open the folder from the previous unit (week 2 unit 3) of this course (or download it from this course's GitHub repository (https://github.com/SAP/openSAP-ui5-course) and import it to your workspace).	 File Edit Build Run Deploy Search View Tools Help Files Project Explorer Workspace check movies webapp controller css i18n images model test view Component.js index.html manifest.json Gruntfile.js neo-app.json package.json openSAP-ui5-course
2. Right click the `view` folder, and choose *New → SAPUI5 View*.	 webapp controller css i18n model view Run Git New Import Export Archive Edit Project Refresh Workspace Items Quick Access Ctrl+3 File Ctrl+Alt+N Folder Ctrl+Alt+Shift+N Annotation File OPA Page OPA Journey QUnit Test Test Structure SAPUI5 View OData Service Extension Project Ctrl+Alt+Shift+E Project from Template Ctrl+Alt+Shift+O Project from Sample Application Quick Start with Layout Editor

Explanation	Screenshot
3. Enter **Detail** as the name for the new view, starting with a capital letter.	 File Edit Build Run Deploy Search View Tools Help Legal Files Template Customization Confirmation New SAPUI5 View Template Customization Initial View Details View Type* XML Namespace opensap.movies View Name* Detail Previous Next Category
4. Confirm the creation wizard message by clicking on *Finish*.	 New SAPUI5 View Confirmation A new component will be created at /testw1u4/webapp. Overwrite existing component
5. You can see that the new `Detail` view was inserted into the view folder.	 webapp controller css i18n i18n.properties model view App.view.xml Detail.view.xml Component.js

Explanation	**Screenshot**
6. In addition, a `Detail` controller was created in the controller folder.	
7. Now open the app descriptor `manifest.json`. It opens by default in the *Descriptor Editor* mode, so click on *Code Editor* at the bottom center. You will notice that even more coding was added. Your new `Detail` view was automatically added as a target for routing. **Note: What Are Targets?** Targets are typically referenced in a route and define which view should be displayed when a route was hit. In the routing configuration, you can even add multiple targets for the same route. All the views configured in the respective targets will be instantiated automatically.	 

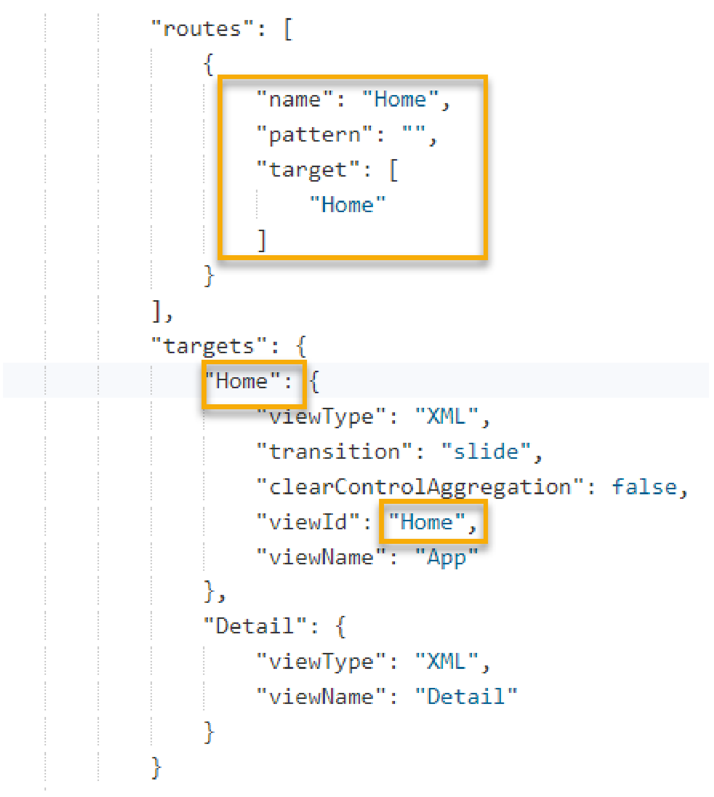
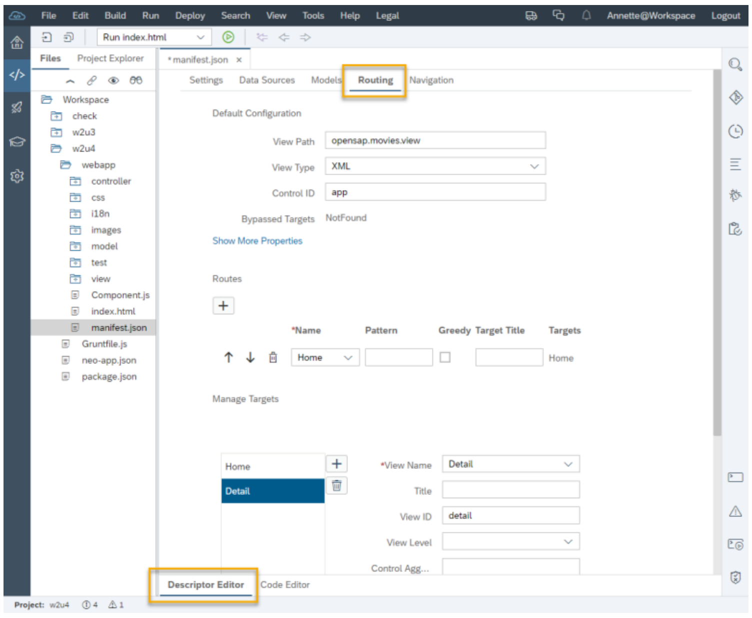
```
webapp
  controller
    App.controller.js
    Detail.controller.js
  css
  i18n
  model
  view
    App.view.xml
    Detail.view.xml
  Component.js
```

manifest.json ×

```
{
	"_version": "1.8.0",
	"sap.app": {
		"id": "opensap.movies",
		"type": "application",
		"i18n": "i18n/i18n.properties",
		"applicationVersion": {
```

Descriptor Editor   Code Editor

```
		"routes": [
			{
				"name": "TargetApp",
				"pattern": "RouteApp",
				"target": [
					"TargetApp"
				]
			}
		],
		"targets": {
			"TargetApp": {
				"viewType": "XML",
				"transition": "slide",
				"clearControlAggregation": false,
				"viewId": "App",
				"viewName": "App"
			},
			"Detail": {
				"viewType": "XML",
				"viewName": "Detail"
			}
		}
```

Explanation	Screenshot
8. To make your code more readable and understandable, change the following in the `manifest.json`: a) *name* → **Home** b) *pattern* → **""** c) *target* → **Home** d) *TargetApp* → **Home** e) *viewId* → **Home**	
9. SAP Web IDE offers you an UI tool to add or change your routing configurations. Open your `manifest.json` file, click on a) *Descriptor Editor* at the bottom left corner and b) *Routing* at the top.	

```
"routes": [
    {
        "name": "Home",
        "pattern": "",
        "target": [
            "Home"
        ]
    }
],
"targets": {
    "Home": {
        "viewType": "XML",
        "transition": "slide",
        "clearControlAggregation": false,
        "viewId": "Home",
        "viewName": "App"
    },
    "Detail": {
        "viewType": "XML",
        "viewName": "Detail"
    }
}
```

Default Configuration

View Path: opensap.movies.view
View Type: XML
Control ID: app
Bypassed Targets: NotFound

Show More Properties

Routes

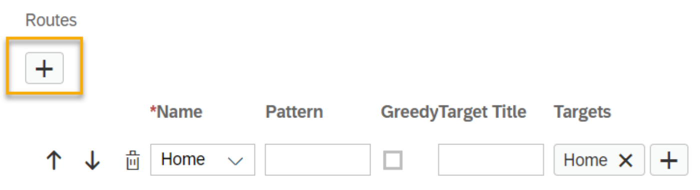
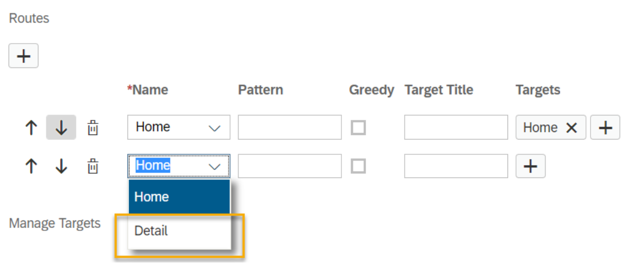
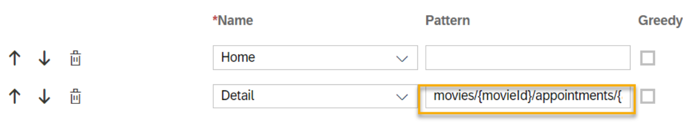
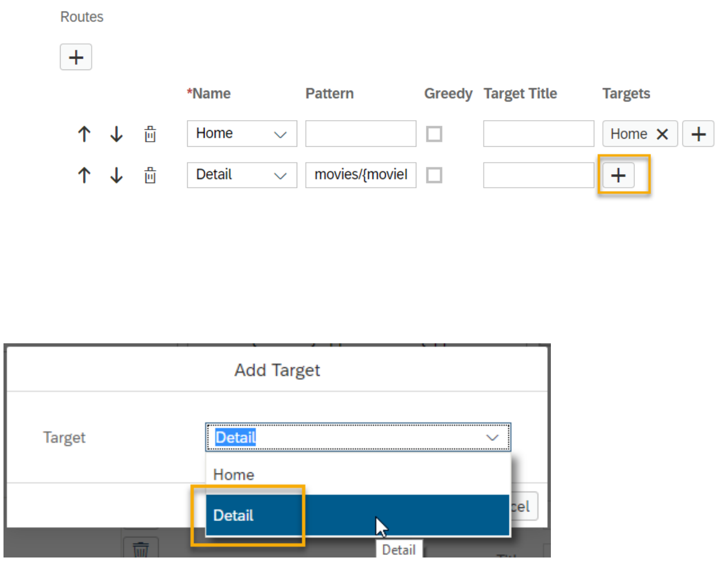
*Name	Pattern	Greedy	Target Title	Targets
Home		☐		Home

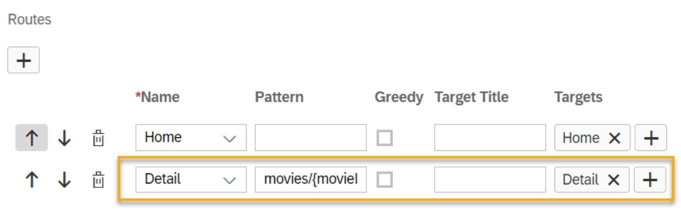
Manage Targets

Home
Detail

*View Name: Detail
Title:
View ID: detail
View Level:
Control Agg...

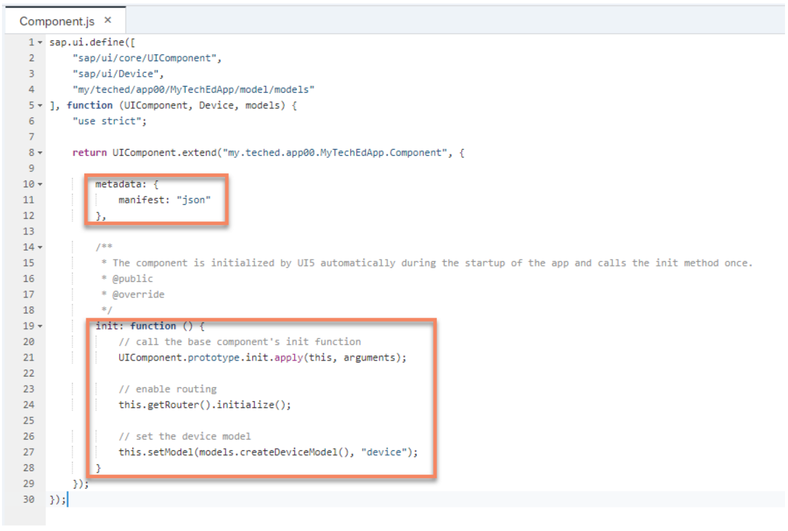
Create a new route for the new view

Explanation	**Screenshot**
1. You now add a new route to make use of the new, automatically created target. Click on the plus sign under *Routes* to add a new route. **Note: What Is a Route?** A route is a configuration using a target and a pattern and is a single route to a certain app view in a project. The routing configuration is responsible for loading and displaying the XML views of your app. You simply connect the views by triggering navigation events and let the router do the work.	 Routes / *Name / Pattern / GreedyTarget Title / Targets / Home / Home
2. Since the target already exists, you can just select it from the drop-down box.	 Routes / *Name / Pattern / Greedy / Target Title / Targets / Home / Home / Home / Home / Detail / Manage Targets
3. As *Pattern*, enter: **movies/{movieId}/appointments/{appointmentId}**	 *Name / Pattern / Greedy / Home / Detail / movies/{movieId}/appointments/{
4. Also, the target on the very right side of your new route can be selected from a popup dialog after clicking the plus icon. The target is needed as the end-point of the route that the pattern is applied to.	 Routes / *Name / Pattern / Greedy / Target Title / Targets / Home / Home / Detail / movies/{moviel / Add Target / Target / Detail / Home / Detail

Explanation	Screenshot
5. This is your final new route entry. **Note: Why do you need two routes?** You have two routes as one came with the initial SAPUI5 Application template. This route is still necessary to allow back navigation to the initial view.	 Routes | *Name | Pattern | Greedy | Target Title | Targets | | Home | | ☐ | | Home | | Detail | movies/{moviel | ☐ | | Detail |

Add code to trigger navigation

The last step to make your navigation work is the coding of the event handler. But let's take a step back for a moment and learn about a piece of configuration that came with the SAPUI5 Application template: the router initialization in the `Component.js` file. The `Component.js` is loaded first, and in it the `manifest.json` is called. At this point, your routing settings are loaded and are ready to be used.

Explanation	Screenshot
All UI assets are encapsulated in a component that is instantiated from your `index.html` page. Components are independent and reusable parts used in SAPUI5 applications. 1. Open your `Component.js` file and look at the implementation of the component. Note that the metadata is loaded from the manifest. 2. The `init` function configures additional models that are not defined in the manifest (e.g. the device model) so that it is created at runtime. The `init` function also initializes the router.	

Component.js ×

```
sap.ui.define([
	"sap/ui/core/UIComponent",
	"sap/ui/Device",
	"my/teched/app00/MyTechEdApp/model/models"
], function (UIComponent, Device, models) {
	"use strict";

	return UIComponent.extend("my.teched.app00.MyTechEdApp.Component", {

		metadata: {
			manifest: "json"
		},

		/**
		 * The component is initialized by UI5 automatically during the startup of the app and calls the init method once.
		 * @public
		 * @override
		 */
		init: function () {
			// call the base component's init function
			UIComponent.prototype.init.apply(this, arguments);

			// enable routing
			this.getRouter().initialize();

			// set the device model
			this.setModel(models.createDeviceModel(), "device");
		}
	});
});
```

webapp/view/App.view.xml

```
   ...
   </f:SimpleForm>
   <PlanningCalendar
      id="calendar"
      startDate="{path: 'movies>/initDate', formatter: '.formatter.formatDate'}"
      rows="{movies>/movies}"
      appointmentsVisualization="Filled"
      appointmentSelect=".onAppointmentSelect(${$parameters>/appointment})">
      <toolbarContent>
      ...
```

Add the `planningCalendar` event `appointmentSelect` and the name of the function it triggers to your XML code.

webapp/controller/App.controller.js

```
sap.ui.define([
   ...
   "sap/ui/model/FilterOperator",
   "sap/ui/core/UIComponent"
], function (Controller, Log, JSONModel, formatter, Filter, FilterOperator, UIComponent) {
   "use strict";
   ...

            oAppointmentsBinding.filter(oFilterCity);
         });
      },

      onAppointmentSelect: function (oAppointment) {
         var oContext = oAppointment.getBindingContext("movies"),
            sPath = oContext.getPath();

         var aParameters = sPath.split("/");
         UIComponent.getRouterFor(this).navTo("Detail", {
            movieId: aParameters[2],
            appointmentId: aParameters[4]
         });
      }

   });
});
```

Add the `sap/ui/core/UIComponent` dependency to the define statement of your `App` controller to be able to use it for your router.
Add the function `onAppointmentSelect` and pass `oAppointment` as a parameter into it. Then you define two variables, `oContext` for capturing the binding context, and `sPath` to capture the data binding information.
Now you need to extract the position of the selected movie and appointment from `sPath` – so you split it at each slash and save the result in an array. Then you call the route you defined previously in the app descriptor using `UIComponent` and provide the two mandatory parameters giving their respective position in the `aParameters` array.
In the `Detail` controller, you now have to receive those two parameters to display the correct information.

Enhance the Detail controller

webapp/controller/Detail.controller.js

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

   return Controller.extend("opensap.movies.controller.Detail", {

      /**
       * Called when a controller is instantiated and its View controls (if available) are
already created.
       * Can be used to modify the View before it is displayed, to bind event handlers and do
other one-time initialization.
       * @memberOf opensap.movies.view.Detail
       */
      onInit: function() {

   UIComponent.getRouterFor(this).getRoute("Detail").attachPatternMatched(this._onDetailMatched,
this);
      },

      _onDetailMatched : function (oEvent) {
         var oView = this.getView(),
            sMovieIndex = oEvent.getParameter("arguments")["movieId"],
            sAppointmentIndex = oEvent.getParameter("arguments")["appointmentId"];

         oView.bindElement({
            path: "/movies/" + sMovieIndex + "/appointments/" + sAppointmentIndex,
            model: "movies",
            events: {
               change : this._onBindingChange.bind(this)
            }
         });
      },

      _onBindingChange : function () {
         var oView = this.getView(),
            oElementBinding = oView.getElementBinding("movies"),
            sPath = oElementBinding.getPath();

         // if the path to the data does not exist we navigate to the not found page
         if (!oView.getModel("movies").getObject(sPath)) {
            //See Challenge at the end:
   UIComponent.getRouterFor(this).getTargets().display("NotFound");
         }
      }

   });
});
```

In the `Detail` controller, you register to the pattern matched event of the `Detail` route attach the `_onDetailMatched` call back function. Here, you fetch the route parameters from the arguments in the `oEvent` parameters by referring to the parameter names: `movieId` and `appointmentId`. Then, you bind the view to the model using the path to the exact appointment that the user clicked on. This will automatically display the chosen appointment details on the `Detail` view. The `_onBindingChange` function transfers the binding context to the `Detail` view.

Note: Download Cinema Images
Before you enhance the code of your Detail view, download the three cinema pictures we prepared for you from the course's GitHub repository. Import all three files `CinemaBerlin.png`, `CinemaHamburg.png`, `CinemaMunich.png` into the `images` folder of your app:
https://github.com/SAP/openSAP-ui5-course/tree/master/import

webapp/view/Detail.view.xml

```
<mvc:View
   controllerName="opensap.movies.controller.Detail"
   xmlns="sap.m"
   xmlns:mvc="sap.ui.core.mvc"
   xmlns:l="sap.ui.layout"
   xmlns:core="sap.ui.core">
   <App>
      <pages>
         <Page
            title="{movies>cinemaName} - {
               path: 'movies>startDate',
               type: 'sap.ui.model.type.Date',
               formatOptions: {
                  source: {
                     pattern: 'MM/dd/yyyy/hh:mm:ss'
                  },
                  relative: true,
                  relativeScale: 'auto'
               }
            }"
            showNavButton="true"
            class="sapUiResponsiveContentPadding"
            navButtonPress=".onNavBack">
            <content>
               <FlexBox wrap="Wrap">
                  <l:VerticalLayout
                     id="generalDetails"
                     class="sapUiMediumMarginEnd sapUiSmallMarginBottom">
                     <core:Icon
                        src="{movies>icon}"
                        size="3rem"
                        class="sapUiMediumMarginBottom"/>

                     <Label text="{i18n>cinemaAddress}" design="Bold"/>
                     <Text text="{movies>cinemaAddress}"
class="sapUiMediumMarginBottom"/>

                     <Label text="{i18n>special}" design="Bold"/>
                     <Text text="{movies>special}" class="sapUiMediumMarginBottom"/>

                     <Label text="{i18n>seats}" design="Bold"/>
                     <Text text="{movies>seats}" class="sapUiMediumMarc

                     <Label text="{i18n>technicalDetails}" design="Bolc
                     <Text text="{movies>technicalDetails}" width="400
                  </l:VerticalLayout>
                  <Image
                     width="100%"
                     src="{movies>pic}"/>
               </FlexBox>
            </content>
         </Page>
      </pages>
   </App>
</mvc:View>
```

Download pictures from `import` folder in the course's GitHub repository: https://github.com/SAP/openSAP-ui5-course

First, you provide a page title. This contains the data binding syntax to display the cinema name that corresponds to the appointment the user clicked on.

In addition, the title displays a relative date as a literal to indicate the user when the event is coming up, like the word(s) "tomorrow", or "in two weeks". The conversion is done by giving the date the `sap.ui.model.type.Date`, `formatOptions` type as the input format and the `relative:true` property for the output.

Then, you add the `navButton` property and set it to `true`. Add the `navButtonPress` event that gets triggered when the button is hit, and the `onNavBack` function that is then called. You create this function in the `Detail` controller.

You apply the `sapUiResponsiveContentPadding` class to the page as it is the container for all your detail controls. This is an example for the predefined CSS classes UI5 offers. There are classes for paddings and margins and they allow programmers to position and layout controls on a page.

The `FlexBox` is a useful container. You can set the property wrap to `Wrap`, which allows the view to be fully responsive (=adjusts with the size of the page when it is re-sized).

webapp/view/Detail.controller.js

```
            ...
            //See Challenge at the end:
UIComponent.getRouterFor(this).getTargets().display("NotFound");
         }
      },

      onNavBack : function () {
         UIComponent.getRouterFor(this).navTo("Home");
      }

   });

});
```

Implement the `onNavBack` function that is called by the `navButtonPress` event by having the router call your `Home` target. Save your change.

Add localized texts to the resource bundle

webapp/i18n/i18n.properties

```
...
messageToast=Do you feel like going to the movies?

#~~~ Detail View ~~~~~~~~~~~~~~~~~~~~~~~~~~

cinemaName=Cinema Name
cinemaAddress=Cinema Address
special=Special of the Day
technicalDetails=Technical Details
seats=Seats
```

It’s good practice to keep any texts in a central resource bundle for easy translation, so you add all the static texts of your new `Detail` view to the i18n properties.
Add a heading `Detail View` for your entries to indicate what view the texts belong to. This way your file will stay neat and readable, even if you add more views. Then you add the key-value pairs for your new entries.

Improve Performance

It is good practice to create an empty app view and let the routing load and place all views inside the app. This way, the views are only loaded when the corresponding route has been hit. We therefore move the initial page from the app view to a separate Home view.

webapp/view/Home.view.xml (New)

```
<mvc:View
   controllerName="opensap.movies.controller.App"
   xmlns="sap.m"
   xmlns:mvc="sap.ui.core.mvc"
   xmlns:core="sap.ui.core"
   xmlns:f="sap.ui.layout.form"
   xmlns:unified="sap.ui.unified">
   <Page title="{i18n>title}">
      <content>
         <Image
            visible="{= !${device>/system/phone} }"
            src="images/MoviesHeader.png"
            width="100%"
            tooltip="{i18n>imageTooltip}"
            press="sap.m.MessageToast.show(${i18n>messageToast})"/>
         <f:SimpleForm
            id="form"
            editable="true"
            layout="ColumnLayout"
            title="{i18n>titleForm}"
            columnsM="2"
            columnsL="3"
            columnsXL="3">
            <f:content>
               <Label
                  text="{i18n>labelCity}"
                  labelFor="city"/>
               <SearchField
                  id="city"
                  width="100%"
                  showSearchButton="false"
                  placeholder="{i18n>cityPlaceholder}"/>
               <Label
                  text="{i18n>labelGenre}"
                  labelFor="genre"/>
               <Select
                  id="genre"
                  width="100%">
                  <core:ListItem icon="sap-icon://video" key=""
text="{i18n>genreAll}"/>
                  <core:ListItem icon="sap-icon://physical-activity" key="Action"
text="{i18n>genreAction}"/>
                  <core:ListItem icon="sap-icon://electrocardiogram" key="Horror"
text="{i18n>genreHorror}"/>
                  <core:ListItem icon="sap-icon://paper-plane" key="SienceFiction"
text="{i18n>genreScienceFiction}"/>
               </Select>
               <Label/>
               <Button
                  type="Emphasized"
                  text="{i18n>buttonMovieSearch}"
                  class="sapUiSmallMarginTop"
                  press=".onPress('for movies!')"/>
            </f:content>
         </f:SimpleForm>
```

```
          <PlanningCalendar
             id="calendar"
             startDate="{path: 'movies>/initDate', formatter:
'.formatter.formatDate'}"
             rows="{movies>/movies}"
             appointmentsVisualization="Filled"
             appointmentSelect=".onAppointmentSelect(${$parameters>/appointment})">
             <toolbarContent>
                <Title text="{i18n>calendarTitle}" titleStyle="H4"/>
             </toolbarContent>
             <rows>
                <PlanningCalendarRow
                   title="{movies>name}"
                   text="{movies>genre}"
                   appointments="{path : 'movies>appointments', templateShareable:
'true'}">
                   <appointments>
                      <unified:CalendarAppointment
                         startDate="{path: 'movies>startDate', formatter:
'.formatter.formatDate'}"
                         endDate="{path: 'movies>endDate', formatter:
'.formatter.formatDate'}"
                         title="{movies>info}"
                         text="{movies>cinemaName}"
                         icon="{movies>icon}"
                         type="{movies>type}">
                      </unified:CalendarAppointment>
                   </appointments>
                </PlanningCalendarRow>
             </rows>
          </PlanningCalendar>
       </content>
       <footer>
          <Toolbar>
             <ToolbarSpacer/>
             <Link emphasized="true" target="_blank" href="https://www.imdb.com/"
text="{i18n>footerLink}"/>
             <ToolbarSpacer/>
          </Toolbar>
       </footer>
   </Page>
</mvc:View>
```

Create a new file `Home.view.xml` and move the page from the `App.view.xml` to it.

webapp/view/App.view.xml

```
<mvc:View
   controllerName="opensap.movies.controller.App"
   displayBlock="true"
   xmlns="sap.m"
   xmlns:mvc="sap.ui.core.mvc"
   xmlns:core="sap.ui.core"
   xmlns:f="sap.ui.layout.form"
   xmlns:unified="sap.ui.unified">
   <Shell>
      <App id="app">
         <pages>
            <Page title="{i18n>title}">
               <content>
                  <Image
                     visible="{= !${device>/system/phone} }"
```

```
                   src="images/MoviesHeader.png"
                   width="100%"
                   tooltip="{i18n>imageTooltip}"
                   press="sap.m.MessageToast.show(${i18n>messageToast})"/>
                <f:SimpleForm
                   id="form"
                   editable="true"
                   layout="ColumnLayout"
                   title="{i18n>titleForm}"
                   columnsM="2"
                   columnsL="3"
                   columnsXL="3">
                   <f:content>
                      <Label
                         text="{i18n>labelCity}"
                         labelFor="city"/>
                      <SearchField
                         id="city"
                         width="100%"
                         showSearchButton="false"
                         placeholder="{i18n>cityPlaceholder}"/>
                      <Label
                         text="{i18n>labelGenre}"
                         labelFor="genre"/>
                      <Select
                         id="genre"
                         width="100%">
                         <core:ListItem icon="sap-icon://video" key=""
text="{i18n>genreAll}"/>
                         <core:ListItem icon="sap-icon://physical-activity"
key="Action" text="{i18n>genreAction}"/>
                         <core:ListItem icon="sap-icon://electrocardiogram"
key="Horror" text="{i18n>genreHorror}"/>
                         <core:ListItem icon="sap-icon://paper-plane"
key="SienceFiction" text="{i18n>genreScienceFiction}"/>
                      </Select>
                      <Label/>
                      <Button
                         type="Emphasized"
                         text="{i18n>buttonMovieSearch}"
                         class="sapUiSmallMarginTop"
                         press=".onPress('for movies!')"/>
                   </f:content>
                </f:SimpleForm>
                <PlanningCalendar
                   id="calendar"
                   startDate="{path: 'movies>/initDate', formatter:
'.formatter.formatDate'}"
                   rows="{movies>/movies}"
                   appointmentsVisualization="Filled">
                   <toolbarContent>
                      <Title text="{i18n>calendarTitle}" titleStyle="H4"/>
                   </toolbarContent>
                   <rows>
                      <PlanningCalendarRow
                         title="{movies>name}"
                         text="{movies>genre}"
                         appointments="{path : 'movies>appointments',
templateShareable: 'true'}">
                         <appointments>
                            <unified:CalendarAppointment
                               startDate="{path: 'movies>startDate', formatter:
'.formatter.formatDate'}"
```

```
                                 endDate="{path: 'movies>endDate', formatter:
'.formatter.formatDate'}"
                                 title="{movies>info}"
                                 text="{movies>cinemaName}"
                                 icon="{movies>icon}"
                                 type="{movies>type}">
                              </unified:CalendarAppointment>
                           </appointments>
                        </PlanningCalendarRow>
                     </rows>
                  </PlanningCalendar>
               </content>
               <footer>
                  <Toolbar>
                     <ToolbarSpacer/>
                     <Link emphasized="true" target="_blank"
href="https://www.imdb.com/" text="{i18n>footerLink}"/>
                     <ToolbarSpacer/>
                  </Toolbar>
               </footer>
            </Page>
         </pages>
      </App>
   </Shell>
</mvc:View>
```

Delete the respective code from the `App.view.xml` so that only the `Shell` and the `App` control remain. As a result, the empty App view is initiated once initially (declared as the property `rootView` in the manifest) and the Home view is only loaded when the route “Home” (the default route of the app) is hit. When a deep link to the detail page is shared, only the detail page is loaded but not the home page.

webapp/manifest.json

```
...
"targets": {
            "Home": {
               "viewId": "home",
               "viewName": "Home"
            },
            ...
```

Set the Home view as target in the app descriptor (manifest.js) so that it can be used for routing.

webapp/view/Detail.view.xml

```
   ...
   xmlns:l="sap.ui.layout"
   xmlns:core="sap.ui.core">
   <App>
      <pages>
         <Page
            title="{movies>cinemaName} - {
               ...

            </FlexBox>
            </content>
         </Page>
      </pages>
   </App>
</mvc:View>
```

The detail view is now placed into the app view by the router whenever the corresponding route is hit, so we can safely remove the `App` control from the detail view. This code came with the Web IDE SAPUI5-view-creation wizard which assumes that this view is used independently.

★ CHALLENGE YOURSELF: ADD A "NOT FOUND" PAGE

This task does not come with a predefined solution and can be solved creatively – dive a bit deeper into the topics and exchange with other learners to make the most out of your learning experience. Good luck!

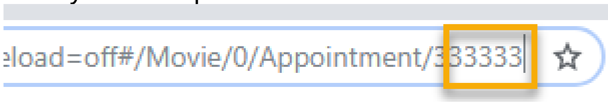
Summary

The users of your app often share a link to a movie with their friends. As the showtimes for movies frequently change, such links get outdated fast. In such cases and when a completely wrong URL is entered, an error message should be shown in the form of a "not found" page. The user should also be able to navigate back to the home screen from there to select another movie.

View: NotFound.view.xml
Title: Bummer!
Icon: "sap-icon://video"
Description: This movie has not been made yet
Conditions: Shown for invalid URL patterns and outdated movie URLs

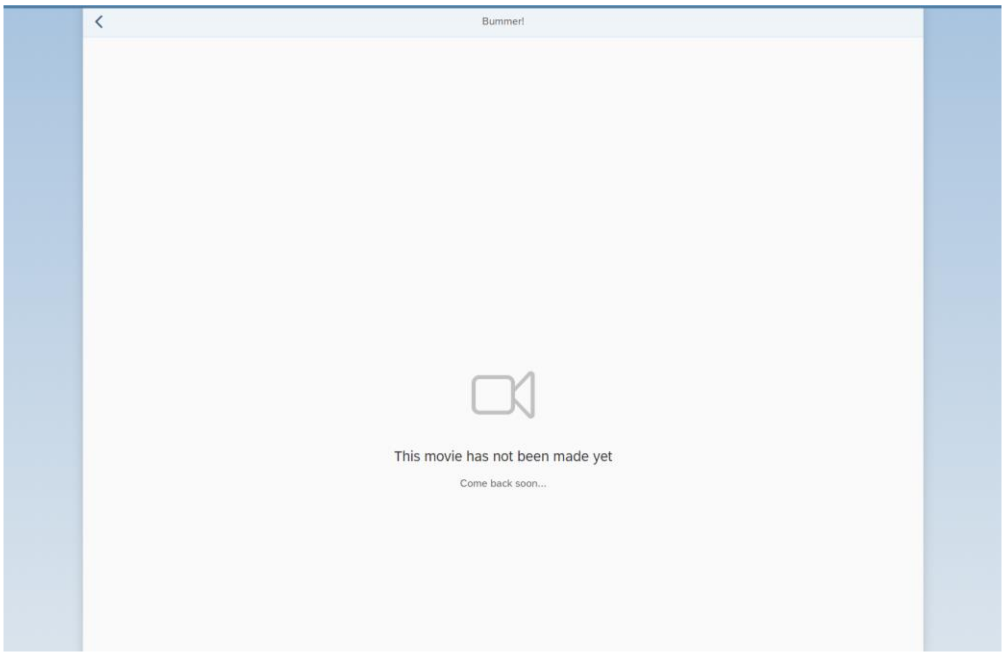
How to Test

Modify the data path in the URL to an invalid movie:



```
eload=off#/Movie/0/Appointment/333333
```

Preview



Bummer!

This movie has not been made yet

Come back soon...

Figure 2 - The "not found" page is shown when a movie does not exist (yet 😉)

Hints:

- There is a `bypassed` event in the routing configuration
- A comprehensive tutorial about routing and navigation comes in handy
- The `sap.m.MessagePage` is a simplified page to display messages

RELATED MATERIAL

- Demo Kit: Tutorial Routing and Navigation
- Demo Kit: Using Predefined CSS Margin Classes

www.sap.com/contactsap

THE BEST RUN