openSAP Evolved Web Apps with SAPUI5
Week 2 Unit 3: Binding Data and Using Models

Exercises

PUBLIC

TABLE OF CONTENTS

BINDING DATA AND USING MODELS

Summary

In this unit, you will use data binding to display movie showtimes at various cinemas in a calendar. In addition, you will implement the search function to plan your evening by city and genre. You will also learn how to prepare your app for internationalization and move all hard-coded texts from the previous units to the internationalization properties.

Preview

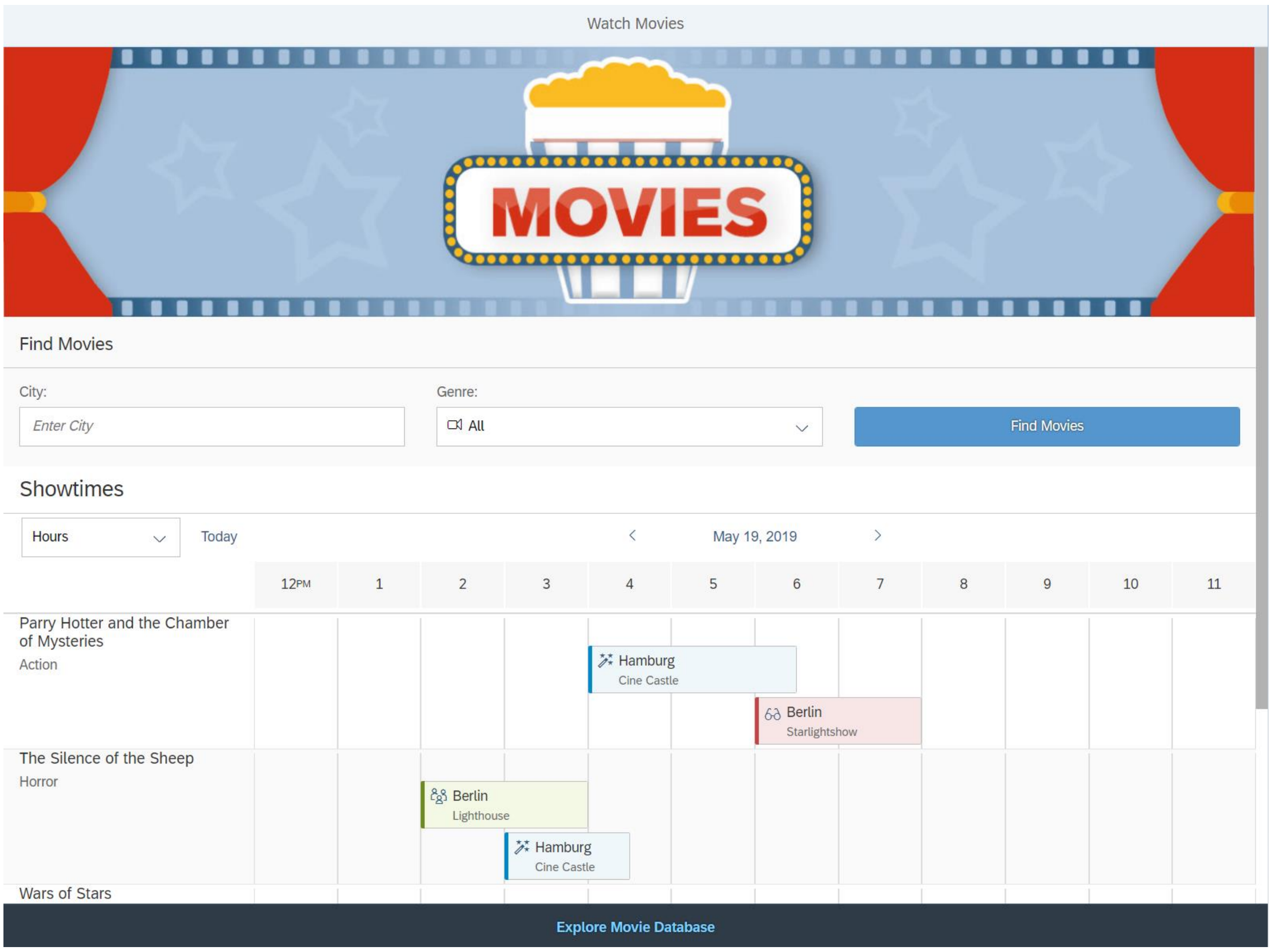


Figure 1 – Movie app with calendar allowing to search for a city and a genre

Add the planning calendar to the app

webapp/App.view.xml

```
<mvc:View
   controllerName="opensap.movies.controller.App"
   displayBlock="true"
   xmlns="sap.m"
   xmlns:mvc="sap.ui.core.mvc"
   xmlns:f="sap.ui.layout.form"
   xmlns:unified="sap.ui.unified">
   <Shell>

…
</f:SimpleForm>
<PlanningCalendar
   id="calendar"
   startDate=""
   rows=""
   appointmentsVisualization="Filled">
   <toolbarContent>
      <Title text="" titleStyle="H3"/>
   </toolbarContent>
   <rows>
      <PlanningCalendarRow
         title=""
         text=""
         appointments="">
         <appointments>
            <unified:CalendarAppointment
               startDate=""
               endDate=""
               title=""
               text=""
                                       icon=""
               type="">
            </unified:CalendarAppointment>
         </appointments>
      </PlanningCalendarRow>
   </rows>
</PlanningCalendar>
</content>
…
```

Underneath the `SimpleForm`, which contains the controls from the previous unit, add the XML code for the `PlanningCalendar`. Add the `rows` and the `appointments` to the aggregation. These will serve as the template for the data binding later.

The `PlanningCalendarRow` shows:

- movie name (`title`) and
- genre (`text`)

The `CalendarAppointment` shows:

- cinema name (`title`)
- city info (`text`)
- icon (`icon`)
- date and time of the show (`startDate` and `endDate`)

Don't forget to define the `unified` namespace at the very top.
For now, leave the properties empty.

Add a new JSON model for movies and appointments

webapp/model/Movies.json (NEW)

```
{
   "initDate": "05/19/2019/12:0:0",
   "movies": [{
      "name": "Parry Hotter and the Chamber of Mysteries",
      "genre": "Action",
      "appointments": [
         {
            "startDate": "05/19/2019/16:0:0",
            "endDate": "05/19/2019/18:30:0",
            "cinemaName": "Cine Castle",
            "cinemaAddress": "Spree Strasse 110, Hamburg",
            "special": "Sneak Preview with 1 ice cream cone for free.",
            "icon": "sap-icon://activate",
            "seats": "50 seats available",
            "technicalDetails":"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.",
            "info": "Hamburg",
            "type": "Type06",
            "pic": "images/CinemaHamburg.png"
         },
…
```

Download entire file from the `import` folder in the course's GitHub repository: https://raw.githubuserco ntent.com/SAP/openSA

We need some movie data in our project, so download the `Movies.json` file from the GitHub repository at https://raw.githubusercontent.com/SAP/openSAP-ui5-course/master/import/Movies.json, then go to the `import` folder. After you saved it to your local computer, go back to your SAP Web IDE workspace, right-click on the `model` folder, and choose *Import* → *File or Project*. Then select the `Movies.json` file on your local computer. Afterwards, open the file and make sure that it contains a list of movies.

webapp/manifest.json

```
…
   "sap.ui5": {
      "models": {
         "i18n": {
            "type": "sap.ui.model.resource.ResourceModel",
            "settings": {
               "bundleName": "movieapp.MovieApp.i18n.i18n"
            }
         },
         "movies": {
            "type": "sap.ui.model.json.JSONModel",
            "uri": "model/Movies.json"
         }
      },
      "resources": {
         "css": [{
            "uri": "css/style.css"
            …
```

In this step we are adding a `JSONModel` to the `models` section in our application descriptor.
It will be created automatically at application start and is available under the configured name `movies`.

Convert date and time format for the PlanningCalendar

The `PlanningCalendar` control needs the date and the time as a JavaScript `Date` object. We'll use a method in the formatter file to convert the raw data.

webapp/model/formatter.js (NEW)

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

   return {
      formatDate: function (sValue) {
         if (!sValue) {
            return null;
         }

         return new Date(sValue);
      }
   };

});
```

Go to the *model* folder and create a new file `formatter.js`. The function receives a string value as parameter into `sValue` (the leading "s" indicates a string data type) and converts it via the JavaScript function `Date` to the JavaScript `Date` object.

webapp/App.controller.js

```
sap.ui.define([
       "sap/ui/core/mvc/Controller",
       "sap/base/Log",
   "../model/formatter"

], function (Controller, Log, formatter) {
   "use strict";

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

      formatter: formatter,

      onInit: function () {

…
```

Add the formatter as dependency to the `sap.ui.define` statement of the app controller and instantiate the formatter in the app controller so that the controller can handle the call for our formatter method.

Bind the movie showtimes to the planning calendar

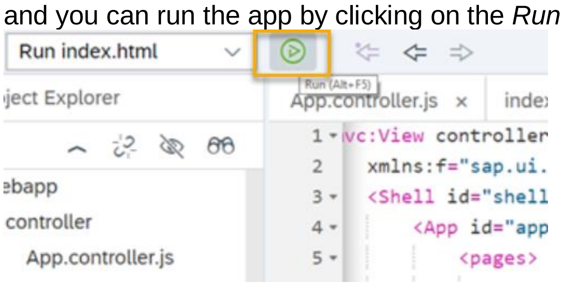
webapp/App.view.xml

```
...
<PlanningCalendar
   id="calendar"
   startDate="{path: 'movies>/initDate', formatter: '.formatter.formatDate'}"
   rows="{movies>/movies}"
   appointmentsVisualization="Filled">
   <toolbarContent>
      <Title text="Showtimes" titleStyle="H3"/>
   </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>
...
```

The data binding syntax allows you to make the following assignments:

- The curly brackets indicate the beginning and end of binding info to the browser.
- `movies` is the data model name we set in your manifest file.
- The close-angular-bracket character “>” separates the model name from the binding path to a particular entry.
- The `path` property of the data binding info is necessary when two properties are listed inside the data binding info. This is called “complex binding syntax”. You apply the `formatter` and set the `templateSharable` property.
- The `formatter` syntax points to your `formatDate` formatter method in your model folder.

Now, all properties of the `PlanningCalendar` are connected to the respective data from your data model, and you can run the app by clicking on the *Run* button:



```
vc:View controller
  xmlns:f="sap.ui.
  <Shell id="shell
     <App id="app
        <pages>
```

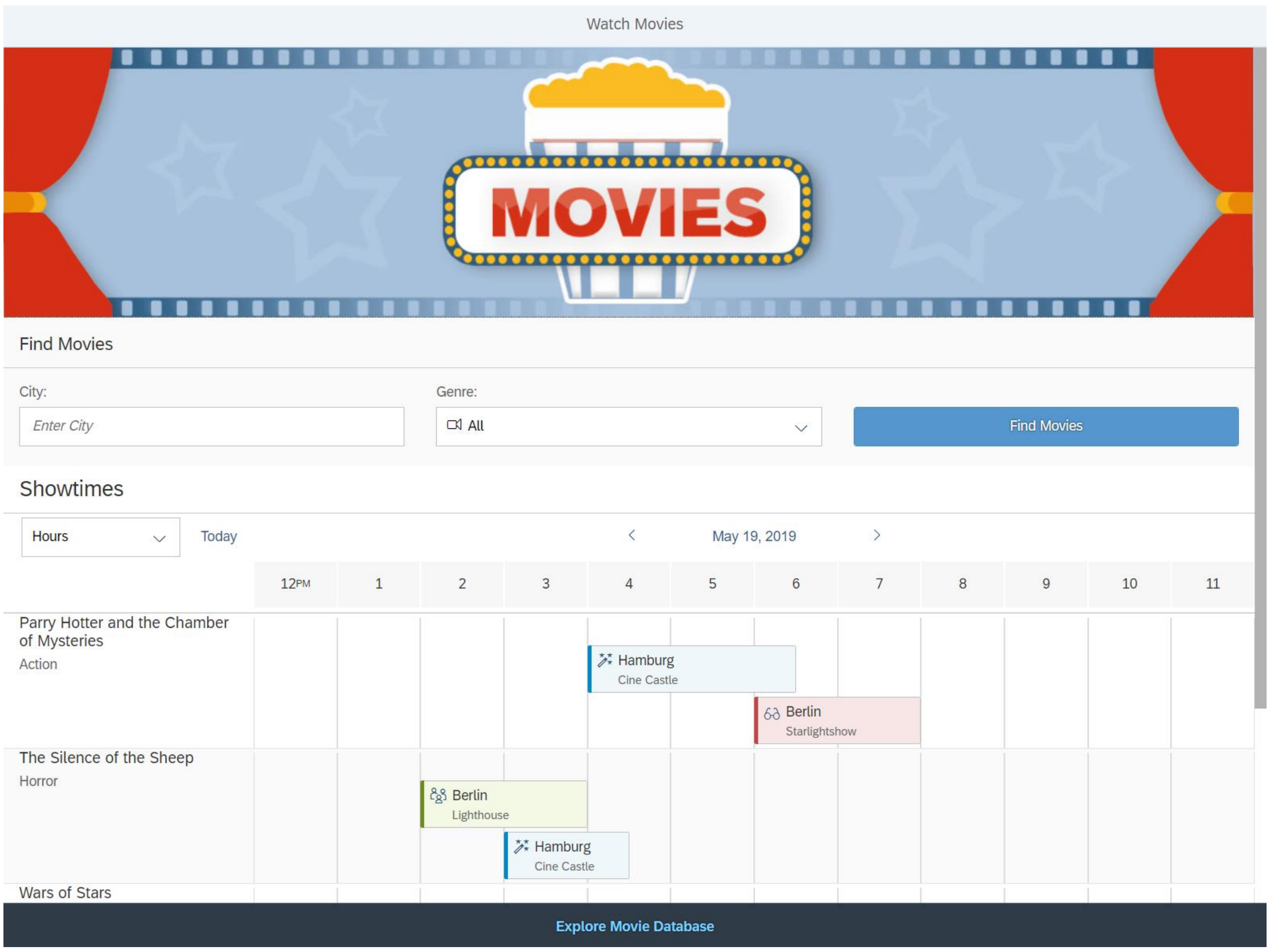


Figure 2 - PlanningCalendar displays movies and presentations in our app by means of data binding

Use expression binding to hide movies image on phone devices

webapp/App.view.xml

```
...
   <App id="app">
    <Page title="{i18n>title}">
       <Image
          visible="{= !${device>/system/phone} }"
          src="images/MoviesHeader.png"
          width="100%"
          tooltip="{i18n>imageTooltip}"
          press="sap.m.MessageToast.show('Do you feel like going to the movies?'
)"/>
       <f:SimpleForm
          id="form"

...
```

You will now use expression binding to hide the movies image for phone devices and show it for desktop and tablet computers.

The `visible` property is bound against the phone property of the device model. The logical "not" (!) will negate the returned value from the binding syntax. The result will be `visible=false` for phone devices and `visible=true` for others.

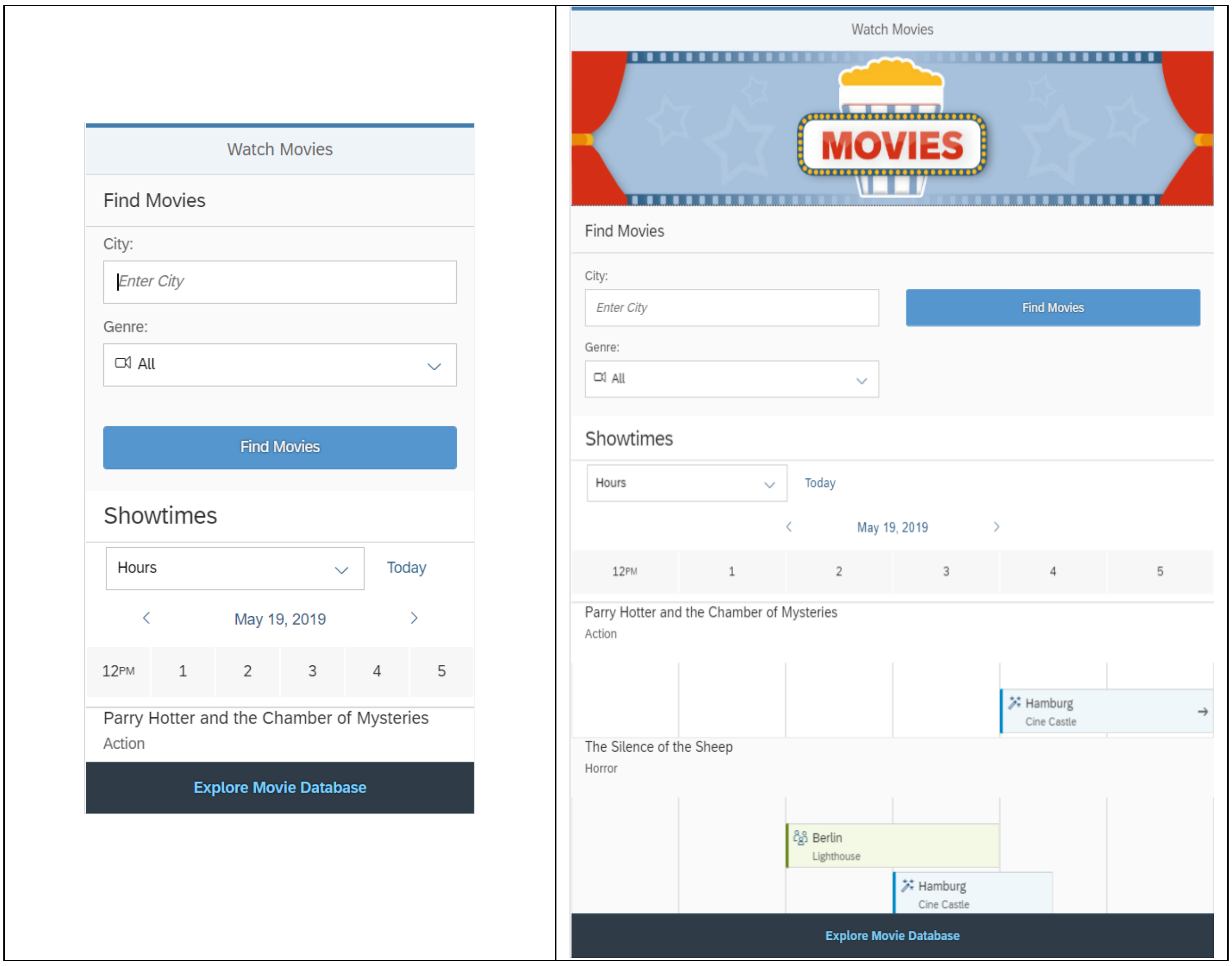


Figure 3 - Movie app in phone and tablet modes

Tip: Device emulation
To test this behavior, you can use the device emulation feature of Google Chrome by clicking on the button in the developer tools. The device model is initialized once at application startup, so you need to reload the page to see the changes in effect once you change the emulated device.

Move the hard-coded texts for headings and captions to the i18n properties

Your UI5 application should be ready to be easily translated into other languages. The localization folder - `i18n`, a short name for internationalization – contains the `*.properties`, one for each language. Our template project contains only one file by default. In real life, you would add more `*.properties` , depending on your target languages.

For example, the locale is declared in the file name suffix. The default file name is *i18n.properties*, then another *i18n_de.properies* should contain German localization. You can test it by setting your browser language to German or adding the URL parameter *sap-ui-language=DE_de* to the URL of your app.

Since you based your app on a SAPUI5 Application template, the I18n model is defined in the manifest with the name `i18n`. You will move your initially hard-coded texts to this file now based on what you have learned about data binding.

webapp/i18n/i18n.properties

```
#~~~ Home View ~~~~~~~~~~~~~~~~~~~~~~~~~~

title=Watch Movies
appTitle=MovieApp
appDescription=App Description
calTitle=Showtimes
imageTooltip=Movie illustration
labelCity=City
cityPlaceholder=Enter City
labelGenre=Genre
buttonMovieSearch=Find Movie
calendarTitle=Showtimes
search=Searching...
titleForm=Find Movies
footerLink=Explore the IMDb Movie Database
messageToast=Do you feel like going to the movies?
genreAll=All
genreAction=Action
genreHorror=Horror
genreScienceFiction=Science Fiction
```

Insert a `Home View` section to structure your entries. This improves readability when you have more than one view. Move the hard-coded texts from the `App.xml` file to the i18n properties. Hard-coded texts are maintained for the `App` title, the image `tooltip`, the `labels` for the `input` fields, the `placeholder` text of the two `input` fields, the button caption, and the title of the planning calendar. The three entries at the top of the i18n properties are pre-set in the template, and we change the title of the app to a more meaningful text. If you now our app again, you will see the updated title.

webapp/App.view.xml

```
...
<Page title="{i18n>title}">
   <Image
         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="ScienceFiction"
        text="{i18n>genreScienceFiction}"/>
        </Select>

…
   <Button
      text="{i18n>buttonMovieSearch}"
      press="onPress"/>
…

<PlanningCalendar
   id="calendar"
   startDate="{path: 'movies>/initDate', formatter: '.formatter.formatDate'}"
   rows="{movies>/Movies}"
   appointmentsVisualization="Filled">
   <toolbarContent>
      <Title text="{i18n>calendarTitle}" titleStyle="H3"/>
   </toolbarContent>
   <rows>
...
        <Toolbar>
           <ToolbarSpacer/>
           <Link emphasized="true" target="_blank" href="https://www.imdb.com/"
        text="{i18n>footerLink}"/>
```

Add the binding info to the controls after moving the text to the i18n properties.

Add the search functionality

Users should be able to search for a city and a genre in our movie app. The result should be displayed in the calendar. Technically, user input is captured in two filters that are then applied to the data model. First, create the filters in the controller.

webapp/App.controller.js

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

   return Controller.extend("movieapp.MovieApp.controller.App", {

      formatter: formatter,

      onInit: function () {
                  Log.info("Controller has been initialized.");

      },
onPress: function (sValue) {
   sap.ui.require(["sap/m/MessageToast"], function (oMessage) {
      var oResourceBundle =
this.getOwnerComponent().getModel("i18n").getResourceBundle();
      oMessage.show(oResourceBundle.getText("search") + sValue
);
   }.bind(this));

   var sCity = this.byId('city').getValue(),
      sGenre = this.byId('genre').getSelectedItem().getKey(),
      oCalendar = this.byId("calendar"),
      oRowBinding = oCalendar.getBinding("rows"),
      oFilterGenre,
      oFilterCity;

   // Create filters for genre and city according to user inputs
   oFilterGenre = sGenre ? new Filter("genre", FilterOperator.EQ, sGenre) : null;
   oFilterCity = sCity ? new Filter("info", FilterOperator.Contains, sCity) :
null;

   // Apply genre filter to calendar rows
   oRowBinding.filter(oFilterGenre);

   // Apply city filter to row appointments
   var aRows = oCalendar.getAggregation("rows");
   aRows.forEach(function (oItem) {
      var oAppointmentsBinding = oItem.getBinding("appointments");
      oAppointmentsBinding.filter(oFilterCity);
   });
}

….
```

You add code for the `onPress` event that gets fired when the *Find Movie* button is chosen. You collect the two string values the user entered into variables `sCity` and `sGenre`. You collect your `PlanningCalendar` in a variable by its ID `calendar` to derive the data binding information for the rows. This allows you to filter the rows by `genre` directly. For the appointments, you need to apply the filter in a loop with `forEach`.

When you create the filters, you check for empty variables. If the user left an input field empty, you need to apply the filters with a null value, so that the data gets refreshed. If you don't do this, filter settings from previous searches will not get overwritten, and the user will continue to see filtered data in your app, even if you have deleted the filter.
Job done!

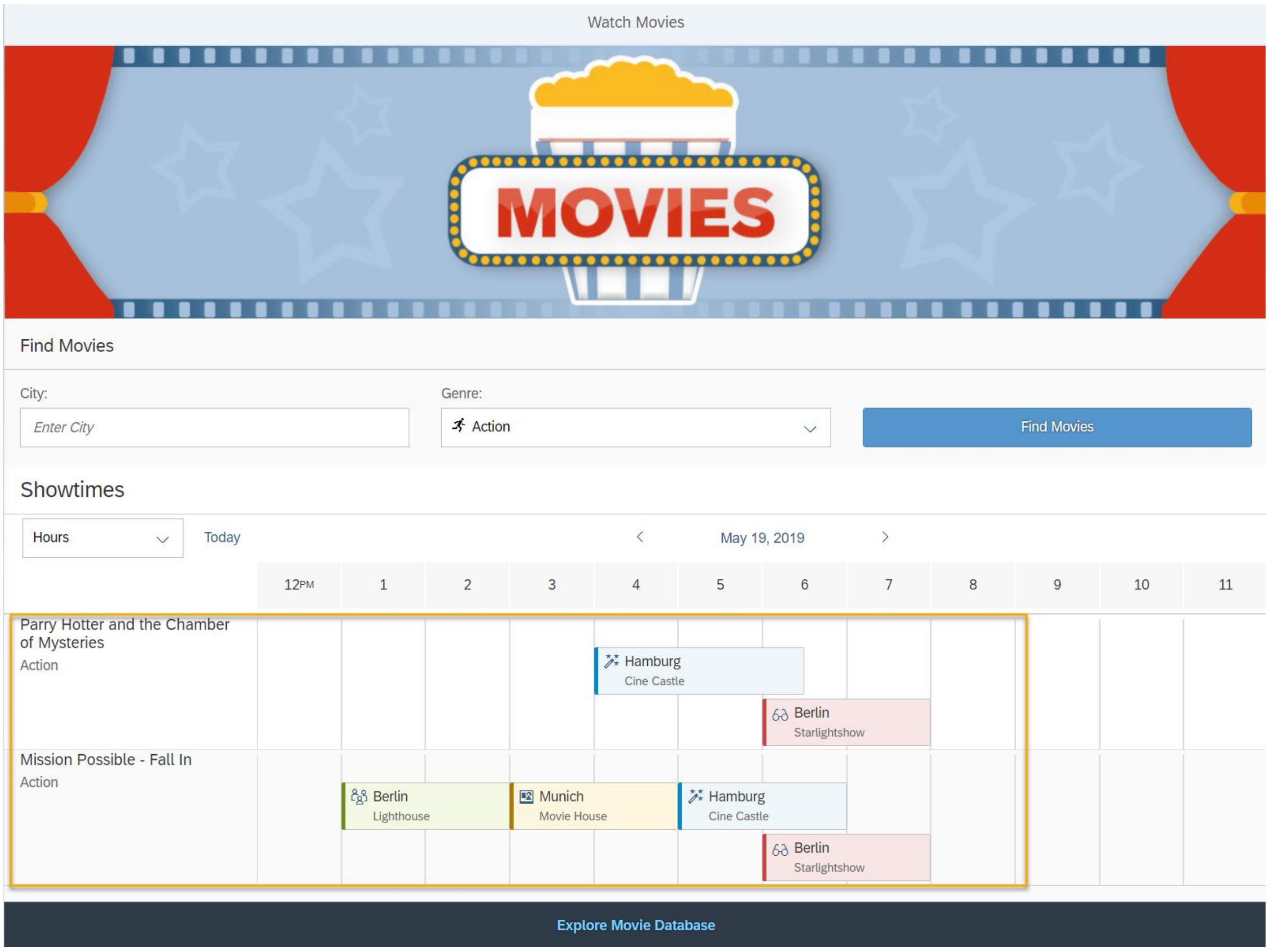


Figure 4 - app showing filter result for action movies

In the next unit, you will learn how to add another view and how to configure navigation.

RELATED MATERIAL

- Demo Kit: Data Binding Tutorial
- Demo Kit: Planning Calendar Samples
- Demo Kit: Expression Binding

www.sap.com/contactsap

THE BEST RUN