openSAP Evolved Web Apps with SAPUI5

Week 2 Unit 1: Starting Your Journey to Evolved Web Apps

Exercises

PUBLIC

TABLE OF CONTENTS

WEEK INTRODUCTION

Summary

In this week's units you will go through the essentials of building a simple UI5 application.
You will get acquainted with:

- The latest best practices for developing web apps
- Using UI controls and control libraries
- Data Binding and usage of models
- Navigation through the views in the app
- How product standards are injected into the framework

Preview

Everybody loves going to the movies! You want to develop an app that gives the users the possibility to show an overview of what's playing according to the movie genre.

Your movie application for movie enthusiasts will contain information about trending movies, and the user can search for a movie they are interested in and see more details about it.

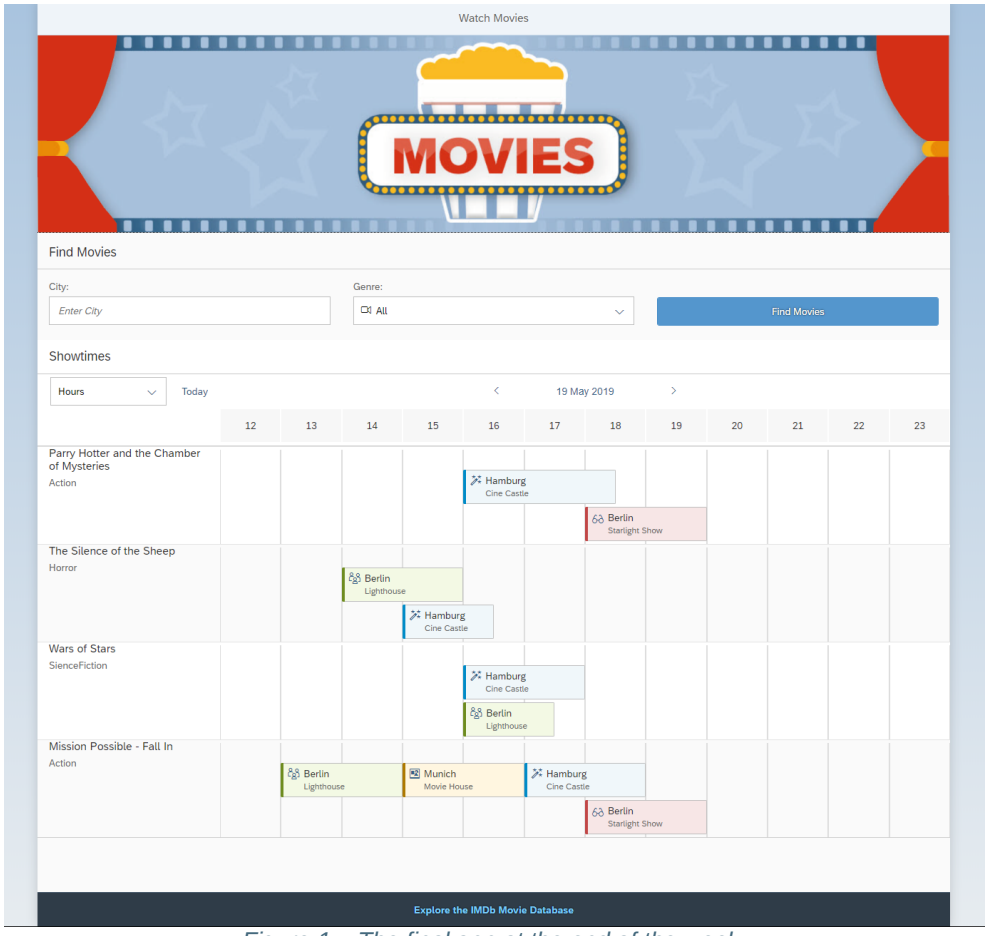


Figure 1 – The final app at the end of the week

STARTING YOUR JOURNEY TO EVOLVED WEB APPS

Summary

In this unit you will learn

- How to start application development with a template in SAP Web IDE
- Which best practices we recommend for UI5 app development
- How to add an image to the app

Preview

In this unit you will start your application from a template and will apply some of the best practices and tips in the development process with UI5. In the end you will add an image as a banner at the top of the movie app.

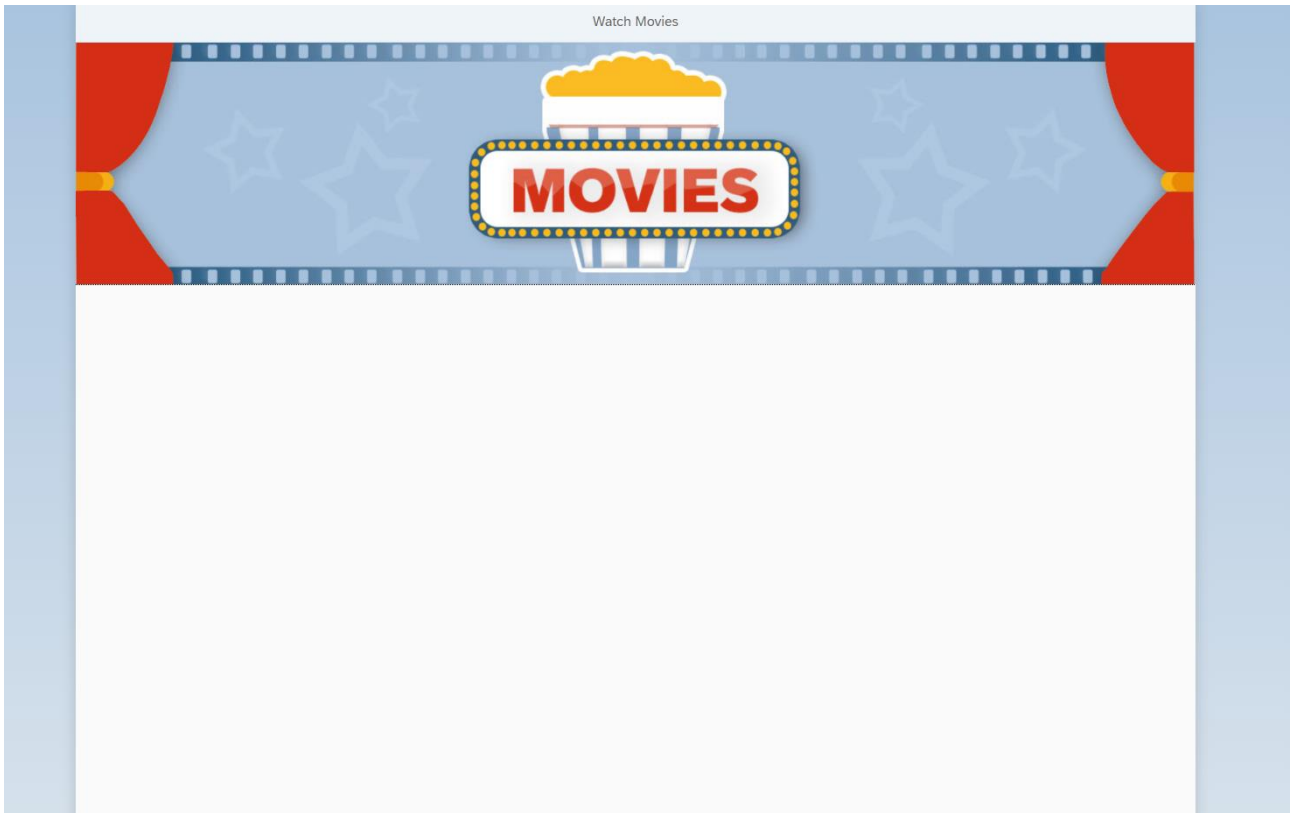
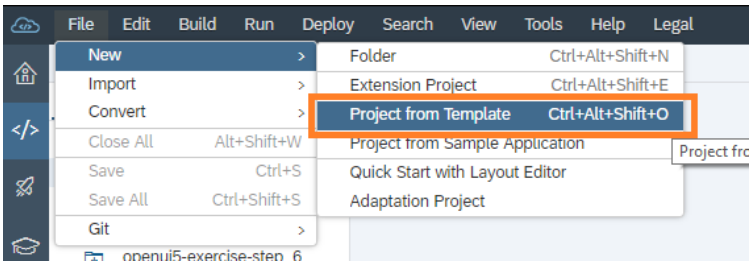
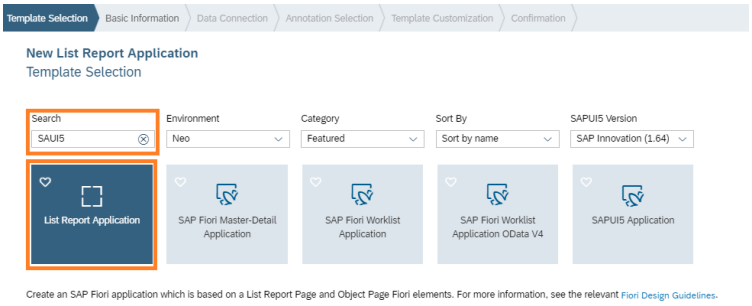
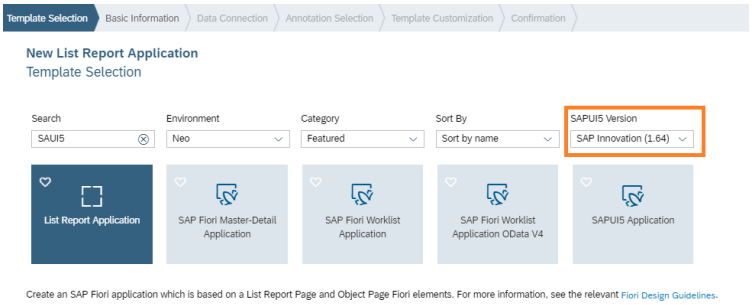
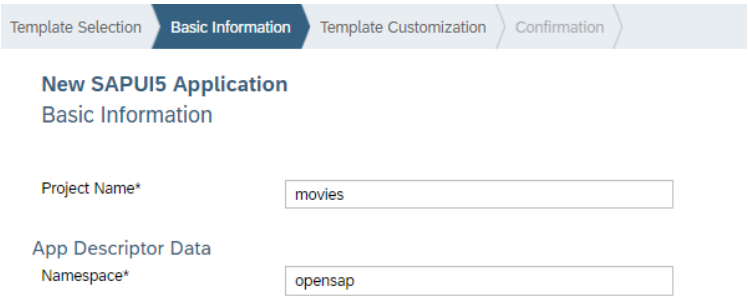
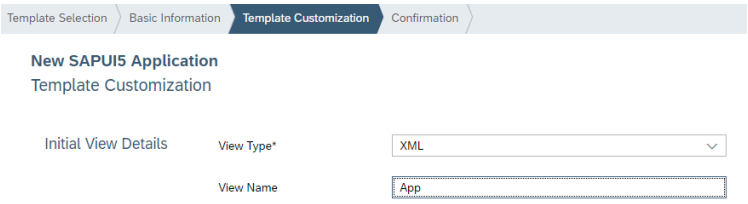
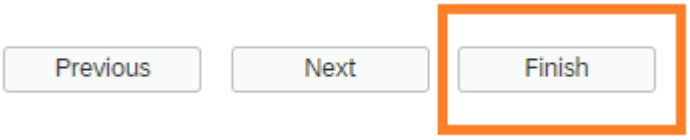
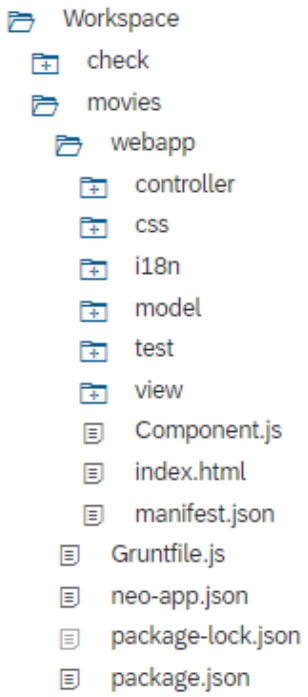
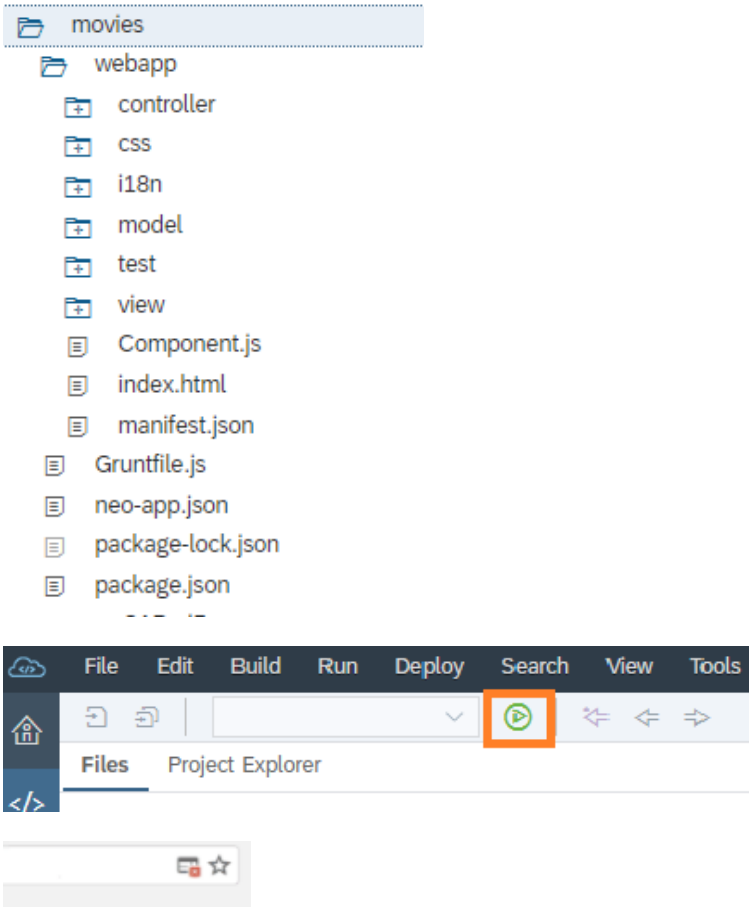


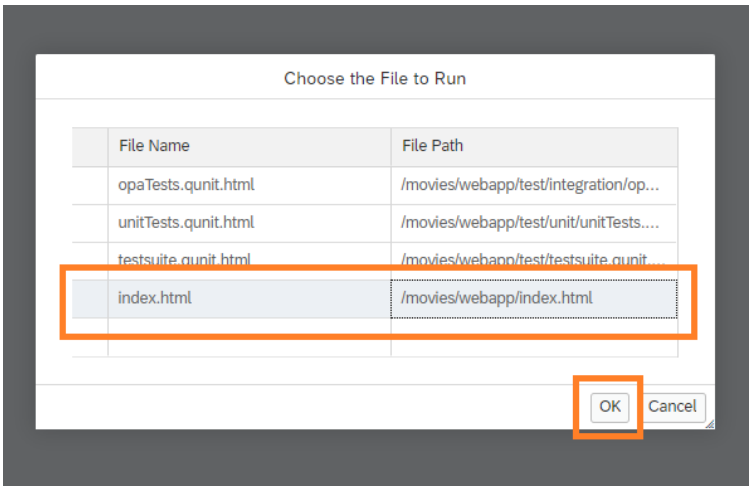
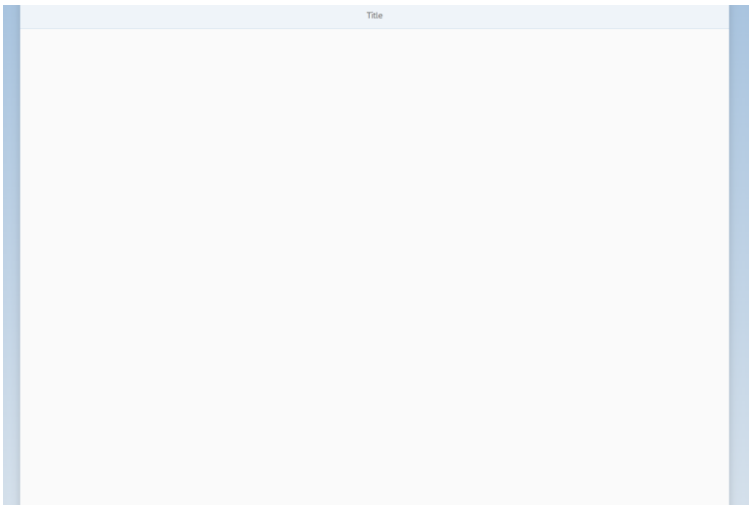
Figure 2 – The SAPUI5 Application template with an additional image

Create the App from the Template

Explanation	Screenshot
1. Choose *File* → *New* → *Project from Template* in your SAP Web IDE workspace to open the wizard with the *Template Selection* step.	
2. Search for **SAPUI5** and select *SAPUI5 Application* to continue the wizard with this template. **Note:** If you cannot find this template, select *All* in the *Category* dropdown menu.	
3. The latest available SAPUI5 version should be selected by default, for example *SAP Innovation (1.64)*. If not, select it, it is generally recommended to use the latest version.	
4. Choose *Next* to go to the next wizard step.	

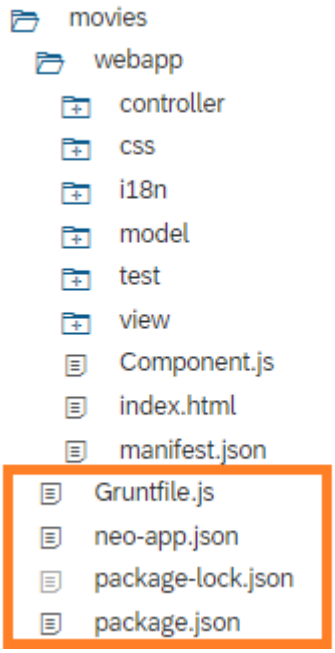
Explanation	Screenshot
5. In the *Basic Information* step, enter the following values: • *Project Name* = **movies** • *Namespace* = **opensap**	
6. Choose *Next* to go to the next wizard step.	
7. In the *Template Customization* step, enter the following values: • *View Type* = *XML* • *View Name* = *App* **Note:** Other view types (JSON, JavaScript, HTML) exist, but we generally recommend using XML views to be able to define the view declaratively and clearly separate view and controller logic.	
8. Choose *Finish* to end the wizard and generate the app.	

Explanation	Screenshot
9. Check out that a new folder – `movies` – is added to your local workspace. It should contain the files and folders of the initial app as displayed in the screenshot on the right.	
10. Run the app: Select the root folder of the project, and choose the *Run* icon to run the app. **Info:** If your browser prevents you from previewing your application, please disable the pop-up blocker by clicking on the blocker icon in the browser search bar.	

Explanation	Screenshot
11. The first time the app is run, a dialog is shown asking which file should be started. Select the `index.html` file and choose *OK*.	
12. A preview of the app opens in a separate browser tab for testing purposes. The app is still empty!	

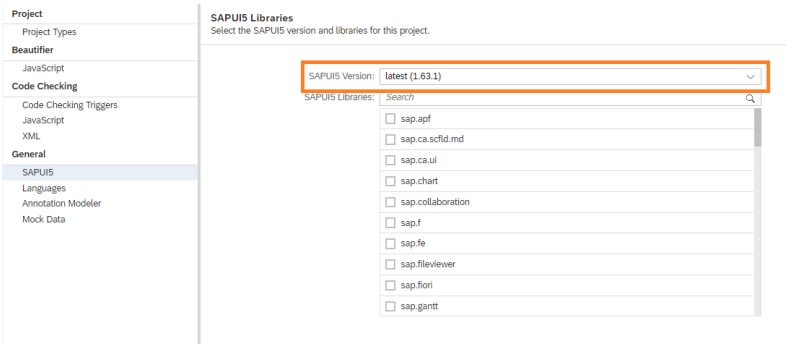
Evolved UI5 Best Practices

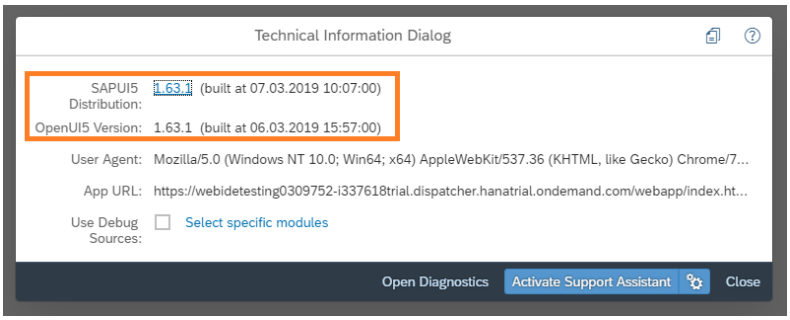
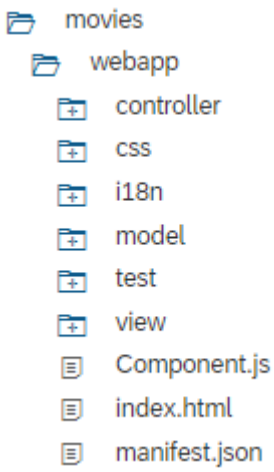
Before you add new features to the app, see which best practices recommended by SAP are already included in the coding and structure of the template. Each template is updated frequently to match the latest recommendations and is set up according to the best practices for app developers.

Explanation	Screenshot
1. Look at the project structure. In the root directory, there are several configuration files.	

Explanation	Screenshot

2. The `neo-app.json` file is needed for the SAP Web IDE. It defines some SAP Cloud Platform-specific settings, such as required services for the application.	neo-app.json ×
	`{`
	`  "welcomeFile": "/webapp/index.html",`
	`  "routes": [`
	`  {`
	`  "path": "/resources",`
	`  "target": {`
	`  "type": "service",`
	`  "name": "sapui5",`
	`  "entryPath": "/resources"`
	`  },`
	`  "description": "SAPUI5 Resources"`
	`  },`
	`  {`
	`  "path": "/test-resources",`
	`  "target": {`
	`  "type": "service",`
	`  "name": "sapui5",`
	`  "entryPath": "/test-resources"`
	`  },`
	`  "description": "SAPUI5 Test Resources"`
	`  }`
	`  ],`
	`  "sendWelcomeFileRedirect": true`
	`}`

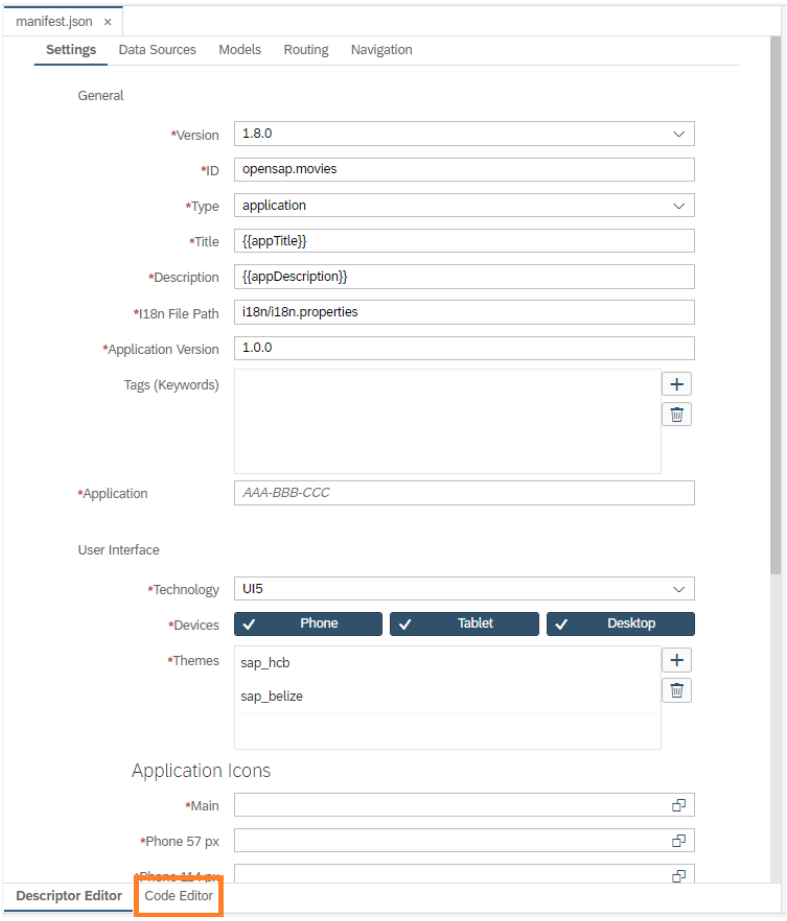
3. You can change the UI5 version either by manually adding the `version` field to the `neo-app.json` file, or by right-clicking on the project and choosing *Project → Project Settings*. 4. Under *General → SAPUI5*, select a suitable version that you want to ship your app with, if the one selected does not fit you need. **Note:** It is recommended to set a specific UI5 version for your app before making it available to your users. This ensures that your app runs in the environment you tested it in during your development phase.	

Explanation	Screenshot
5. Run the app again, and press **Ctrl+Alt+Shift+P** to open the *Technical Information Dialog*. This contains some technical configurations of the app. For example, notice that the UI5 runtime version is the one you selected.	
6. All the application code is placed in the `webapp` folder. **Info:** The content of the `webapp` folder is typically deployed to a Web server to run the app in the cloud**.**	
UI5 is bootstrapped relatively in the `index.html` file, as we can see in the screenshot on the right, and configured in the `neo-app.json` file as explained above. This allows you to easily update the UI5 version without touching the application code.	
The initial components are defined in a declarative way. Directly executable code is not used in the HTML files, because this makes the files vulnerable. Instead, a good practice is to enable the `ComponentSupport` module in the bootstrapping script. Then, to declare the desired component in the body via a `div` tag. This will instantiate the component when the `onInit`	

Explanation	Screenshot
method is executed. The `ComponentSupport` class provides functionality which allows you to declare your components in HTML.	

The bootstrapping tag `data-sap-ui-async="true"` in your `index.html` file loads the modules for all declared libraries asynchronously. This way the files are retrieved in parallel which speeds up the loading of the background processes and speeds up the whole app too.	index.html
	`<!DOCTYPE html>`
	`<html>`
	`  <head>`
	`  <meta charset="utf-8">`
	`  <meta name="viewport" content="width=device-width, initial-scale=1.0">`
	`  <title>movies</title>`
	`  <script id="sap-ui-bootstrap"`
	`  src="../../resources/sap-ui-core.js"`
	`  data-sap-ui-theme="sap_belize"`
	`  data-sap-ui-resourceroots='{"opensap.movies": "./"}'`
	`  data-sap-ui-compatVersion="edge"`
	`  data-sap-ui-oninit="module:sap/ui/core/ComponentSupport"`
	`  data-sap-ui-async="true"`
	`  data-sap-ui-frameOptions="trusted">`
	`  </script>`
	`  </head>`
	`  <body class="sapUiBody">`
	`  <div data-sap-ui-component data-name="opensap.movies" data-id="container" data-settings='{"id" : "movies"}'></div>`
	`  </body>`
	`</html>`

All UI assets are encapsulated in a component that is instantiated from our `index.html` page.
 	
Components are independent and reusable parts used in UI5 applications. The component configuration is stored in the `manifest.json` – the so-called application descriptor.
 	
Open the `Component.js` file and take a look at the implementation of the component. Note that the metadata is loaded from a so-called `manifest`.
 	
The `init` function typically configures additional models that are not defined in the manifest and initializes the router.
	Component.js
	`sap.ui.define([`
	`  "sap/ui/core/UIComponent",`
	`  "sap/ui/Device",`
	`  "opensap/movies/model/models"`
	`], function (UIComponent, Device, models) {`
	`  "use strict";`
	
	`  return UIComponent.extend("opensap.movies.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");`
	`  }`
	`  });`
	`});`

Explanation	Screenshot
The `manifest.json` or app descriptor file clearly separates the application coding from the configuration settings and makes the app more flexible. There are two modes – *Descriptor Editor* (default) and *Code Editor*. Both show the same information, but in a different way. Switch to *Code Editor* mode and explore it.	

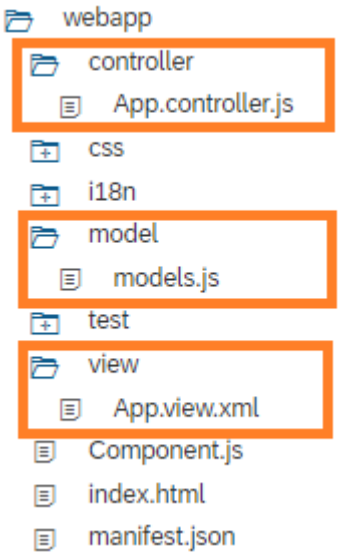
Explanation	Screenshot

You can also use the descriptor file to define application settings, load additional resources, and instantiate models like the `i18n` resource bundle automatically.	manifest.json
	`{`
	`"_version": "1.8.0",`
	`"sap.app": {`
	`"id": "opensap.movies",`
	`"type": "application",`
	`"i18n": "i18n/i18n.properties",`
	`"applicationVersion": {`
	`"version": "1.0.0"`
	`},`
	`"title": "{{appTitle}}",`
	`"description": "{{appDescription}}",`
	`"sourceTemplate": {`
	`"id": "ui5template.basicSAPUI5ApplicationProject",`
	`"version": "1.40.12"`
	`}`
	`},`
	
	`"sap.ui": {`
	`"technology": "UI5",`
	`"icons": {`
	`"icon": "",`
	`"favIcon": "",`
	`"phone": "",`
	`"phone@2": "",`
	`"tablet": "",`
	`"tablet@2": ""`
	`},`
	`"deviceTypes": {`
	`"desktop": true,`
	`"tablet": true,`
	`"phone": true`
	`},`

The `routing` configuration is used to load and show the XML views of the application. It shows a single route to the app view in this project. The connection of the views is accomplished by triggering navigation events and letting the router do the work.
 	
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.
	`"settings": {`
	`"bundleName": "opensap.movies.i18n.i18n"`
	`}`
	`}`
	`},`
	`"resources": {`
	`"css": [{`
	`"uri": "css/style.css"`
	`}]`
	`},`
	`"routing": {`
	`"config": {`
	`"routerClass": "sap.m.routing.Router",`
	`"viewType": "XML",`
	`"async": true,`
	`"viewPath": "opensap.movies.view",`
	`"controlAggregation": "pages",`
	`"controlId": "idAppControl",`
	`"clearControlAggregation": false`
	`},`
	`"routes": [{`
	`"name": "RouteApp",`
	`"pattern": "RouteApp",`
	`"target": ["TargetApp"]`
	`}],`
	`"targets": {`
	`"TargetApp": {`
	`"viewType": "XML",`
	`"transition": "slide",`
	`"clearControlAggregation": false,`
	`"viewName": "App"`
	`}`
	`}`
	`}`
	`}`
	`}`

Explanation	Screenshot

To improve the performance of your app, you should always load resources asynchronously.
 	
The `manifest.json` contains additional settings for asynchronous loading of the root view and the views instantiated by the routing configuration.
	``
	60 "settings": {
	61 "bundleName": "opensap.movies.i18n.i18n"
	62 }
	63 }
	64 },
	65 "resources": {
	66 "css": [{
	67 "uri": "css/style.css"
	68 }]
	69 },
	70 "routing": {
	71 "config": {
	72 "routerClass": "sap.m.routing.Router",
	73 "viewType": "XML",
	74 "async": true,
	75 "viewPath": "opensap.movies.view",
	76 "controlAggregation": "pages",
	77 "controlId": "idAppControl",
	78 "clearControlAggregation": false
	79 },
	80 "routes": [{
	81 "name": "RouteApp",
	82 "pattern": "RouteApp",
	83 "target": ["TargetApp"]
	84 }],
	85 "targets": {
	86 "TargetApp": {
	87 "viewType": "XML",
	88 "transition": "slide",
	89 "clearControlAggregation": false,
	90 "viewName": "App"
	91 }
	92 }
	93 }
	94 }
	95 }
	``
	(highlighted: `"async": true,`)

The model-view-controller pattern (MVC) applied in UI5, is reflected in the `webapp` folder structure: • The `model` folder contains additional logic related to data models. Models are used for data management and control filtering, sorting and formatting of data. • The `view` folder contains views and fragments which define the UI of your app. • The `controller` folder contains controllers and helper classes with logic to define the behavior of your views.	

Explanation	Screenshot

Asynchronous loading of dependencies can also be seen in the `App.controller.js` file included in the template.
 	
In the controller file, `sap.ui.define` is used for asynchronous loading of the controller base class before extending it.
	App.controller.js ×
	`sap.ui.define([`
	`  "sap/ui/core/mvc/Controller"`
	`], function (Controller) {`
	`  "use strict";`
	
	`  return Controller.extend("opensap.movies.controller.App", {`
	`  onInit: function () {`
	
	`  }`
	`  });`
	`});`

The `CSS` and `i18n` folders contain the style-related and internationalization-related files.	webapp controller css i18n model test view Component.js index.html manifest.json

Customize the template

Let's change the template a little bit to make it more applicable to the user scenario and change the generated strings from the template to more meaningful ones.

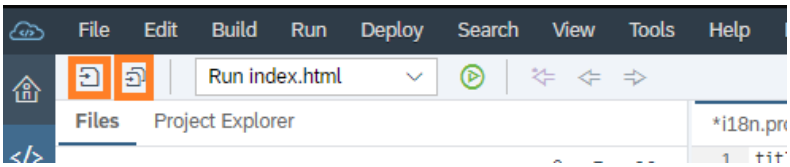
webapp/i18n/i18n.properties

```
title=Title
appTitle=MovieApp
appDescription=App Description

title=Watch Movies
appTitle=Watch Movies
appDescription=Find your favorite movie showtimes
```

Delete the existing definitions of the properties and define the new texts.

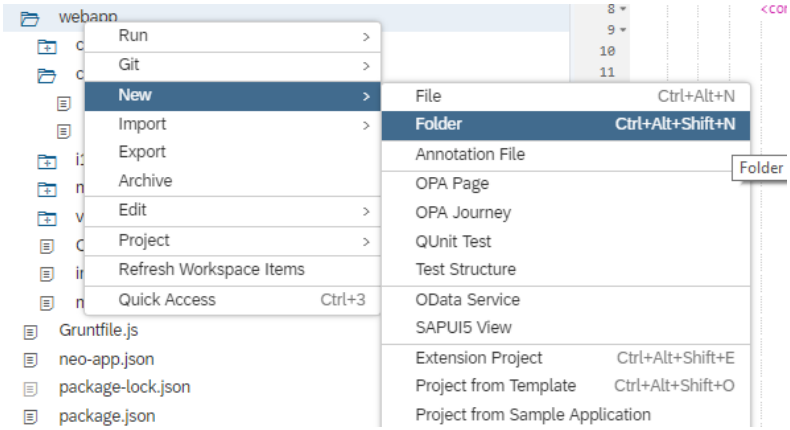
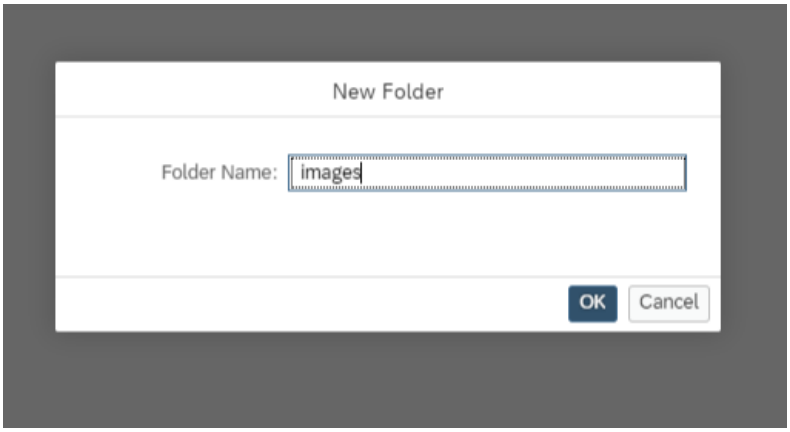
Change the `title` from the title area of the Page to **Watch Movies**. The `appTitle` and `appDescription` strings are used in the `manifest.json` for describing the whole app. Use the same `appTitle` as the `title` and a brief description of the main app purpose as `appDescription`.

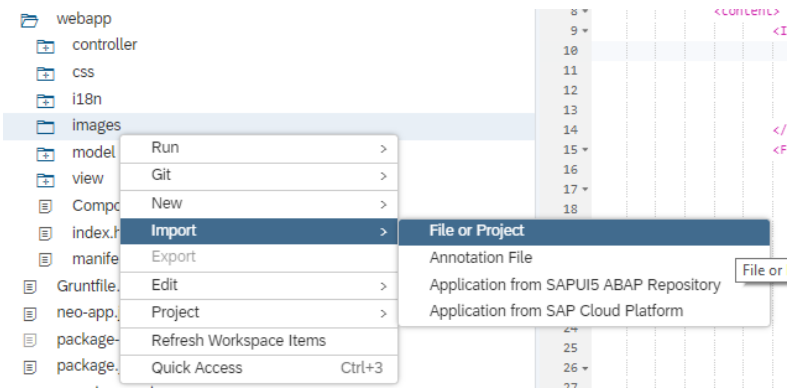
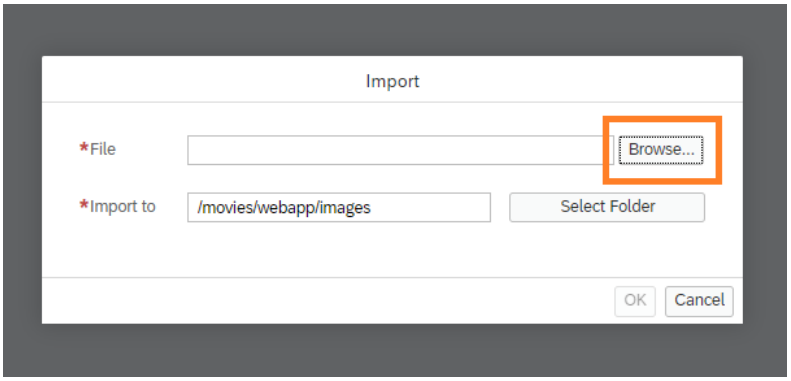
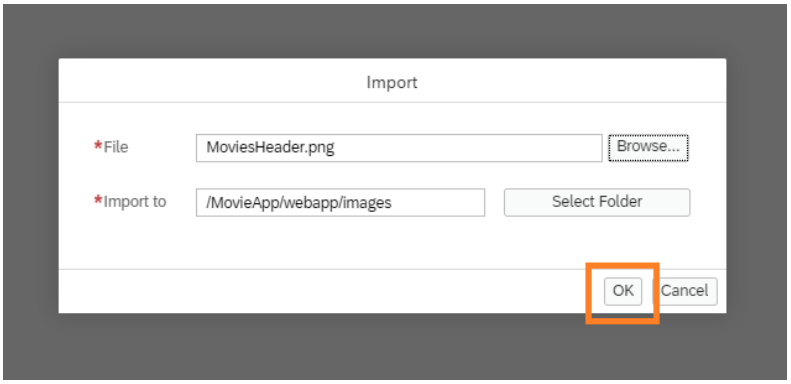
Explanation	Screenshot
1. After writing the code, save the changes with one of the buttons at the top of SAP Web IDE. The button on the left is used for saving the current file. The next one is used for saving the whole project. The keyboard shortcuts could be used accordingly too – **Ctrl+S** and **Ctrl+Shift+S**.	

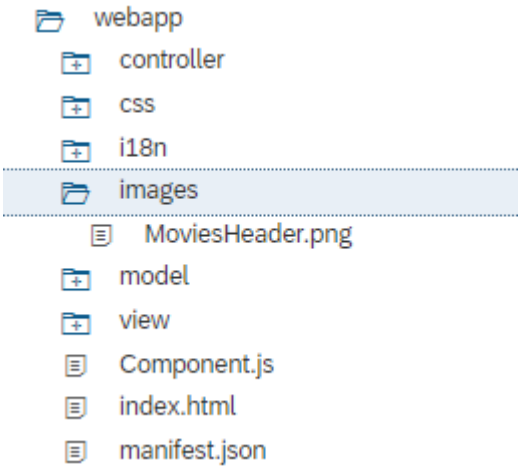
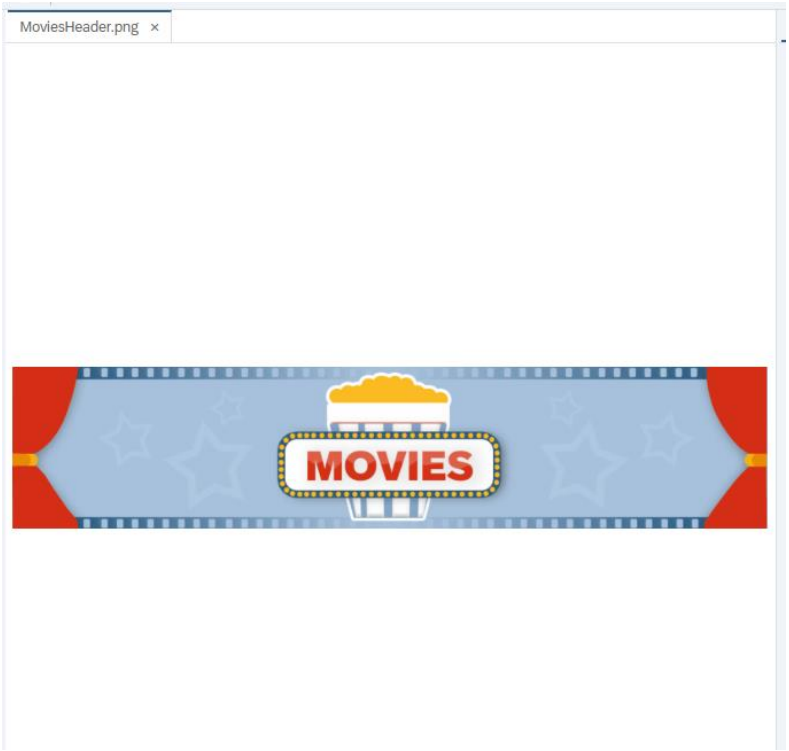
Import an image

As a next step, include an image to the app.

First, you add the image resource file to the project. Then you display it on the page via the UI5 controls `sap.m.Image`.

Explanation	Screenshot
1. First, create a new folder `images`, where the image resources should be stored. Right-click on the `webapp` folder. Then choose *New* → *Folder.*	
2. Name the folder **images** and choose *OK*.	
3. Download the image to be imported from the course repository: https://raw.githubusercontent.com/SAP/openSAP-ui5-course/master/import/MoviesHeader.png	

Explanation	Screenshot
4. To import the image to the newly created folder, right-click on the `images` folder. Then choose *Import → File or Project.*	
5. A dialog will appear asking for the image destination on your machine and for the path to the folder in the SAP Web IDE project where it will be imported. Choose *Browse*, and search for the image on your machine. The path to the import destination will automatically be populated with the `images` folder you just created.	
6. Choose *OK* button to confirm the action.	
7. The image resource should appear in the `images` folder.	

Explanation	Screenshot
	
8. Preview the image. It should appear on the right.	

Use the image in the app

Now you add the `sap.m.Image` control. For now, you can think of it as a wrapper around the actual image resource.

webapp/view/App.view.xml

```
…
        <App id="app">
                <pages>
                        <Page title="{i18n>title}">
                                <content>
                                        <Image
                                                src="images/MoviesHeader.png"
                                                width="100%"
                                                tooltip="Movie illustration">
                                        </Image>
                                </content>
                        </Page>
                </pages>
        </App>
 …
```

With adding the `sap.m.Image` to the main view, you specify where the source image is contained in the project via the `src` property. You also set a tooltip text via the `tooltip` property of the control.

More information about the settings of each UI5 element will be explained in the next unit. As a reference, you can look at the Demo Kit.

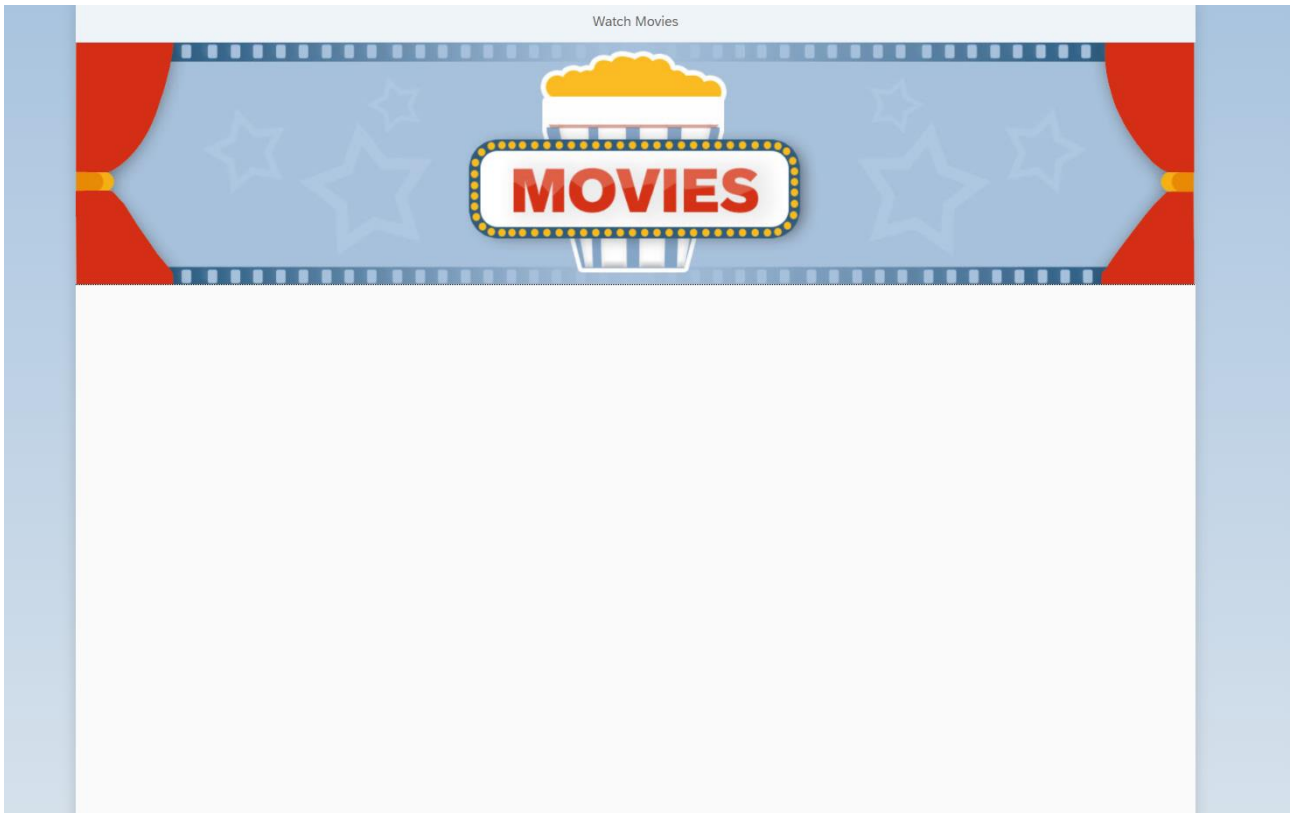


Figure 3 – The result app at the end of the unit

★ CHALLENGE YOURSELF: LINK A MOVIE DATABASE

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

Early user feedback has shown that a desired feature for the movie app is to have a quick link to a movie database in the footer. That way the users can explore a movie before choosing to reserve seats for it in the cinema. One good answer to this requirement is the **Internet Movie Database** (IMDb).

Can you help increasing the user experience by adding this cool feature?

Details

- **Position:** Footer
- **Text:** Explore Movie Database
- **Link:** IMDb (should open in a new tab)

Preview

Figure 4 – The footer shows a link to a movie database

Hints:

- `sap.m.Page` has nice examples of using a footer
- `sap.m.Link` might be helpful as well
- Toolbars control the position of its content

RELATED MATERIAL

- Demo Kit: Controls section
- Demo Kit: Best Practices for App Developers

Coding Samples

Any software coding or code lines/strings ("Code") provided in this documentation are only examples and are not intended for use in a production system environment. The Code is only intended to better explain and visualize the syntax and phrasing rules for certain SAP coding. SAP does not warrant the correctness or completeness of the Code provided herein and SAP shall not be liable for errors or damages cause by use of the Code, except where such damages were caused by SAP with intent or with gross negligence.

www.sap.com/contactsap

THE BEST RUN SAP®