WEEK 1 UNIT 4
CREATING A CONFIGURABLE APP COMPONENT

Please perform the exercises below in your app project as shown in the video.

Table of Contents

Preview

Figure 1 - Preview of the app after doing this unit's exercises

1 CREATING A CONFIGURABLE APP COMPONENT

In this step, we will encapsulate all UI assets in a component and introduce the app descriptor.

Components are independent and reusable parts used in SAPUI5 applications. For example, SAP Fiori apps usually run in an SAP Fiori launchpad which acts as a so-called component container. It manages and loads the apps by loading the corresponding component of the app. Whenever we address resources of the app, we will now do this relatively to the component (instead of relatively to the index.html). This architectural change allows our app to be used in more flexible environments than our static index.html page.

webapp/Component.js (NEW)

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

   return UIComponent.extend("opensap.myapp.Component", {

      metadata : {
         manifest: "json"
      },

      init : function () {
         // call the init function of the parent
         UIComponent.prototype.init.apply(this, arguments);

         // additional initialization can be done here
      }

   });
});
```

First, in the webapp folder, we create an initial `Component.js` file that will hold our application setup. The `init` function of the component is automatically invoked by SAPUI5 when the component is instantiated. Our component inherits from base class `sap.ui.core.UIComponent` and must call the `init` function of the base class in the overridden `init` method. The `Component.js` file consists of two parts: The new metadata section that refers to the app descriptor which we still have to implement, and the previously introduced `init` function that is called when the component is initialized. Instead of instantiating the root view directly from the index.html file as we did previously, the component will now manage the display of the app view.

webapp/index.html

```
<!DOCTYPE html>
<html>
<head>
   …
   <script>
      sap.ui.getCore().attachInit(function () {
         new sap.ui.core.ComponentContainer({
            name : "opensap.myapp"
         }).placeAt("content");
      });
   </script>
</head>
<body class="sapUiBody" id="content">
</body>
</html>
```

On the index page, we now instantiate the component instead of the app view. The factory method `sap.ui.core.ComponentContainer` instantiates the component by searching for a `Component.js` file in the namespace that is passed as an argument. The component automatically displays the root view that is defined within the app descriptor.

webapp/manifest.json (NEW)

```
{
   "_version": "1.3.0",
   "sap.app": {
      "_version": "1.3.0",
      "id": "opensap.myapp",
      "type": "application",
      "title": "{{appTitle}}",
      "description": "{{appDescription}}",
      "applicationVersion": {
         "version": "1.0.0"
      }
   },
   "sap.ui": {
      "_version": "1.3.0",
      "technology": "UI5",
      "deviceTypes": {
         "desktop": true,
         "tablet": true,
         "phone": true
      },
      "supportedThemes": [
         "sap_bluecrystal"
      ]
   },
   "sap.ui5": {
      "_version": "1.2.0",
      "rootView": {
         "viewName": "opensap.myapp.view.App",
         "type": "XML",
         "id": "app"
      },
      "autoPrefixId": true,
      "dependencies": {
         "minUI5Version": "1.34",
```

```
        "libs": {
          "sap.ui.core": {
            "minVersion": "1.34.0"
          },
          "sap.m": {
            "minVersion": "1.34.0"
          },
          "sap.ui.layout": {
            "minVersion": "1.34.0"
          }
        }
      },
      "contentDensities": {
        "compact": true,
        "cozy": true
      }
    }
}
```

All application-specific configuration settings will now be put into a separate descriptor file called `manifest.json`. This clearly separates the application code from the configuration settings and makes our app even more flexible. The SAP Fiori launchpad acts as an application container and instantiates the app without having an application-specific HTML file for the bootstrap. Instead, the descriptor file is parsed and the component is loaded into the current HTML page. This allows several apps to be displayed in the same context. Each app can define local settings, such as dependent libraries, supported themes and device types, and more.

The content of the `manifest.json` file is a configuration object in JSON format that contains all application settings and parameters. The manifest file is also called the “descriptor for applications, components, and libraries”, and is sometimes abbreviated as “descriptor” or “app descriptor” when used for applications. It is stored in the `webapp` folder and read by SAPUI5 to instantiate the component that we will create in this step. For SAPUI5 applications, there are three important sections defined by namespaces in the `manifest.json` file: `sap.app`, `sap.ui` and `sap.ui5`.

If you now call the `index.html` file, the app should still look the same, but is now packaged into a UI component.

Conventions

- The component file is named `Component.js`.
- Together with all UI assets of the app, the component is located in the `webapp` folder.
- If the `index.html` file is used productively, it is located in the `webapp` folder.
- The descriptor file is named `manifest.json` and located in the `webapp` folder.
- Use translatable strings for the title and the description of the app.

Related Information

Descriptor for Applications, Components, and Libraries