WEEK 1 UNIT 3
STRUCTURING WITH CONTROLLERS AND MODULES

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

Table of Contents

1 Controllers 2
2 Modules 4

Preview

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

1 CONTROLLERS

In this step, we replace the text with a button and show the “Hello World” message when the button is chosen. The handling of this event is implemented in the controller of the view.

Preview

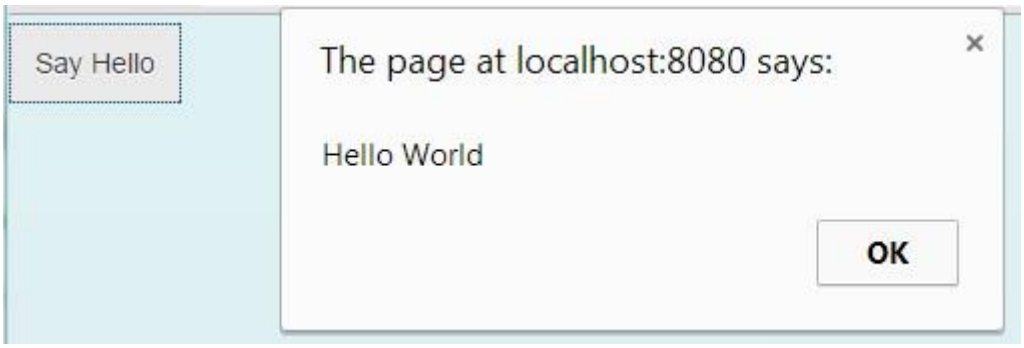

Figure 2: A “Say Hello” button is added

webapp/view/App.view.xml

```
<mvc:View
   displayBlock="true"
   xmlns:mvc="sap.ui.core.mvc"
   xmlns="sap.m"
   controllerName="opensap.myapp.controller.App">
   <Carousel>
      …
   </Carousel>
   <Button
      text="Say Hello!"
      press="onShowHello"/>
</mvc:View>
```

We add a reference to the controller, and add a button that carries the text “Say Hello”. The button triggers the `onShowHello` event handler function when being pressed. For the event handler to be found, we also have to specify the name of the controller that holds the `onShowHello` function with the view attribute `controllerName`.
A view does not necessarily need an explicitly assigned controller. You do not have to create a controller if the view is just displaying information and no additional functionality is required. If a controller is specified, it is instantiated after the view has been loaded.

webapp/controller/App.controller.js (NEW)

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

});
```

We create the folder `webapp/controller` and a new file `App.controller.js` inside. For now, we ignore the code that manages the required modules. We will explain this part in the next exercise.

webapp/controller/App.controller.js

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

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

      onShowHello : function () {
         // show a native JavaScript alert
         alert("Hello World");
      }
   });
});
```

We define the app controller in its own file by extending the `Controller` object of the SAPUI5 core. In the beginning, it holds only a single function called `onShowHello` that handles the pressing of the button by showing an alert.

Conventions

- Controller names are capitalized.
- Controllers carry the same name as the related view (if there is a 1:1 relationship) .
- Event handlers are prefixed with `on`.
- Controller names always end with *.controller.js.

Related Information

API Reference: sap.ui.define

2 MODULES

In this step, we replace the alert from the last exercise with a proper "Message Toast" from the `sap.m library`. The required modules are loaded asynchronously.

Preview

Figure 6: A message toast displays the "Hello World" message

webapp/controller/App.controller.js

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

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

      onShowHello : function () {
         MessageToast.show("Hello openSAP");
      }
   });
});
```

We extend the array of required modules with the fully qualified path to `sap.m.MessageToast`. Once both modules, `Controller` and `MessageToast`, have been loaded and the button has been pressed, the callback function is called and we can use both objects by accessing the parameters passed to the function.
This asynchronous module definition (AMD) syntax allows to clearly separate module loading from code execution and greatly improves the performance of the application. The browser can decide when and how the code resources are loaded before executing them.

Conventions

- Use sap.ui.define for controllers and all other JavaScript modules to define a global namespace. With the namespace, the object can be addressed throughout the application.
- Use sap.ui.require to load dependencies asynchronously, but without declaring a namespace, for example, code that is directly executed and is not referenced in other places.
- Name the parameters of the callback function according to the names of the respective dependencies.

Related Information

API Reference: sap.ui.define
API Reference: sap.ui.require

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 productive 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.