SAPUI5 Overview

Leonardo Britz, SAP
2024

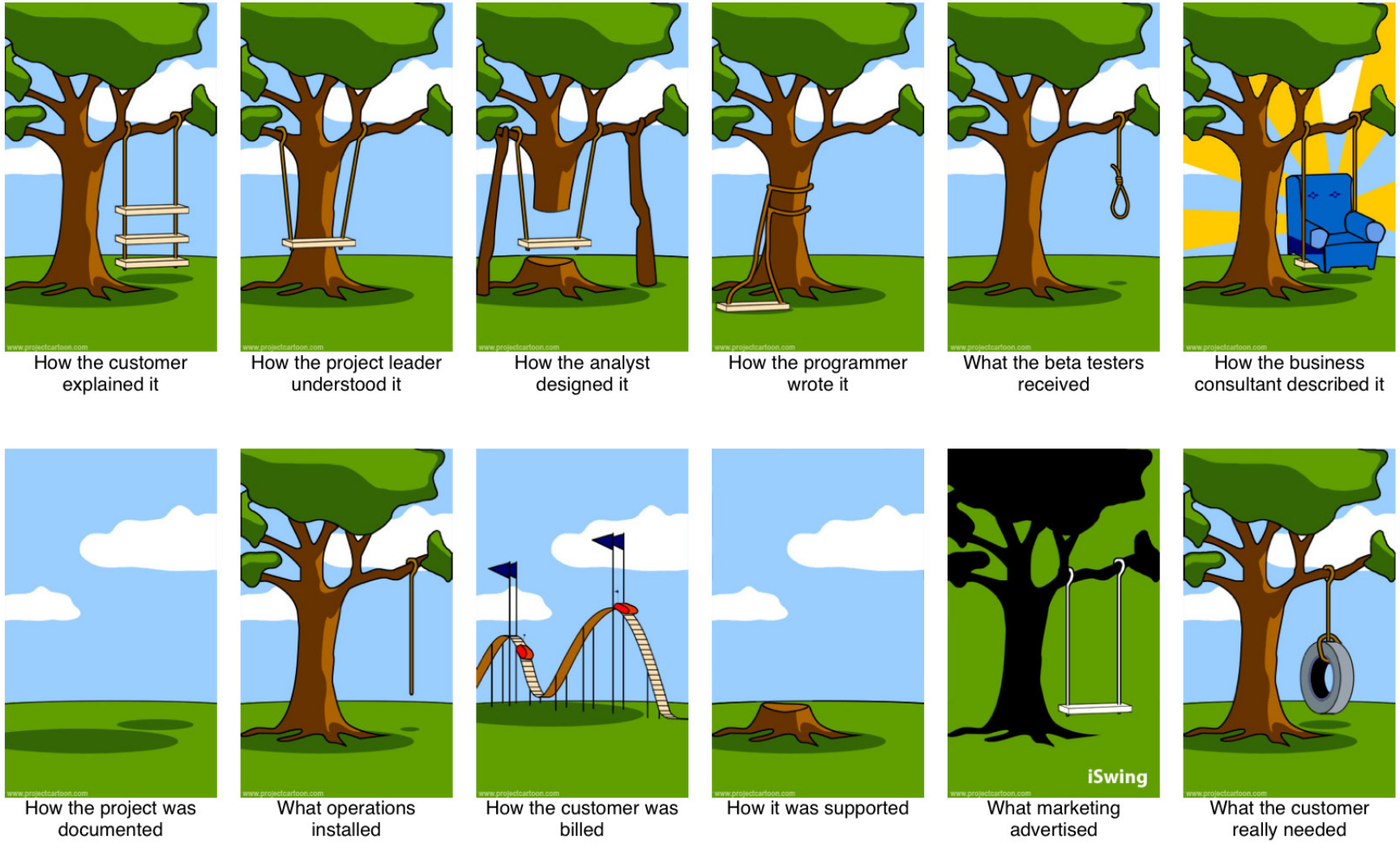
INTERNAL – SAP and Customers Only

Agenda

- Introduction
- Understanding Design-led Development Process
- Basic Considerations – Always keep them in mind!
- Do’s and Don’ts
- 9 Golden Rules for Fiori Development
- UI5 Coding Guidelines

Understanding the Design-led development process

Do you know your user's needs?

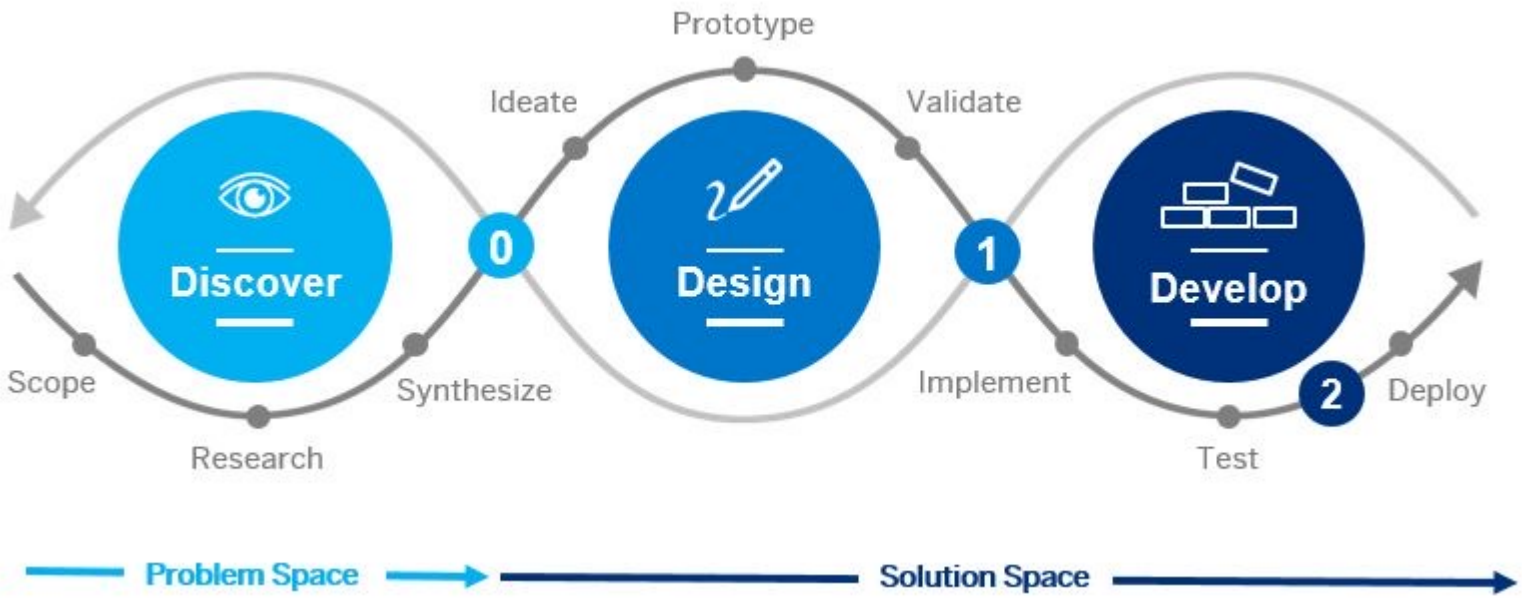


Where to start?

USER RESEARCH?

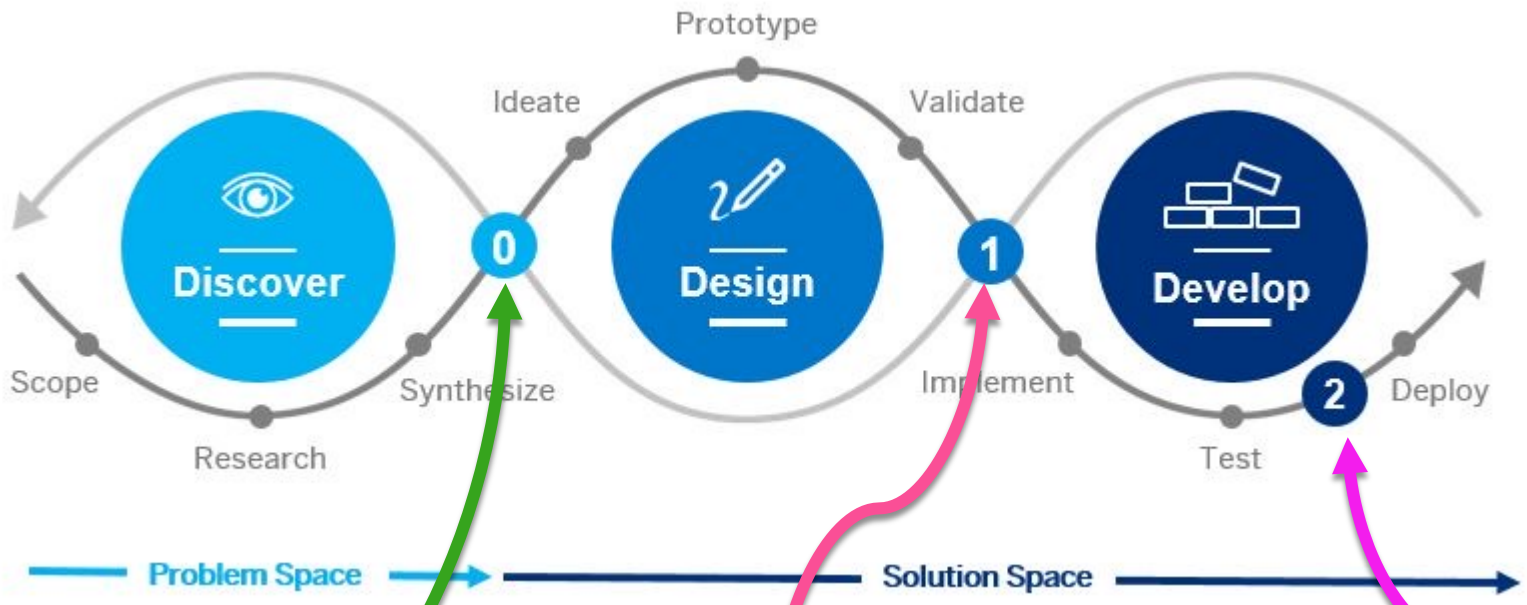
- ✓ YES
- ○ NO
- ○ I‘M AN EXPERT

Design-led Development Overview



The design-led development process spans across the entire development life cycle and consists of **three main phases** and their corresponding design gates. Of course, iterations are part of the approach and strongly recommended.

Design-led Development Overview



Do we know what we need to deliver?

Do we have the right design for the user?

Have we developed according to the design?

Introduction to Important Fiori and SAPUI5 Concepts

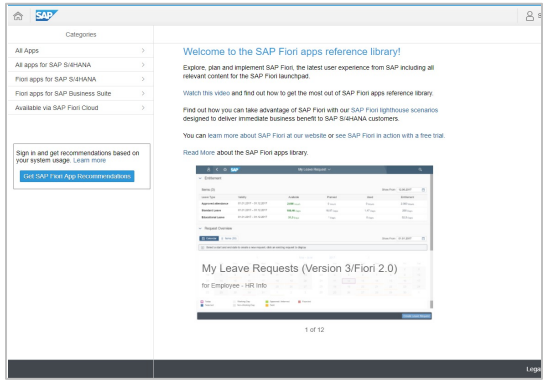
SAP Fiori UI technologies

There are several paths to the SAP Fiori user experience (UX)

Use or adapt SAP-delivered apps

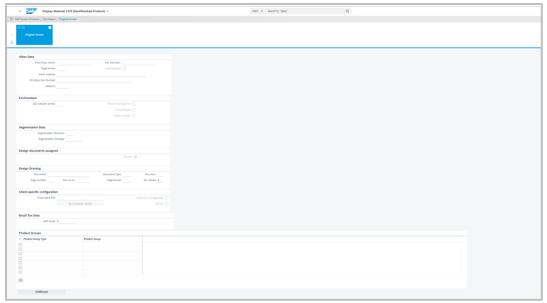
SAP Fiori apps

Out-of-the-box solutions developed by SAP



SAP Screen Personas

Personalized and simplified SAP GUI screens



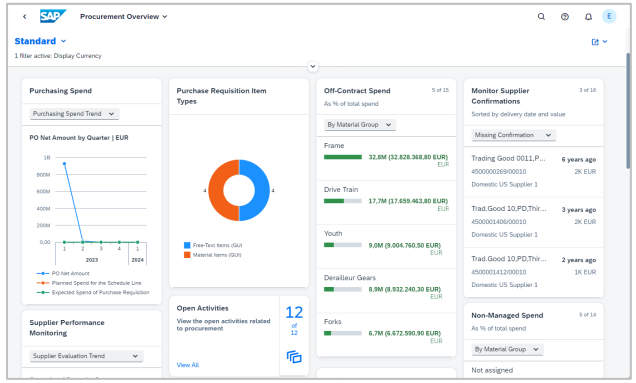
Develop new apps

SAP Fiori elements

Templates for low-code development with SAPUI5

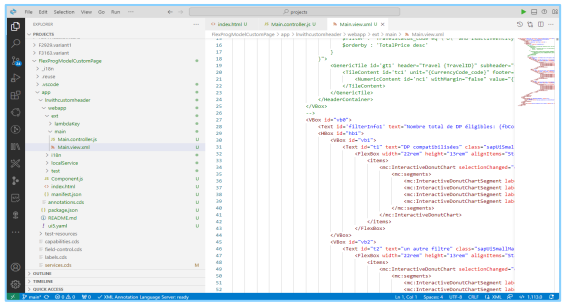
SAP Fiori tools

Productivity tools to make developers more efficient



Freestyle SAPUI5 or native mobile apps

Custom apps developed by you or an SAP partner

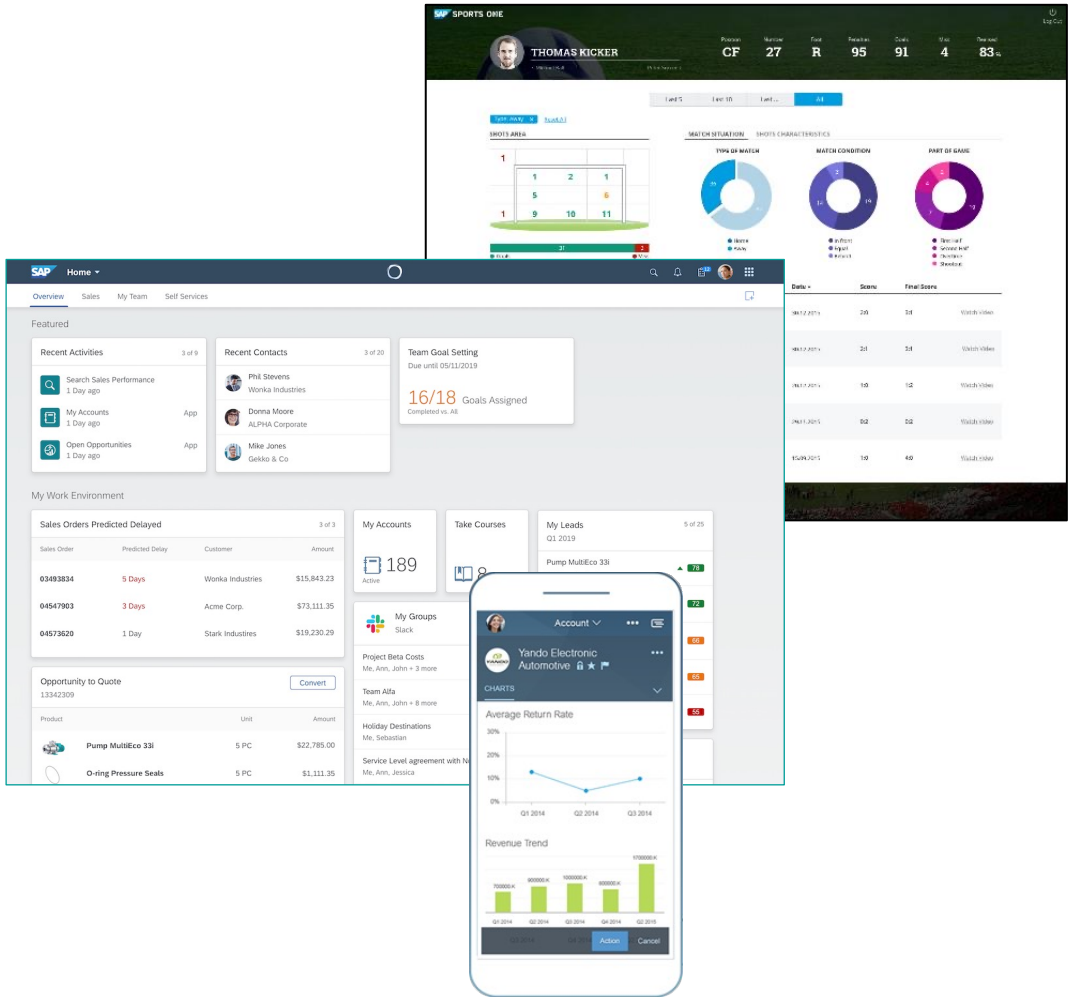


Building Modern Web Applications with SAPUI5

SAPUI5 is an HTML5 framework for creating cross-platform, enterprise-grade applications

As the reference UI technology for the SAP Fiori user experience, SAPUI5 provides:

- A consistent SAP Fiori user experience across SAP solutions (and beyond)
- Efficient development for Web applications
- Easy integration into SAP's single point of entry
- Enterprise qualities like accessibility
- Generic adaptation capabilities that provide UI flexibility



Building Modern Web Applications with SAPUI5

UI5 Web Components deliver SAP's user experience out-of-the-box

UI5 Web Components deliver SAP Fiori 3 design, enterprise-grade stability, accessibility, internationalization, and theming:

- Share fundamental UI5 qualities
- Provide easy consumption of UI5 elements
- Consume with a small footprint
- Enable Web developers

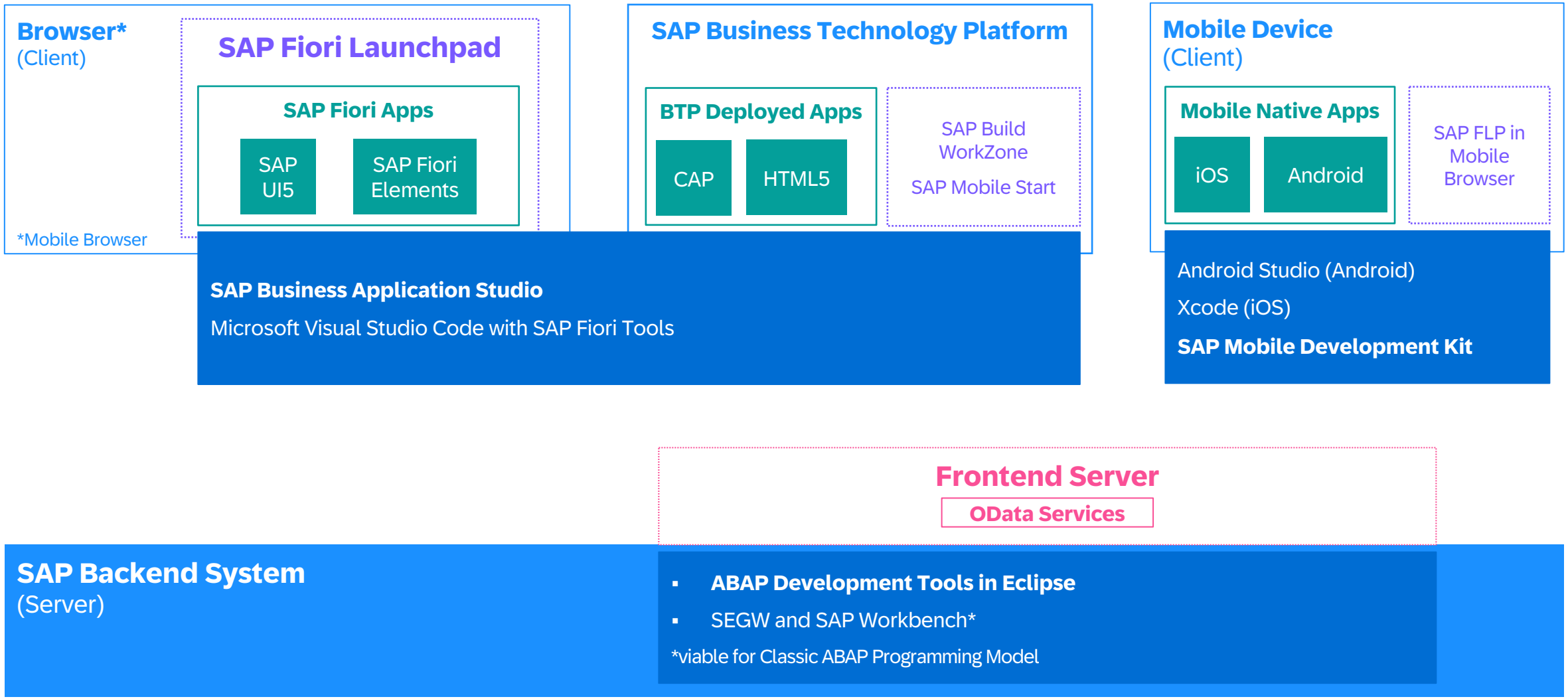
UI5 Web Components provide UI5 elements as reusable Web components with:

- **HTML** structure and content
- **CSS** for the current theme and internationalization
- **JavaScript** to handle user interaction (behavior)

Applications implement UI5 Web Components via ubiquitous HTML:

```
<ui5-button id="myButton"> Click here! </ui5-button>
```

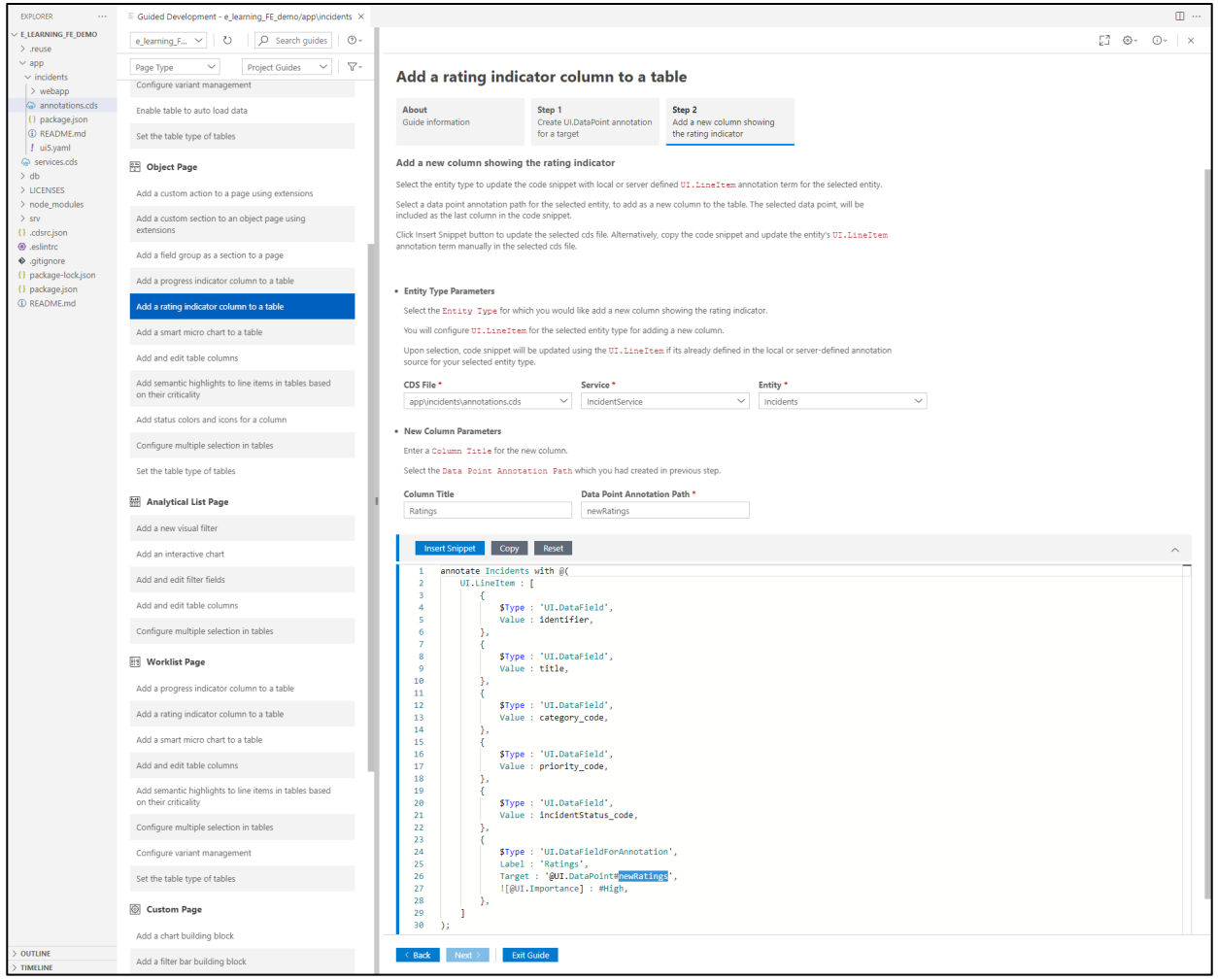
SAP Fiori runtime architecture with SAP-specific Technologies (on Premise) and SAP Business Technology Platform



SAP Fiori tools makes it easier to build SAP Fiori elements apps

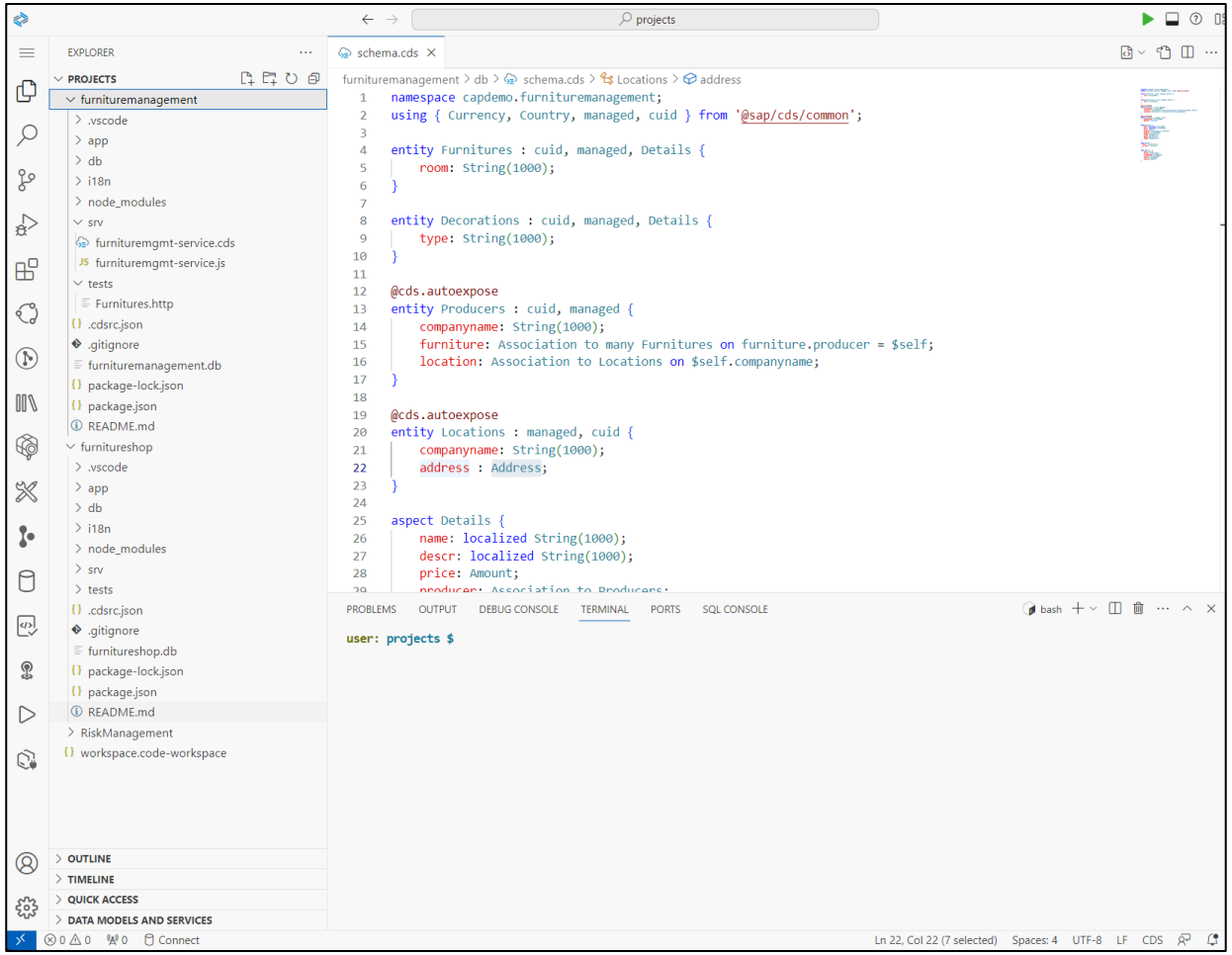
- Uses a **wizard-style** approach
- Ensures **consistent** app structure
- Provides **step-by-step** development instructions
- **Reduces effort** and skill level required for creating annotations
- Improves **code consistency**
- **Simplifies** app maintenance

Available in
SAP Business Application Studio
and **Visual Studio Code**



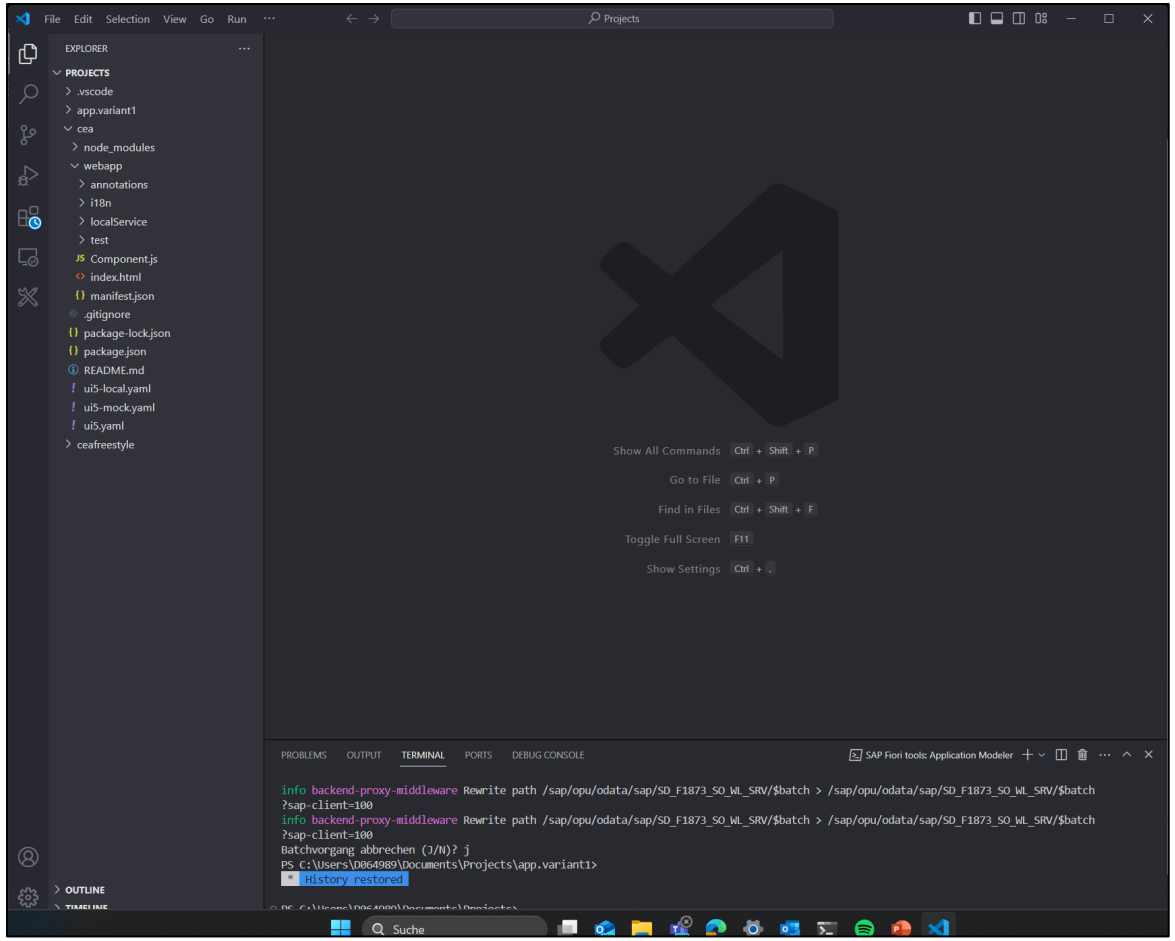
SAP Business Application Studio (cloud based)

- Provides a modern, maintenance-free, preconfigured, **hosted environment***, optimized for SAP developers
- Can be **centrally administrated** with tools repositories, systems access and company policies
- Integrates with existing SAP solutions, systems & services
- Easy access to Visual Studio Code-compatible extensions from open source Theia marketplace **



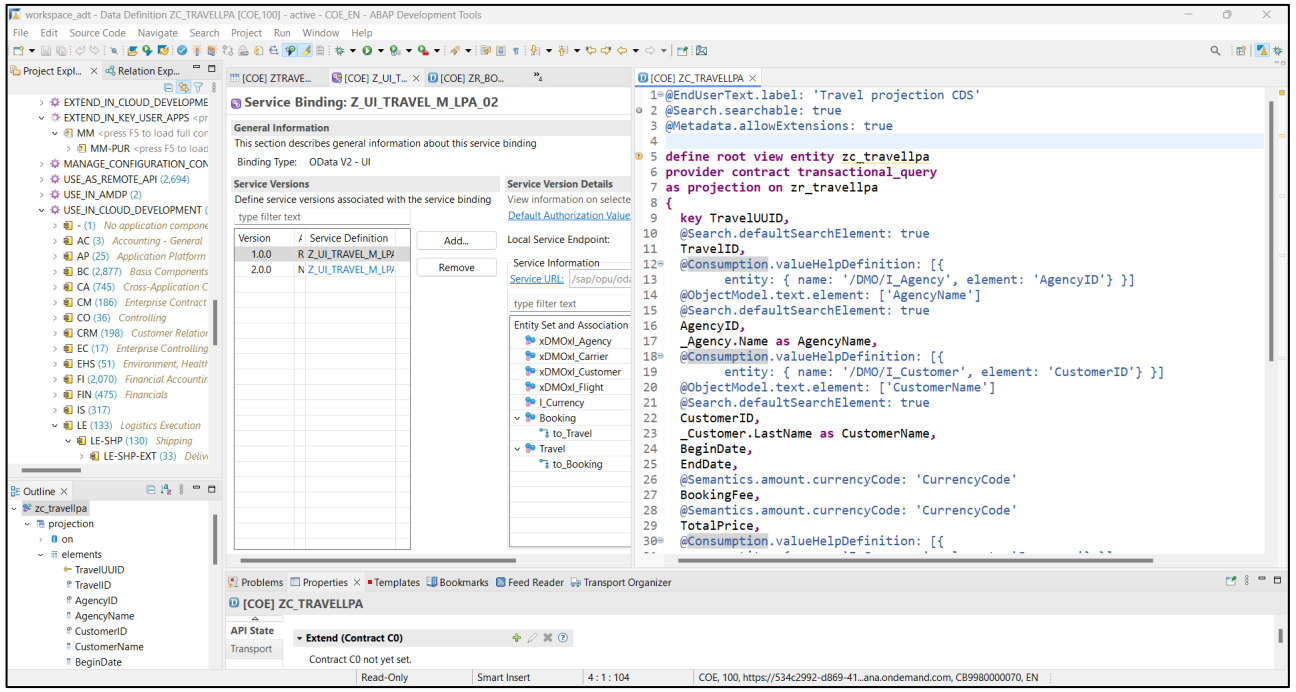
Microsoft Visual Studio Code (not-cloud based)

- Provides development flexibility with **offline** capabilities
- Features a **user-controlled**, highly configurable development environment
- Allows access to thousands of extensions from the Visual Studio Code marketplace
- Is part of a fast growing open source community



ABAP Development Tools on Eclipse

- Empowers SAP's state-of-the-art development concepts
- Enables developers to build, extend and run ABAP applications based on SAP Products
- Provides a rich toolset to support ABAP CDS View and RESTful ABAP Programming Model development
- Smooth and easy migration from the SAP GUI Workbench (SE80)



Bootstrapping SAPUI5

To use SAPUI5, you must load and initialize the SAPUI5 library. The bootstrapping is configured in **index.html** file from a SAPUI5 project.

The figure in the right shows a typical bootstrap script tag used to load the sap-ui-core.js file. When this script tag is included in a page, the SAPUI5 runtime will automatically be initialized as soon as the script is loaded and executed by the browser.

SAPUI5 provides several bootstrap files for different use cases. **sap-ui-core.js** is the standard bootstrap file, which is recommended to use for typical use cases. It already contains jQuery, jquery-ui-position and only the minimum required parts of the core library (sap.ui.core).

```
<script
    id="sap-ui-bootstrap"
    src="resources/sap-ui-core.js"
    data-sap-ui-theme="sap_fiori_3"
    data-sap-ui-libs="sap.m"
    data-sap-ui-compatVersion="edge"
></script>
```

Bootstrapping SAPUI5

The SAPUI5 runtime can be configured using so-called configuration options. Below we will see the most used:

- To define the Theme used in the application, **data-sap-ui-theme** must be configured
- The configuration option **libs** defined a list of libraries that shall be loaded automatically. A SAPUI5 library bundles a set of controls and related types
- To ensure that the latest SAPUI5 features are used, data-sap-ui-compatVersion must always be set to **edge**. Other version definitions are deprecated

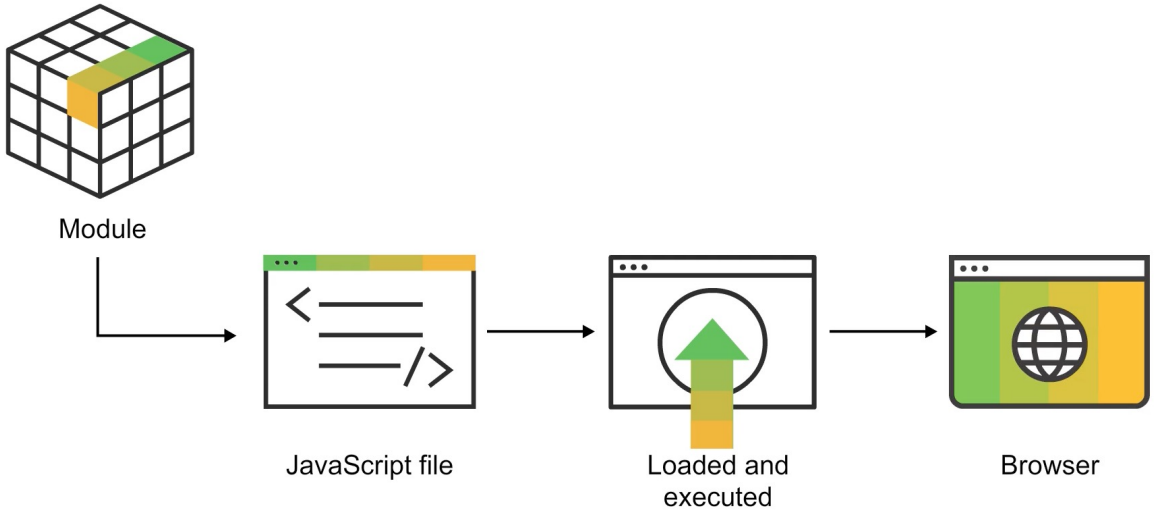
```
<script
    id="sap-ui-bootstrap"
    src="resources/sap-ui-core.js"
    data-sap-ui-theme="sap_fiori_3"
    data-sap-ui-libs="sap.m"
    data-sap-ui-compatVersion="edge"
></script>
```

Defining and Using Modules

The SAPUI5 framework provides built-in support for application modularization. That is, instead of defining and loading a large bundle of JavaScript code, an application can be split into smaller parts that can then be loaded at runtime when they are needed.

SAPUI5 module is defined in file index.js, that should be located inside webapp folder, by calling the **sap.ui.define** method. The implementation made via sap.ui.define determines the so-called export value of a module.

This export value is typically a JavaScript object that represents the value of a module from a usage perspective. Usually, the export value is determined via a function.



index.js

webapp > index.js > ...

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

    new Text({ text: "Hello World" }).placeAt("content");

});
```

Loading of Modules

In the bootstrap script shown in the figure, the default bootstrap file sap-ui-core.js is loaded from the resources folder. By default, SAPUI5 therefore **loads all modules** relative to this folder.

For modules like sap.m there are so-called library preload files named library-preload.js. These files contain all modules of the respective library. Adding the attribute **data-sap-ui-async** enables the module loader to load single modules as well as library preload files asynchronously.

The attribute **data-sap-ui-onInit** can be used to define code that is executed after initialization.

Using the **resourceRoots** configuration option, it is possible to redirect requests from application-specific modules to corresponding Web application.

```
<script
    id="sap-ui-bootstrap"
    src="resources/sap-ui-core.js"
    data-sap-ui-theme="sap_fiori_3"
    data-sap-ui-libs="sap.m"
    data-sap-ui-compatVersion="edge"
    data-sap-ui-async="true"
    data-sap-ui-onInit="module:sap/training/exc/index"
    data-sap-ui-resourceroots='{
        "sap.training.exc": "./"
    }'
></script>
```

Model View Controller Concept

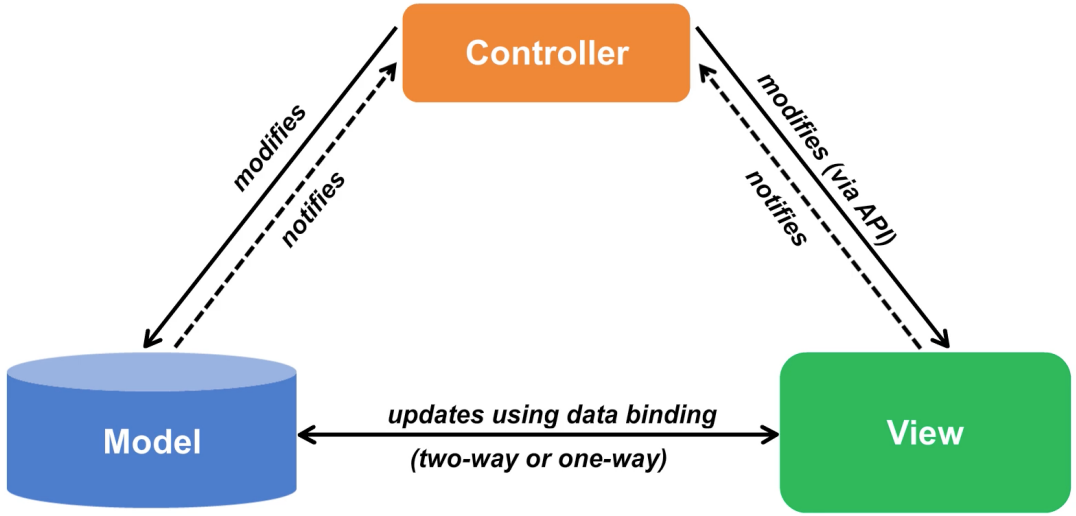
The Model View Controller (MVC) concept is used in SAPUI5 to separate the presentation of information from user interaction. This separation facilitates the development and modification of an application.

Model contains the application data and provide methods for reading and changing the data

View is responsible for defining and rendering the UI. Views are linked to Models via data binding. In the One-way binding, data is only displayed in the UI, while in Two-way binding the data can be written back in the Model

Controller defines how model and views interacts. Views and Controllers often forms an 1:1 relationship

It's possible to have Controllers without Views. They are called **application Controllers**



Views

There are three types of View:

- XML View: declarative views in which the user interface is defined in a file or string in XML format. It is recommended to use XML views because XML views enforce a clear separation of the UI definition from the application logic
- JSON View: declarative views in which the user interface is defined in a file or string in JSON format
- Typed View: user interface is defined in a programmatic manner using JavaScript, so it is defined using a separate view class

View Instantiation:

To instantiate an XML View, SAPUI5 provides the factory method **create** in the class sap.ui.core.mvc.XMLView

The create method loads view asynchronously via the module system. The advantage of this type of loading is that the UI does not freeze for the duration of the loading process and the functionalities are not blocked during view instantiation

Therefore, on the returned promise the **then** function is triggered to pass the view instance via the interface oView

index.js

webapp > index.js > ...

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

    XMLView.create({
        id: "App",
        viewName: "sap.training.exc.view.App"
    }).then(function (oView) {
        oView.placeAt("content");
    });

});
```

index.js

webapp > index.js > ...

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

    XMLView.create({
        id: "App",
        viewName: "sap.training.exc.view.App"
    }).then(function (oView) {
        oView.placeAt("content");
    });

});
```

Controllers

Assigning a View Controller

To assign a controller to an XML view, the attribute controllerName can be used in the <View> tag (see the figure Referencing a Controller). The name of the controller class is entered as the value of the attribute.

App.view.xml

webapp > view > App.view.xml

```xml
<mvc:View controllerName="sap.training.exc.controller.App"
    xmlns:mvc="sap.ui.core.mvc"
    xmlns="sap.m">

    <Button text="Say Hello" press=".onSayHello"/>

</mvc:View>
```

Controller Implementation

Controllers should always be created under webapp/controller folder

Controllers are implemented as separate modules via sap.ui.define and are usually created as subclasses of sap.ui.core.mvc.Controller

App.controller.js

webapp > controller > App.controller.js > ...

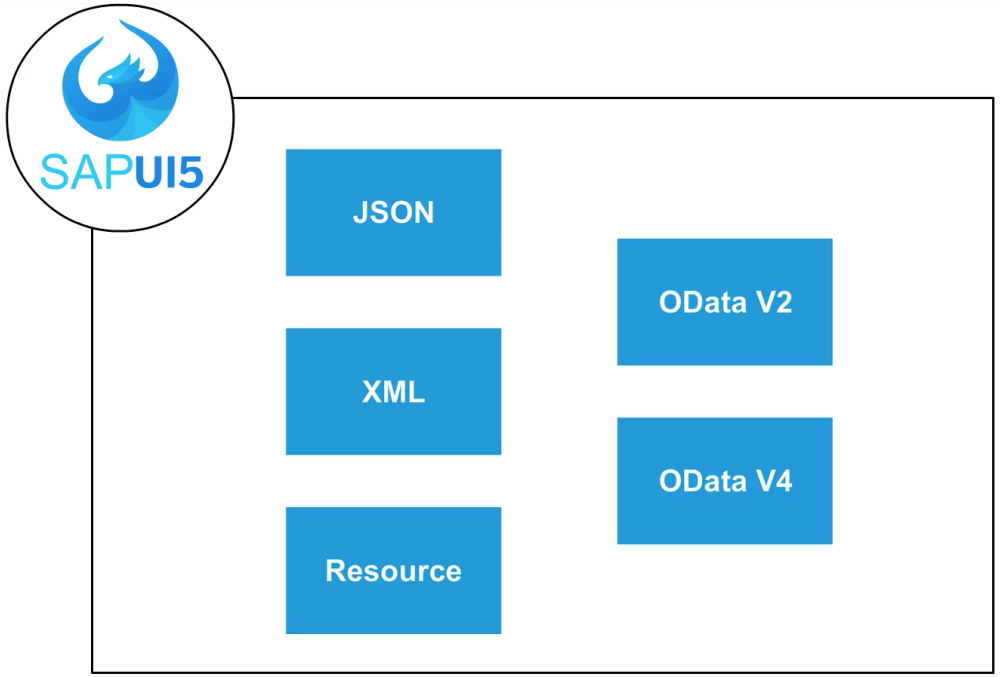
```js
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/m/MessageBox"
],
    function (Controller, MessageBox) {
        "use strict";

        return Controller.extend("sap.training.exc.controller.App", {

            onSayHello: function () {
                MessageBox.information("Hello World");
            }

        });
    });
```

Models



A model in the Model View Controller concept holds the data and provides methods to retrieve the data and to set and update data.

OData Model enables binding of controls to OData Services. It currently supports OData V2 and V4

JSON Model can be used to bind controls to JavaScript object data, following JSON format when instantiating the model

XML Model can be used to bind controls to JavaScript object data, following XML format when instantiating the model

Resource is designed for providing texts in different languages

JSON, XML and Resource models are **Client-side** models, meaning that the model data is loaded and is available on the client. Sorting and filtering are executed on the client.

OData models are **Server-side** models, and only loads the data requested from server

Models
Instantiating a JSON Model

The JSON model is implemented using the sap.ui.model.json.JSONModel class.

The figure, shows how to instantiate a JSON model in the **onInit** initialization method of a view controller.

After the instance is created, there are several ways to get data into the model. In addition to data binding, data can also be set and changed in the model via API:

- The setData method sets the data passed as JS Object tree to the model
- The loadData method loads JSON-encoded data from the passed URL using a GET HTTP request and stores the resulting JSON data in the model
- The getData method, on the other hand, returns the current data of the model.

Overview.controller.js

webapp > controller > Overview.controller.js > ...

```
1  sap.ui.define([
2      "sap/ui/core/mvc/Controller",
3      "sap/ui/model/json/JSONModel"
4  ],
5      /**
6       * @param {typeof sap.ui.core.mvc.Controller} Controller
7       */
8      function (Controller, JSONModel) {
9          "use strict";
10
11         return Controller.extend("sap.training.exc.controller.Overview", {
12
13             onInit: function () {
14                 var oModel = new JSONModel();
15
16                 ...
17             }
18
19         });
20     });
```

Models
Assigning the Model to the UI

You must assign a model instance to the UI before you can bind controls to it.

For this purpose, you can use the **setModel** method, which is available on every control.

- The first parameter passed to this method is the model to be set
- The second (optional) parameter can be used to define a name for the model.
- If the name is omitted, this will set the so-called default model without a name.

When you specify a model for a section or control, it is propagated to all aggregated child controls. So, if you set a model for a container control, all controls contained (aggregated) in that container will have access to that model.

Overview.controller.js

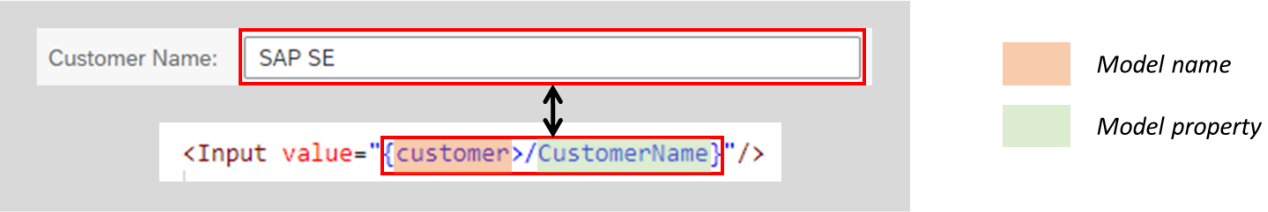
webapp > controller > Overview.controller.js > ...

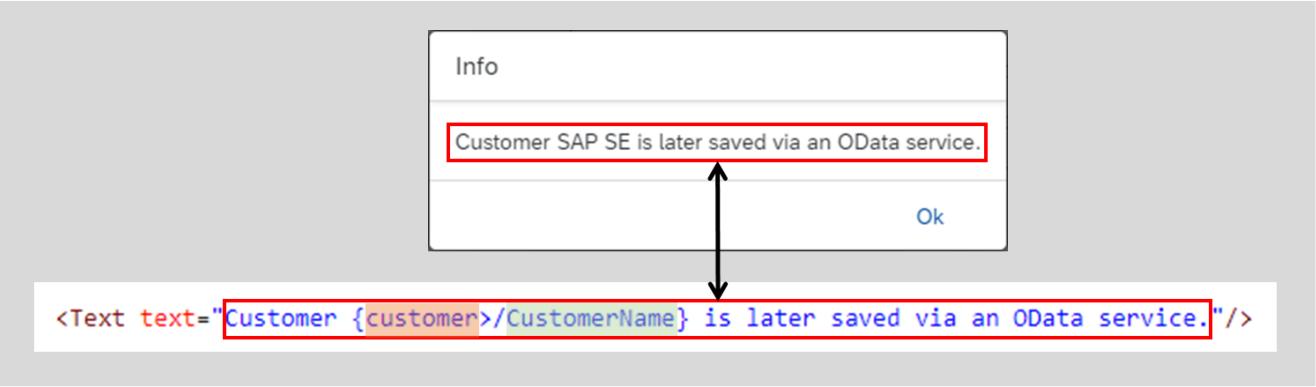
```
1  sap.ui.define([
2      "sap/ui/core/mvc/Controller",
3      "sap/ui/model/json/JSONModel"
4  ],
5      /**
6       * @param {typeof sap.ui.core.mvc.Controller} Controller
7       */
8      function (Controller, JSONModel) {
9          "use strict";
10
11         return Controller.extend("sap.training.exc.controller.Overview", {
12
13             onInit: function () {
14                 var oModel = new JSONModel();
15
16                 var oView = this.getView();
17                 oView.setModel(oModel, "customer");
18             }
19
20         });
21     });
```

Models
Data Binding

To bind controls to a model, various so-called binding types are available.

In the example shown in the figure Binding Syntax, property binding is used. Property binding makes it possible to initialize and update the properties of a control automatically based on the model data.





Models
Data Binding

The binding mode defines how a model is bound to the UI. The two-way binding mode was already discussed, which is used by default for the JSON model.

SAPUI5 provides the following binding modes:

- One-way
- Two-way
- One-time

You can overwrite the default binding mode after model creation. To change the default binding mode, call the **setDefaultBindingMode** method on the mode.

Model	One-time	One-way	Two-way
JSON Model	✓	✓	✓
XML Model	✓	✓	✓
Resource Model	✓	✓	✗
OData V2 Model	✓	✓	✓
OData V4 Model	✓	✓	✓

Default Binding Mode

Basic Considerations
Always keep them in mind

Basic Considerations – Always keep them in mind

1. Fiori apps do not run in their own container, javascript files and styles are loaded into the central web-page of the unified shell. Be aware that some screen space is occupied by the shell – ***This may affect your design!***

2. Only the visual **instances** are destroyed, code and styles remain, global statics remain. Thus, with multiple apps running one has to take care to associate one's data with a Javascript instance associated with your application, not with a "global name".

3. One also has to avoid cluttering global memory or namespaces with app-specific information. Thus it is crucial that the application correctly registers and unregisters all app-specific information on global static framework constants, even in the case of crashes and or forced unloading by the container. An effective method to achieve this is: ***Don't do it in the first place!***

Do’s and Don’ts

DO – Build your application as a self-contained SAPUI5 component

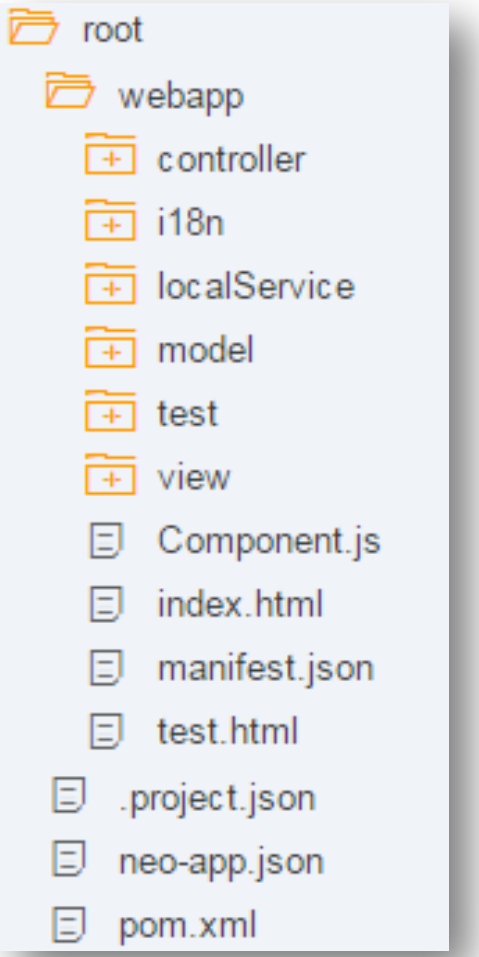
Build your application as self-contained SAPUI5 component using the SAPUI5 Mobile Library and following the Fiori UX guidelines (see Fiori Design).

The Fiori Launchpad is designed as a **single page** SAPUI5 application. This means that at runtime only the *FioriLaunchpad.html* page is loaded. Since Fiori applications are implemented with the same UI technology, it is possible to embed them directly into the launchpad page via DOM injection. This approach allows smooth, animated UI transitions and reuse of shared components at runtime.

The shell utilizes a so-called **Application Container** for embedding a component. The container is configured with

- the URL (root path) of the application - this is the path where the component descriptor file (*Component.js*) is located
- the name of the SAPUI5 component

The container registers the component namespace as module path for the application URL.



Folder Structure – Where to put your files?

DO – Declare configuration information in manifest.json

Declare configuration information (e.g. icons) and library dependencies in the manifest.

The Fiori Launchpad shell renderer evaluates parts of this configuration and uses it in several scenarios

The shell evaluates the following properties of the component configuration:

- "**icon**": URI to the app's icon, typically an SAPUI5 icon URI using the "Fiori" icon collection
- "**favIcon**": Reference to the "favicon" (*.ico) file for the app, which is typically displayed in the address bar or window/tab title of the web browser
- "**Phone**", "**Phone@2**", **"Tablet", "Tablet@2"** : References to icons with different resolutions that are used when a user adds the app to her mobile device's home screen
- "**version**": app version displayed in the about dialog.

If you've developed an app using the Component Configuration approach we recommend switching to the new Manifest.json → [Creating a Descriptor File for Existing Apps]

Component Configuration

```
sap.ui.core.UIComponent.extend("sap.abc.Component", {
    metadata : {
        name: "Master Detail Sample",
        version : "1.0.0",
        library : "sap.abc",
        includes : [
        ],
        dependencies : {
            (...)
        },
        config : {
            "resourceBundle" : "i18n/i18n.properties",
            "titleResource" : "shellTitle",
            // Refers to an icon registered with UI5. In this case, it's an icon named "F0000" (i.e the app ID
            // from the Fiori 2 icon font (registered with UI5 as icon collection "Fiori2").
            "icon" : "sap-icon://Fiori2/F0002",
            // The following properties reference dedicated image files. Note that relative links are
            // always relative to the location of the Component.js of the app, NOT to the location of
            // the HTML file that is displayed in the browser (typically: FioriLaunchpad.html).
            // Typically, the library (sap.ca.ui) containing the icons is mapped into the "resources"
            // folder of each app automatically, so the following examples for icons always start in
            // that folder.
            "favIcon" : "./resources/sap/ca/ui/themes/base/img/favicon/F0002_My_Accounts.ico",
            "homeScreenIconPhone" : "./resources/sap/ca/ui/themes/base/img/launchicon/F0002_My_Accounts/57_iPh
            "homeScreenIconPhone@2" : "./resources/sap/ca/ui/themes/base/img/launchicon/F0002_My_Accounts/114_
            "homeScreenIconTablet" : "./resources/sap/ca/ui/themes/base/img/launchicon/F0002_My_Accounts/72_iP
            "homeScreenIconTablet@2" : "./resources/sap/ca/ui/themes/base/img/launchicon/F0002_My_Accounts/144
        }
    },
    (...)
});
```

Manifest.json

```
"sap.ui": { //general ui attributes - in sap.ui namespace

    "technology": "UI5" //mandatory; app type such as UI5, WDA, NWBC, URL, GUI (new)
    "icons": { //name/value pairs with defined attribute names for the usage, formerly stored under "config"
        "icon": "sap-icon://add-contact",
        "favIcon": "icon/F1373_Approve_Purchase_Orders.ico", //the ICO file to be used inside the browser and for desktop shortcuts
        "phone": "icon/launchicon/57_iPhone_Desktop_Launch.png", //57x57 pixel version for non-retina iPhones
        "phone@2": "icon/launchicon/114_iPhone-Retina_Web_Clip.png", //114x114 pixel version for retina iPhones
        "tablet": "icon/launchicon/72_iPad_Desktop_Launch.png", //72x72 pixel version for non-retina iPads
        "tablet@2": "icon/launchicon/144_iPad_Retina_Web_Clip.png" //144x144 pixel version for retina iPads
    },
    "deviceTypes": { //mandatory; needed for docu
        "desktop": true, //mandatory; true, false
        "tablet": true, //mandatory; true, false
        "phone": false //mandatory; true, false
    },
    "supportedThemes": [ //mandatory; only SAP themes, needed for docu and mobile packaging
        "sap_hcb", "sap_bluecrystal"
    ],
    "fullWidth": true //to indicate whether app should run in full screen mode (formerly under sap.ui5/config), true, false
},
```

DO – Use startup parameters to pass information

Use startup parameters to pass information to the next application (do not use deep links into the Application-specific hash!). Only startup parameters can be remapped to a different application and allow the flexible configuration of Intent to application required for device specific resolution, extensibility and customer manageability.

```
var href_For_Product_display = ( sap.ushell && sap.ushell.Container &&
sap.ushell.Container.getService("CrossApplicationNavigation").hrefForExternal({
  target : { semanticObject : "Product", action : "display" },
  params : { "ProductID" : "102343333", SupplierId : "90210" }
})) || "";
```

Passing Startup Parameters Dynamically with JavaScript

```
sap.ui.core.UIComponent.extend("AppNavSample.Component", {
  ...
  createContent : function() {
    /*
     * getComponentData() returns an object like { startupParameters : { AAA : ["BBB"], DEF: ["HIJ","KLM"] }
     * NOTE: parameters values are passed via arrays
    */
    var oComponentData = this.getComponentData();
    jQuery.sap.log("app was started with parameters " +
JSON.stringify(oComponentData.startupParameters || {} ));
  ...
  }
});
```

Receiving Startup Parameters Dynamically with JavaScript

Handling Startup Parameters

DON'T – Use Application.js, Global Singletons, sap.ui.core.routing.history, global event bus

- **Do not use Application.js**
  - **Deprecated since UI5 1.15.1 https://sapui5.netweaver.ondemand.com/sdk/#/api/sap.ui.app.Application/overview**
- **Do not use global singletons**
  - **This is not a good example: https://blogs.sap.com/2015/12/29/ui5-classes-and-objects-singleton-pattern/**
- **Do not use sap.ui.core.routing.history.**
  - It is fundamentally broken. Simply call browser.back, window.history.back() or CrossApplicationNavigation.backToPreviousApp
- **Do not use the global event bus sap.ui.getCore().getEventBus().subscribe( )**
  - Use the navigation API to navigate between views.

DON’T – Use sap.ui.getCore(), Global Static variables, sap.ui.localResources or jQuery.sap.registerModulePath

- **Do not use sap.ui.getCore() to register models**
  - Do not use sap.ui.getCore() to register models. Rather register models on the root view or root component of your application
- **Avoid global statics variables**
  - Do not use global statics to pass data, these are error prone and may cause name collisions, garbage collection /aka performance issues and distribution of state among several instances of your application.
- **Do not use "sap.ui.localResources" or "jQuery.sap.registerModulePath”**
  - Do not use "sap.ui.localResources" or "jQuery.sap.registerModulePath" inside your Component.js file. Declare your dependencies in the manifest file.
  - "sap.ui.localResources" will register a path relative to the main page (FioriLaunchpad.html). Components must not make implicit assumptions on this location.
  - "jQuery.sap.registerModulePath" will create a platform dependency and should be avoided if possible.

DON'T – Read or write to window.location.hash, deep link into hashes, make your urls too long

- **Do not read or write directly to window.location.hash window.location etc. when dealing with navigation hashes**
  - Use only the UI5 and shell navigation services (Cross Application Navigation) to trigger a navigation event.
  - There are subtle encoding issues involved and the interfaces take care of it in a consistent manner.
- **Do not deep link into application specific hashes of your target app**
  - Use only the intent and startup parameters to pass state to a target app.
- **Assure the url constructed by your application does not exceed reasonable limits.**
  - The URL length is restricted in browser platform. E.g. in IE 2083 characters are a fixed limit. Be aware that the URL is composed of several different parts, only some are under application control:
    - Server, port, path
    - Launchpad url parameters, e.g. client, system, theme, debug settings
    - Shell hash, startup parameters (restricted to around 512 characters).
    - Application Specific hash
  - If parts of this string are constructed by your application, you have to assure not to exceed reasonable limits even under the most adverse administrator configurations, user actions and data. It is reasonable to restrict the startup parameter length to 512, the application hash to 512 characters. This leaves roughly 1000 characters for the launchpad url, including semantic object and action .

DON’T – Use multivalued Parameters, deploy local files, make assumptions on the Intent, put sensitive data into the URL

- **Do not use multivalued Parameters**
  - According to unified shell architecture, multivalued arguments violate the intent of the Fiori Launchpad intent, which should be composed of single valued parameters. Thus the following URL is not ok:
  - *SemanticObject-Action?A=Value1&A=Value2&A=Value3*

- **Don't deploy your noShellIndex.html or localIndex.html to the production system**
  - The start pages contained in the application templates used by the creation wizards are only intended for developer tests. For productive use cases, dedicated start pages have to be built which bootstrap the unified shell for the target platform.

- **Do not make assumptions on the Intent (Semantic Object - action) your application is started with.**
  - Conceptually the Intent is distinct from the physical application and may be reconfigured by the Customer. Your actual application may be started with a different action than envisioned.

- **Do not put sensitive data into the URL**
  - Instead, store the data on your server and pass an ID.

9 Golden Rules for Fiori Development

9 Golden Rules for Fiori Development

- ✓ SAP Fiori apps must have an approved UX design
- ✓ SAP Fiori UIs are built with SAPUI5
- ✓ SAP Fiori apps are based on OData services
- ✓ One SAP Fiori app - one OData service
- ✓ Every SAP Fiori app is defined by a set of metadata
- ✓ No custom CSS is allowed for SAP Fiori apps
- ✓ Every SAP Fiori app must run as a web app
- ✓ Every SAP Fiori app must run in the SAP Fiori Launchpad
- ✓ Every SAP Fiori app must run on mobile devices along the native app paradigm

SAP Fiori apps must have an approved UX design

- **The Design-led Development Process is valid for all UI scenarios which have to follow the SAP Fiori design principles.**
- **Designs have to comply with the Fiori Design Guidelines.**
- Prefer designs that make use of existing controls and patterns.
  - Available controls and patterns can be found on the Fiori Design Guidelines.
  - Live examples of the available controls can be found in the UI5 Explored App.
  - Make use of Smart Templates and Smart Controls where possible to reduce development effort (increase extensibility). See also the Smart Templates section in the UI5 Development Toolkit.
  - If new control implementation is required consider regression testing and additional efforts to release your application.

SAP Fiori UIs are built with SAPUI5

- **New Fiori applications should be built using the SAPUI5 UI Technology.**
- **The recommended approach for standard S/4HANA applications is to make use of Smart Templates, that reduce the UI coding (Java Script, ViewXML) to the bare minimum.**
- **For pixel perfect applications that do not follow a pattern-like approach, UI5 freestyle in combination with XMLViews is the recommended way. If the backend supports annotations, the usage of Smart Controls is strongly recommended also inside freestyle Fiori applications.**
- **For pixel perfect applications that only have minor deviations from the Smart Template applications, a combination is recommended:**
  - Start by using Smart Templates and execute/unlock the templates at design time and continue to manually write code.

SAP Fiori apps are based on OData services.

- **Fiori applications access backend data via the OData protocol.**
- **OData services are easy-to-consume RESTful services.**
  - Each service is based on and described by its individual entity-relationship model.
  - A machine-readable representation of this entity-relationship model is part of each OData service.
  - The structural standard metadata can be enriched and extended with additional metadata describing the semantics of the services data, the service behavior and interaction capabilities, and even hints on how to present the data to human users.
- **This machine-readable metadata is leveraged by the UI5 OData Model, the UI5 Smart Controls and Smart Templates.**

One SAP Fiori app - one dedicated OData service

- **Fiori applications access backend data via the OData protocol.**
  - Each OData service is based on and described by its individual entity-relationship model. This model is tailored to the data needs of the Fiori application so that it allows access to all the data needed by that application, and only the data needed by that application.
- **This apparent restriction simplifies the life-cycle of a Fiori application and its data access service in several ways:**
  - authorization for only one service needs to be granted to a user in order to use the application.
  - only one service needs to be activated in order to operate the application.
  - only one service needs to be upgraded in sync with the application.
  - only one application needs to be upgraded in sync with the service.
- **This rule is not a technical restriction: an app must be able to use several OData services indirectly via reusable app parts, or via dynamic code or data binding that is at run-time derived from metadata.**

Every SAP Fiori app is defined by a set of metadata.

- **Fiori applications define their set of metadata in an app descriptor (manifest.json) file.**
- **The purpose of the app descriptor is to ease the orchestration and automation of the development, delivery, deployment and operational aspects of an app.**
- **The focus is on creating an experience, where the app is perceived as one single entity, even if its content and services are spread over the system landscape and different repositories.**
- **The information in the app descriptor can be evaluated for several features and functions.**
- **For apps developed with UI5 >= 1.30 the app descriptor is mandatory.**

No custom CSS is allowed for SAP Fiori apps.

- **The HTML/CSS generated by UI5 is not part of the public API and may change in patch and minor releases.**
- **UI5 is a shared resource of all applications. As such overriding UI5 with CSS in Fiori Apps bears the risk of breaking your application on updating UI5.**

Every SAP Fiori app must run as a platform independent web app

- **An app must be built in a platform-independent way to be able to run it on ABAP, HCP or as part of a capsuled hybrid application.**
- **Avoid the following most common mistakes:**
  - Absolute URLs generated in front or back-end
  - Hard dependencies between apps
  - Usage of private UI5 properties or functions
  - Calling UI2 services directly instead of the APIs
  - Usage of platform-specific functionality without availability check

Every SAP Fiori app must run in the SAP Fiori launchpad.

- **Fiori Applications must be implemented as self-contained UI5 Components which can be embedded into different launchpages**
  - noShellIndex.html: local, isolated test without shell services
  - FioriSandbox.html: local, integrated test with mock shell services
  - FioriLaunchpad.html: productive usage
- **You have to further obey the rules regarding cross app navigation:**
  - In order to navigate from one app to the other you always have to call sap.ushell.services.CrossApplicationNavigation
  - It is not allowed to use any other way of launching applications.
  - It is not allowed to call absolute URLs directly.
  - Specifically: You should never build and launch absolute URLs to WDA or WebGUI applications within your SAP Fiori App.
- **Enable your App to be a Web App**
  - Every SAP Fiori app should provide a signature. This has to be done via Target Mapping or the AppDescriptor. The AppDescriptor is preferred if available and sufficient.

Every SAP Fiori app must run on mobile devices along the native app paradigm

- **Fiori applications can run on a mobile device within the browser, Fiori Client and as self contained native Packaged Apps.**
- **Mobile qualities are native features of a mobile device that can be leveraged by Fiori applications (e.g. Camera, PIM integration, Location, Offline, etc.).**
  - These are exposed as additional APIs to the Fiori developer that become available when running inside Fiori Client or as Packaged Apps.
  - This means, if Fiori apps want to benefit from the native features of the device they must run inside Fiori Client or deployed as Packaged Apps.
  - For offline enablement these apps must be deployed as Packaged Apps only.

Every SAP Fiori app must run on mobile devices along the native app paradigm

Fiori Client

- **All Fiori application must run within the Fiori Client.**
- **The Fiori Client is available on the Apple App Store, Google Play and is the most generic way Fiori apps can run on a mobile device and still have access to the native features (Mobile qualities) outside of offline enablement.**
- **Developers must test their apps with the Fiori Client and make this part of their testing planning.**
  - Fiori Client is based on Cordova which uses an embedded WebView, and has differences with native browsers of the device (usually around rendering).
  - In other words: one cannot assume that if their app runs smoothly on Safari on an iPhone that it will also run perfectly in Fiori Client.

Every SAP Fiori app must run on mobile devices along the native app paradigm

Packaged Apps

- **Fiori application can be pre-packaged as self contained native applications.**
  - This means all static web resources (javascript, html5, xml, images, etc.) are bundled together into a Cordova container and the apps are delivered to end users using standard app store concepts.
  - The Mobile Platform in SCP provides cloud based tools so that the customers' admin can perform this packaging and deployment to the app store of their choice.
- **The requirement for an app to be "packageable" is to be deployed to a UI Add-on 2.0 SP00 or higher (UI5 1.30 and higher).**
- **The app also requires at minimum an App Descriptor as this would define the intents for the app, and allow packaging multiple apps together and benefit from cross-app navigation within the same native app bundle.**

Every SAP Fiori app must run on mobile devices along the native app paradigm

Mobile Qualities

- **Some Fiori apps may want to benefit from native mobile features and yet still be able to run in a browser/desktop where these features are not available.**
- **It is important for the app developer to always check if the name space of the mobile quality is available before their code starts to use it.**
  - For example, if the BarcodeScanner is not available or disabled by the admin the name space cordova.plugins.barcodeScanner will become null. To use this plugin the user should always check if cordova.plugins.barcodeScanner is not null before using any of its APIs.

Helpful Links

- **SAP Fiori Tools**
  - SAP Fiori Tools | SAP Community
  - SAP Road Map Explorer
- **Setup Visual Studio Code with Fiori Tools**
  - Visual Studio Code | SAP Help Portal
  - Fiori Development with VSCode and NodeJS - SAP Community
    - This blog post also includes many links to other useful tutorials and blog posts concerning how to use VS Code for SAP Fiori Development.
  - SAP Fiori Tools - Extension Pack - Visual Studio Marketplace
- **Getting Started with SAP Business Application Studio**
  - Set Up SAP Business Application Studio for Development | SAP Tutorials
- **Install ABAP Development Tools for Eclipse**
  - Download the Eclipse IDE and add the ABAP Development Tools (ADT) Plugin | SAP Tutorials
  - ABAP Development Tools for Eclipse | SAP Help Portal

Thank you.

Contact information:

Leonardo Britz
Senior UX Consultant
leonardo.britz@sap.com

THE BEST RUN SAP