SAPUI5 & FIORI
IBM Best Practices and Guidelines
2021

Version	**:**	**6.0**
Date Created	**:**	**10-May-2021**
Prepared By	**:**	**IBM CIC India**

Approval and Distribution

Approved by	Name	Role	Signature	Date
Kousik Goon	Kousik Goon/IN/IBM	Product Lead		30/06/2021

Document History

Version	Author	Reason for change	Date
1.0	IBM CIC India	First Draft	03-April-2015
2.0	IBM CIC India	2nd Final	01-Apr-2016
3.0	IBM CIC India	Draft	12-Feb-2018
4.0	IBM CIC India	Draft	16-Feb-2018
5.0	IBM CIC India	3rd Final	23-Feb-2018
6.0	IBM CIC India	Updated for latest technical aspects and new topics	10-May-2021

Table of Contents

1 SAP FIORI – DESIGN PRINCIPLES **6**

2 CODING PRACTICES **7**

2.1 LOAD GLOBAL DATA IN COMPONENT.JS 7
2.2 MINIMAL CONTENT IN INDEX.HTML 7
2.3 USE APPLICATION DESCRIPTOR (MANIFEST.JSON) 8
2.4 DO NOT USE HARDCODED STRINGS 8
2.5 DO NOT MIX VIEW AND CONTROLLER LOGIC 8
2.6 DO NOT MODIFY UI ELEMENTS OF OTHER VIEWS 8
2.7 DO NOT USE ANY DEPRECATED METHOD 8
2.8 APPLICATION STRUCTURE 8
2.9 DO NOT HARDCODE USERNAME AND PASSWORD 11
2.10 NAMING CONVENTIONS 11
2.11 CODE COMMENT 11
2.12 DECLARATION WITH 'VAR' ALWAYS 11
2.13 ALWAYS USE SEMICOLON AFTER EVERY STATEMENT. 12
2.14 USE ARRAY AND OBJECT LITERALS INSTEAD OF ARRAY AND OBJECT CONSTRUCTORS 12
2.15 AVOID USING EVAL() FUNCTION 12
2.16 CONTROL ID ASSIGNMENT 13
2.17 CALL BACK FUNCTIONS 13
2.18 ALWAYS SHOW OVERLAY 13
2.19 WHEN NOT TO USE 'THIS' 13
2.20 ALWAYS USE STRICT EQUALS OPERATOR 13
2.21 CODE INDENTATION 13
2.22 DEFINING A STYLE CLASS 13
2.23 PROPER USE OF CONFIGURATION PARAMETERS 13
2.24 DEPRECATED APIS 14
2.25 PRIVATE AND PROTECTED METHODS OR PROPERTIES 14
2.26 OVERRIDING OR ADDING CONTROL METHODS 14
2.27 HOW TO USE OF DOM EVENT HANDLERS 15
2.28 OTHER FACTORS 15
2.29 CODE DOCUMENTATION 15
2.30 SUPPORT ASSISTANT 16
2.31 GENERAL RULES TO KEEP IN MIND 16

3 LAYOUT **18**

3.1 PREVIEW THE APPLICATION IN DIFFERENT BROWSER 18
3.2 DO NOT SPECIFY FIXED SIZE 18
3.3 CHOOSE CORRECT LAYOUT 18

4 CONTROLLER **18**

4.1 BASE CONTROLLER 19

5 MODEL **20**

5.1 ODATA 20

5.2 Use of non-SAP oData/JSON based data sources 20
5.3 When to load Data........ 21
5.4 OData Vs REST 22

6 DESKTOP AND MOBILE APPLICATIONS 22

6.1 Do not mix UI elements from Mobile and Desktop libraries. 22
6.2 Create separate views for Mobile and Desktop devices. 22

7 INTERNATIONALIZATION........ 22

8 STANDARD FIORI LAUNCHPAD CONFIG FOR ECC........ 22

8.1 Create Tile and Target Mapping References 22
8.2 Naming Conventions for custom catalog and groups........ 22
8.3 Activating OData services 23
8.4 Activating SICF nodes 23
8.5 To check inconsistency in system alias........ 23
8.6 No Reference should be broken from standard in our custom catalog 23
8.7 Group personalization should be disabled while creating group 23

9 STANDARD FIORI LAUNCHPAD CONFIG FOR S4 HANA........ 24

9.1 Naming Conventions for custom catalog and groups........ 24
9.2 Create Tile and Target Mapping........ 24
9.3 Create group........ 25
9.4 System Alias Maintain........ 26
9.5 Service Activation 26
9.6 Performance Role........ 27
9.7 App Activation – SICF 28
9.8 App Dependencies that needs to be called out 28
9.9 Enterprise Search 28

10 FIORI LAUNCHPAD THEMING 28

11 FIORI ELEMENTS 29

12 SAP SMART BUSINESS APPLICATION-KPI WORKSPACE 34

13 TROUBLESHOOTING COMMONLY FACED CHALLENGES........ 36

14 GUIDELINES FOR COMMONLY USED UI5 CONTROLS 39

14.1 Table (sap.m.Table)........ 39
14.2 Smart Table 41
14.3 Form / Simple Form / Smart Form........ 41
14.4 Date 43
14.5 Button........ 44
14.6 Busy States........ 44
14.7 Barcode Scanning 45
14.8 PDF Viewer 45
14.9 Chart (VizFrame)........ 45

15 REUSABILITY CONCEPT 46

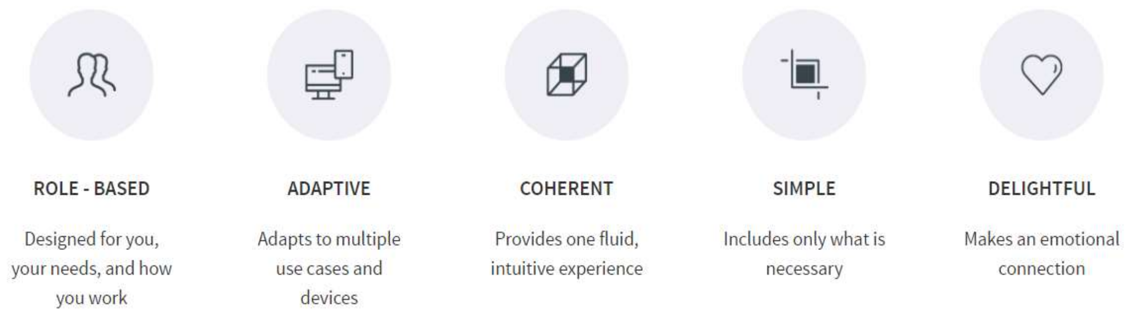
16 TESTING 47

17 DEBUGGING........ 47

18 PERFORMANCE CONSIDERATIONS......48
18.1 Asynchronous Module Definition......49
19 TRANSPORT AND VERSIONING......50
19.1 Using Git Repository......50
20 SECURING SAPUI5 APPLICATIONS......51
21 REVIEW CHECK POINTS......52
22 LIBRARIES AND REFERENCES......57

1 SAP Fiori – Design Principles

The user interfaces of the Fiori Applications are implemented with SAPUI5 technology built on top of OData services in the SAP NetWeaver Gateway, SAP HANA systems and S4HANA. These are again running on top of standard business logic in the ECC system or HANA tables/views. Currently there are more than 10000 prebuilt standard application which can be configured and used on top of these system. These apps adhere to "Fiori Design Principles" giving user a seamless experience across various interaction channels – desktop, tablet, and mobile. Below snapshot- taken from SAP Teched- explains the benefit of Fiori design principles



ROLE - BASED
Designed for you, your needs, and how you work

ADAPTIVE
Adapts to multiple use cases and devices

COHERENT
Provides one fluid, intuitive experience

SIMPLE
Includes only what is necessary

DELIGHTFUL
Makes an emotional connection

Fiori Launchpad

The SAP Fiori launchpad is a shell that hosts SAP Fiori apps, and provides the apps with services such as navigation, personalization, embedded support, and application configuration. The launchpad is the entry point to SAP Fiori apps on mobile and desktop devices. The launchpad displays a home page with tiles. Each tile represents a business application that the user can launch.

Main elements of the SAP Fiori launchpad are:

- **Home Page**: Its central access point for SAP Fiori apps. The page contains tiles, which are used to launch apps and can also show additional application information. Users can personalize the home page by adding, removing, and grouping app tiles. Because the launchpad is role-based, only apps that are relevant for the user's role profile are shown.
- **User Actions Menu**: The user actions menu offers a range of user-specific services, . It is accessed by clicking the icon or photo on the right-hand side of the shell bar.
  - General settings and preferences
  - A catalog of available apps (the app finder)
  - Objects and apps recently visited by the user
  - An About dialog, with details about the SAP Fiori launchpad or app version.
  - A Sign Out option for logging off the SAP Fiori launchpad.
- **Notifications**: Users can access notifications by clicking the *Notifications* button on the right of the shell bar

Feature of Fiori Launch

- *Simple* – intuitive, easy and coherent user experience.
- *Role based* – simplified role based navigation and business function access.
- *Contextual* – real time, contextual and personalized access.
- *Responsive* – consumption a cross devices, versions and channels with a single user experience.

Fiori Applications

Fiori applications cater for individual workflow task scenarios, for example a leave request approval task. The Fiori application provides the user with an overview of all his tasks of the given type (for example all his leave request approval tasks). A number of standard Fiori applications are delivered by SAP (for example "Leave Request"). These can be extended with custom functionality by leveraging the built-in extension points in the Fiori SAPUI5 applications. In addition, Fiori can be extended with custom scenarios. A Fiori app has been categories into following types:

- **Transactional Apps**:With transactional apps, users are able to perform activities such as creating, changing, and approving requests or orders, via guided navigation.
- **Fact Sheet Apps**: These apps are used to view essential contextual information or a 360-degree view of specific central objects used in business operations.
- **Analytical Apps**:With analytical apps, users are provided with business information and have the ability to analyze and evaluate strategic or operational KPIs in real time. This can be done on a large volume of data in a simplified front-end for enterprise control.
- **SAP Fiori Elements**: SAP Fiori elements are low-code templates that support popular application patterns. Developers can use SAP Fiori elements to create apps based on OData services and annotations that don't require JavaScript UI coding.

2 Coding practices

2.1 *Load Global data in Component.js*

When you are developing SAPUI5 application, if the application is multichannel (i.e. for desktop, mobile & tab) then always use 'sap.m' library. All UI controls should be from this library. Make sure, you are using UI controls, which are optimized for touch screen and smartphone. The globally required data such as device model, i18n resource model, data model etc. should be loaded in Component.js. The data required for a specific view should be in the controller of that view.

2.2 *Minimal content in index.html*

The heart of an application is defined in its Component part, which means that we should aim for very minimal content in the index.html file itself.

Please note if application is deployed in FLP, then you need to handle it through manifest.json and component.js as index.html is not used for FLP.

Few More Points: -

1. Do not use hardcoded String in the <title></title>
2. Apply i18n translation in <title></title>
3. Do not use HANA On demand SAPUI5 core lib link for any project application (Only for Developer end testing you can use)

4. Apply SAPUI5 cache-buster mechanism is a must activity. For not to use we should have proper supporting document.
5. Use bootstrap script configuration wisely. Do not apply unnecessary configuration which is not all used in the respective project.

2.3 *Use Application Descriptor (manifest.json)*

This application descriptor file is must for all application which having SAPUI5 version equal or above 1.30. And this file contain all application specific configuration settings. For any old application which is having older SAPUI5 version than 1.30, developer can use Component.js for this purpose.

2.4 *Do not use hardcoded strings*

The strings like labels, titles etc. should be retrieved from the i18n resource. Having the string in a property file gives the flexibility required for the multilingual support.

2.5 *Do not mix view and controller logic*

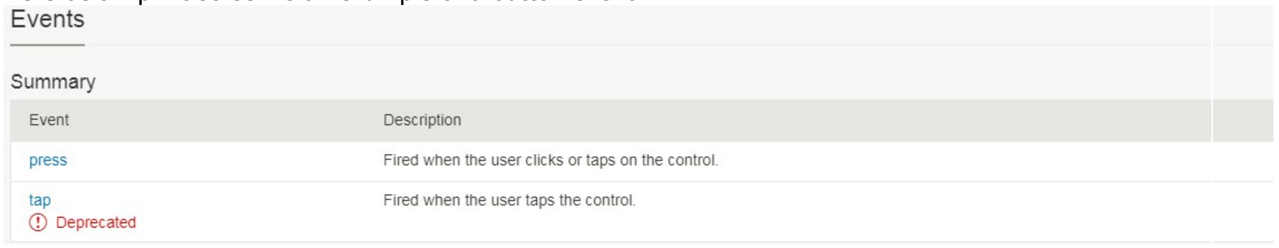
There should be a separation between view and controller. The event handlers should be put in the controller. This is to follow MVC patter, which helps in application sustain, modularization and improved performance. View should contain only UI elements. Ideally XML view should be used.

2.6 *Do not modify UI elements of other views*

Only the UI elements of the belonging view should be manipulated in that view. If there is a need to update a UI element of any other view, do it via event listeners. This makes it easy to understand the application flow.

2.7 *Do not use any deprecated method*

Here below print screen is an example of a button event.



Events

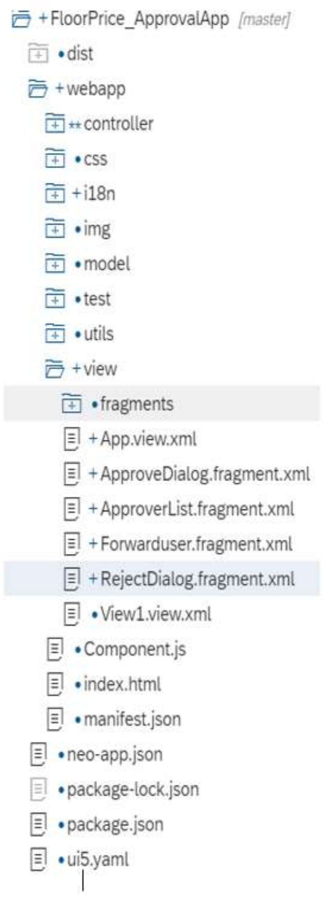
Summary

Event	Description
press	Fired when the user clicks or taps on the control.
tap Deprecated	Fired when the user taps the control.

2.8 *Application structure*

Currently we have two standard development environments for SAPUI5 WebIDE and the New BAS IDE. From app development point of view the structure of UI5 application differs between BAS and WebIDE, Below are the standard and suggested application structure for UI5 application. Please note the webapp folder inside project folder will be almost Identical in structure in both environments.

WebIDE Project Structure



```
+FloorPrice_ApprovalApp [master]
  •dist
  +webapp
    ++controller
    •css
    +i18n
    •img
    •model
    •test
    •utils
    +view
      •fragments
      +App.view.xml
      +ApproveDialog.fragment.xml
      +ApproverList.fragment.xml
      +Forwarduser.fragment.xml
      +RejectDialog.fragment.xml
      •View1.view.xml
    •Component.js
    •index.html
    •manifest.json
  •neo-app.json
  •package-lock.json
  •package.json
  •ui5.yaml
```

Project Folder
Dist -Folder created by SAPUI5 in when run Build command

webapp-Contains UI5 Application code as same as WebIDE project
Controller-Hold controller files
css- Contains application css
i18n->contains i18n file

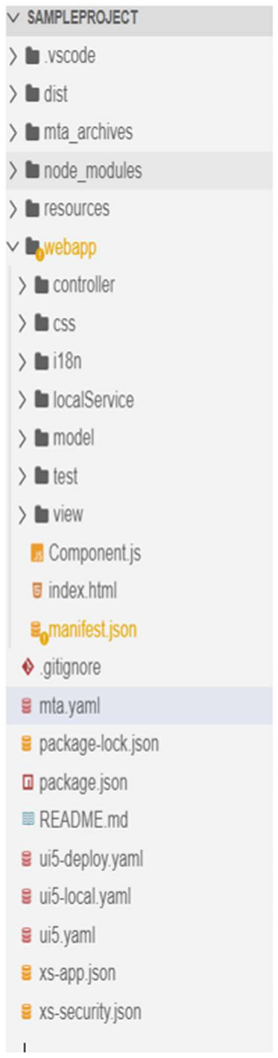
img-> Custom folder to organize project asset
model-model creation files

Test-> contains unit and integration testing file
utils-> its a custom folder, Any custom file can be put as in this structure manner
View folder- Contains application view files
view-> All views fragments go here; all file names shall start with uppercase
fragments->custom folder all fragments go here and name should follow convention *.fragment.xml

package.json- contains configuration detail of application as a node app contain dependency
ui5-yaml- Contains information to create build while deploying

S

Business Application Studio(BAS) UI5 Project Structure



```
SAMPLEPROJECT
  .vscode
  dist
  mta_archives
  node_modules
  resources
  webapp
    controller
    css
    i18n
    localService
    model
    test
    view
    Component.js
    index.html
    manifest.json
  .gitignore
  mta.yaml
  package-lock.json
  package.json
  README.md
  ui5-deploy.yaml
  ui5-local.yaml
  ui5.yaml
  xs-app.json
  xs-security.json
```

Project Folder
.vscod folder -Bas config file(For Internal uses by BAS ide)
Dist -Folder created by SAPUI5 in when run Build command

MTA Archives -Folder created by SAPUI5 in when run Build command contains deployable tar file

Node_Modules- Contains node library module
resources -Folder created by SAPUI5 in when run Build command and used to make tar file by the script

webapp-Contains UI5 Application code as same as WebIDE project
Controller-Hold controller files

mta.yaml- It contains the configuration and setting and use to make build of application

package.json- contains configuration detail of application as a node app contain dependency
information and Script commands

ui5-deploy - Contain deployment information and used by script
ui5-local- Used by script to run app locally
ui5-yaml- Contains information about backend connection-Destination and other settings

xs-app.json and xs-security.json contains information about approuter configuration and security profiling

2.9 *Do not hardcode username and password*

Do not in any case, even testing purpose, hard code the user ID/Password. Usually, SSO should be used for authentication for both application and gateway server. This could lead to serious security issues at run time in PROD systems. It's best to start with the right approach from the beginning.

2.10 *Naming Conventions*

Variables and functions must be correctly named:

- Functions must be in lower camelCase.
- Variables from jQuery begin with $.
- Mention object type at the beginning of variable name.

Objects such as Views, Custom controllers, and Components should be named in "UpperCamelCase" e.g. AddProduct.view.xml

Objects such as UI Control ID, and Model Name should be named as "camelCase" e.g. Page id = "rootPage".

For String variable give name as "sText", where "s" stands for "String".
For Object variable give name as "oData" where "o" stands for object.

Using the naming convention leads to a more readable and maintainable code.

2.11 *Code Comment*

Method/Function should start with comment explaining the purpose of it and the parameters. Also, it is a good practice to have inline comments in the code.

```
/**
 *@method greet
 *@param employee {string} The name of the Employee to greet
*/
doGreet: function (employee) {
   //Some code …
}
```

2.12 *Declaration with 'var' always*

When you fail to specify var, the variable gets placed in the global context, potentially clobbering existing values. Also, if there's no declaration, it's hard to tell in what scope a variable lives (e.g., it could be in the Document or Window just as easily as in the local scope). So always declare with var.

Note: Declare 'var' wisely, do not use this keyword repeatedly.

Recommendation:-
```
var foo = "",
    boo = "",
        woo = [];
```

2.13 *Always use semicolon after every statement.*

Relying on implicit insertion can cause subtle, hard to debug problems.

2.14 *Use Array and Object literals instead of Array and Object constructors*

Avoid:

```
var a1 = new Array(x1, x2, x3);
var a2 = new Array(x1, x2);
var a4 = new Array();

var o = new Object();
var o2 = new Object();
o2.a = 0;
o2.b = 1;
o2.c = 2;
o2['strange key'] = 3;
```

Use:

```
var a = [x1, x2, x3];
var a2 = [x1, x2];
var a3 = [x1];
var a4 = [];

var o = {};
var o2 = {   a: 0,
             b: 1,
             c: 2,
             'strange key': 3
};
```

2.15 *Avoid using eval() function*

Let's assume we have a server that returns something like this:

```
{
    "name": "Alice",
    "id": 31502,
    "email": "looking_glass@example.com"
}
var userInfo = eval(feed);
var email = userInfo['email'];
```

If the feed was modified to include malicious JavaScript code, then if we use eval then that code will be executed.

```
var userInfo = JSON.parse(feed);
var email = userInfo['email'];
```

With JSON.parse, invalid JSON (including all executable JavaScript) will cause an exception to be thrown.

2.16 *Control ID assignment*

To assign an ID to a control manually, always use this.createId(). Avoid using direct.

2.17 *Call back functions*

Always write code for success and error call back function while calling the gateway services or backend data calling either 'GET' or 'POST'.

2.18 *Always show overlay*

Always show overlay (busy screen) with a timeout while firing gateway services or backend calling and don't forget to remove overlay in both success & error callback functions.

More Points:-

- Whenever possible use setBusy() function for showing control specific busy state.

2.19 *When not to use 'this'*

The methods where a service will be called write the following in the first line and do NOT use directly 'this' afterwards in that method. 'this' is not be accessible inside callback methods and this leads to lot of problems and sometimes takes time to debug.

Proper Use:-

Below sample code is for using bind(this) / $.proxy()

```
var handleWOReadSuccess = function(oResponse) {
    var oDataArr = oResponse.results;
    // Write Required Logic Here
}.bind(this);

var handleWOReadSuccess = $.proxy(function(oResponse) {
    var oDataArr = oResponse.results;
    // Write Required Logic Here
}, this);
```

2.20 *Always use Strict Equals Operator*

=== and !== instead of '==' and '!=' according to latest JavaScript convention. '==' & '!=' does type corrections which should be avoided.

2.21 *Code indentation*

Always indent the code using 'Ctrl + Shift + F'. Unintended code looks like garbage.

2.22 *Defining a style class*

Use '-' only for defining CSS style class

2.23 *Proper use of configuration parameters*

Provide all configuration parameters of all controls in the instantiation time as far as possible instead of setting it later. Like:

Instead of

```
    var label = new sap.m.Label();
    label.setText('Abcd');
```
(54 characters)

Use

```
    var label = new sap.m.Label({
            text: 'Abcd'
    });
```
(48 characters)

This reduces character count drastically for complex controls and the code also looks neater.

2.24 *Deprecated APIs*

Don't use deprecated APIs. Entities marked as "deprecated" in the API Reference documentation (this includes properties, methods, events, and their parameters as well as entire controls and other APIs) are no longer intended to be used. They will not get feature updates in the future. Alternatives, if available, are described in the API Reference documentation.

One prominent example is the old jQuery.sap.device API that has been replaced with sap.ui.Device.

2.25 *Private and protected methods or properties*

Don't use or override "private" and "protected" functions. Private functions are typically (but not always) prefixed with "_". Protected functions are indicated by a yellow diamond in front of the function name within the SAPUI5 API Reference documentation.

Always double check in the SAPUI5 API Reference documentation. If UI5 changes the implementation in a future release, your code will break if you fail to follow this guideline.

Bad Examples	Good Example
oSelectDialog._oList.setGrowing(false)	oControl.getText();
oControl.mProperties["text"]	
oEvent.oSource.oBindingContexts.description.sPath.split('/')[3]	

2.26 *Overriding or adding control methods*

If you override methods like onBeforeRendering, onAfterRendering, or getters and setters, the original methods will no longer be called. You have to make sure that you call them in your method explicitly. Even if they are not implemented right now, they could be added in the future. This applies to control inheritance in particular.

Instead, you should consider using delegates.

Bad Examples	Good Example
oControl.onAfterRendering = fnMyFunction	`oControl.addEventDelegate({ onAfterRendering:function() { // do something } });`
oControl.prototype.setText = function(){ ... };	

2.27 *How to use of DOM event handlers*

Use attachBrowserEvent() if you need to listen to any DOM event on UI5 controls. An even better approach is to use addEventDelegate() for the most important event types instead, as it avoids additional event registrations and listens to the regular UI5 event dispatching.

If you are creating event handlers in custom controls, you can use listen to DOM events directly, but make sure that the listeners are properly deregistered in onBeforeRendering() and in exit(), and registered in onAfterRendering().

Good example for arbitrary events:

```
oControl.attachBrowserEvent("mousemove", function () {
    // do something
});
```

Good example for wide but limited selection of browser events:

```
oControl.addEventDelegate({
        onmouseover:function() {
            // do something
        }
    });
```

Please note - First, try to use SAPUI5 control's standard event. If requirement is not achievable, then use browser event.

2.28 *Other Factors*

- Don't build apps without reasonable automated tests
- Don't use timeouts
- Don't use console.log(), use jQuery.sap.log.* instead
- Don't hard code or concatenate strings that need to be translatable
- Don't forget about control lifecycle management
- Don't manipulate the DOM structure within controls

2.29 *Code Documentation*

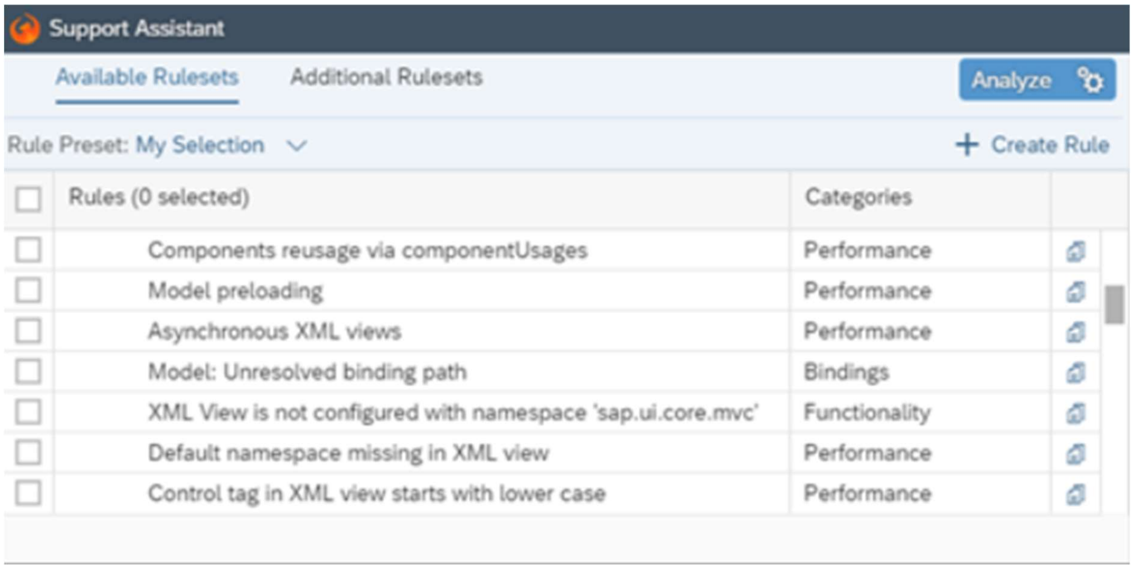
- *Please put comment against code.*
- *Use JSDoc tag against each method as below.*
- Generate JSDoc against code.

```
• /**
•  * Represents a book.
•  * @constructor
•  * @param {string} title - The title of the book.
•  * @param {string} author - The author of the book.
•  */
• function Book(title, author) {
• }
```

2.30 *Support Assistant*

UI5 comes with a built in tool called Support Assistant to check whether the application is build according to SAP Best Practices. Every project developing custom UI5 apps should use this tool to make sure that the app is built according to the design guidelines and best practices.

There are inbuild rules in the tool that can readily used as below.



	Rules (0 selected)	Categories
☐	Components reusage via componentUsages	Performance
☐	Model preloading	Performance
☐	Asynchronous XML views	Performance
☐	Model: Unresolved binding path	Bindings
☐	XML View is not configured with namespace 'sap.ui.core.mvc'	Functionality
☐	Default namespace missing in XML view	Performance
☐	Control tag in XML view starts with lower case	Performance

2.31 *General rules to keep in mind*

Few coding approaches one need to follow as a best practice for web and mobile applications.

Web & Mobile:

1. DONOT refresh ODATA model which is bind with UI.
2. Try to avoid use of id if possible.
3. Entire view should have controlled through a MODEL (JSON / ODATA).

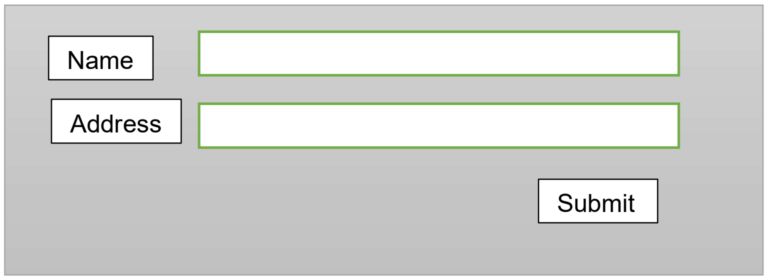
   e.g. Suppose there is a user registration screen as follows.

   Name and Address need to provide. Bind a JSON model the view.

   var jsonModel = new sap.ui.model.json.JSONModel();
   var data = {};

```
data.userName = null;
 data.address = null;
  jsonModel.setData(data);
```

Bind this model to view. If name or address value are changed, for JSON two-way binding data will be
automatically reflected in model.



4. Try to use validation on top of model.

   e.g. On submit, data can be checked from model, don't require any id for input validation.

   ```
   If(this.getView().getModel().getProperty("/userName") === null ) {
       // set Error message
   }
   ```

5. Use ODATA batch request instead of individual request.
6. Don't accept too multi label expand of oData. Instead of use navigation. This is hitting the performance badly specially when large amount of data are handled.

   e.g. "/Entities?$expand=Details,Details/Tips, Details/Tips/Details" - Try to avoid this.

   Use like

   ```
   /Entities('123456')/Details
   /Entities('123456')/Details/Tips
   /Entities('123456')/Details/Tips/Details
   ```

7. Don't use nested for loop while writing any logic. Use jQuery functions to handle this scenario.

   e.g. $.map , $.filter, $.each , $.extend etc.

8. Don't fetch large data at a time instead of fetch small chunk of data for performing operation specially for mobile application. Sometimes application got crashed for that.

9. Don't modify any SAP standard css. Instead custom css extending the standard.

10. Use promise/Deferred object to perform Async or dependency task like fetching data from service.

```
this.oGeoVaritiesFinishedDeferred = jQuery.Deferred();
- -
- -
$.when(this.oGeoVaritiesFinishedDeferred).then($.proxy(function() {
    this.getView().byId("pnVarietyPlot").setVisible(true);
}, this));
Somewhere in code or other function:
…………………….
    this.oGeoVaritiesFinishedDeferred.resolve();
 - - - ------ - - - - - - -  -- -  - -  - -  --
     - -
     - -
     - -
     this.oGeoVaritiesFinishedDeferred.reject();
      - -
      - -
```

3 Layout

3.1 *Preview the application in different browser*

Always preview your application in different browser (as necessary) to keep close watch on layout related changes and support of features used in your application.

3.2 *Do not specify fixed size*

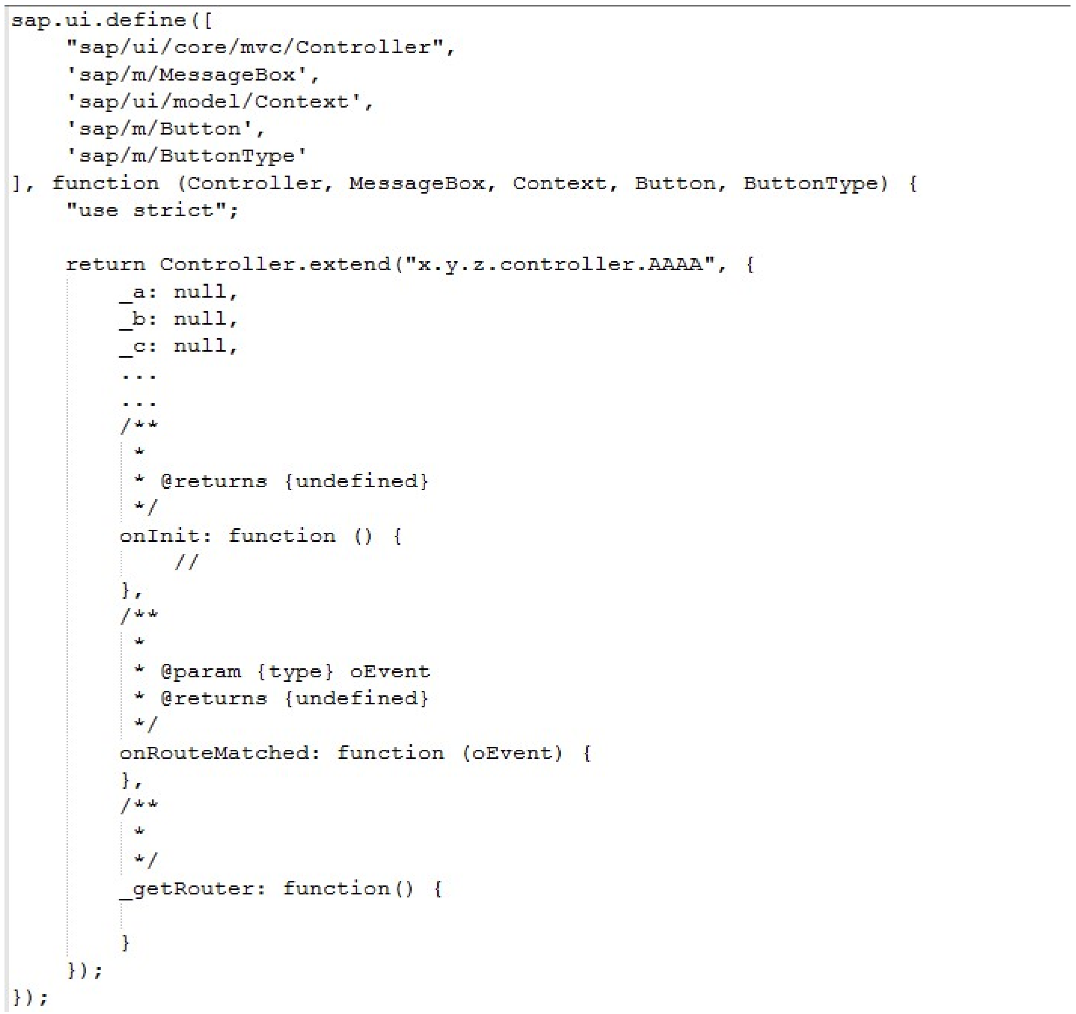
Don't specify fixed sizes for layouts or panel like controls just only to get the desired look in one browser. This may cause additional layout adjustment effort later. So always use percentage while specifying the size as necessary.

3.3 *Choose correct layout*

Choosing a correct layout to design any user interface can save lot of effort behind adjusting the screen size for different screen resolutions or browsers. Also avoid using so many combinations of different layouts unless absolutely necessary or can't be avoided.

4 Controller

All controllers should be in 'controller' folder as mentioned in section 1.6. The controller definition should start like (where 'AAA' is the controller name, 'x.y.z' is the namespace and 'controller' is the name of the folder):



```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    'sap/m/MessageBox',
    'sap/ui/model/Context',
    'sap/m/Button',
    'sap/m/ButtonType'
], function (Controller, MessageBox, Context, Button, ButtonType) {
    "use strict";

    return Controller.extend("x.y.z.controller.AAAA", {
        _a: null,
        _b: null,
        _c: null,
        ...
        ...
        /**
         *
         * @returns {undefined}
         */
        onInit: function () {
            //
        },
        /**
         *
         * @param {type} oEvent
         * @returns {undefined}
         */
        onRouteMatched: function (oEvent) {
        },
        /**
         *
         */
        _getRouter: function() {

        }
    });
});
```

As shown in the screenshot above, all the controls/classes used should be defined in the top and its alias should be the argument of the main function. All over this controller never use the fully qualified name, rather use the alias. For example, if the screenshot sap.m.Button is defined at the top and its alias is passed as argument with the name 'Button'. While creating or using button,
DO NOT use

    var button = new sap.m.Button ();

Use

    Var button = new Button();

This is true for any control/class to be added in the controller. No fully qualified (like x.y.ZZZZ) notation should be present in any controller. Be careful about the order of the arguments and their definitions, specifically when something is removed from that list.

4.1 *Base Controller*

All the controllers in a project should be inherited form a global base controller. Functions that are common across controllers only should be written in a global base controller and all the view controllers should inherit this base controller by extending it. Base controller should be defined light and should not contain intensive logic.

Base controller should be defined as just a .js file in your controller folder. Something like *BaseController.js*

base controller should be defined as below.

```
sap.ui.define([
        "sap/ui/core/mvc/Controller",
        "sap/ui/core/UIComponent"
], function (Controller, UIComponent) {
        "use strict";
        return Controller.extend("my.application.controller.BaseController", {
                function1(){},
                function 2(){},
                ……
        }
```

All your view controllers should extend this BaseController.js as below.

```
sap.ui.define([
  "./BaseController"
], function(BaseController) {

  "use strict";

  return BaseController.extend("my.application.controller.BaseController", {

    ...

  });

)};
```

5 Model

The DeviceModel should be named “device”.
The ResourceModel should be named “i18n”.

5.1 *OData*

As per the standard definition “*OData is a standardized protocol for creating and consuming data APIs. OData builds on core protocols like HTTP and commonly accepted methodologies like REST. The result is a uniform way to expose full-featured data APIs.*"

In SAP NetWeaver Gateway, entities are defined and connected in data models. These data models can be exposed as OData services during runtime. The recommendation is to use OData model for any server data access

5.2 *Use of non-SAP oData/JSON based data sources*

SAPUI5 very much can be used with non-sap REST data source as well. But make sure how authentication mechanism can be established. To get data from authenticated non-SAP system/ajax service , JSONModel can be used, To avoid the CORS issue configure you service url as destination. Use below step to use service outside form SAP system without CORS issue:

1. First configure it as Destination in you cloud platform cockpit under Destination.
2. Update neo-app.json with the newly created destination:
3. Use JSON model to load data in model fetched from Ajax Services.

If you are calling ajax services and then make sure you have added the valid X-CSRF-Token in ajax header.
To Post Data first must fetch the X-CSRF-Token and store it in variable, The call the post Service using this fetched token, To avoid authentication issue. Like below
To fetch Token:

```
var that =this;
  $.ajax({
     type: "GET",
     url: "/yourDestination/servicepath",
     headers:{"x-CSRF-Token": "Fetch"}
  }).always(function(data, status,response){
     //store in class vaiable to use
     that.myToken = response.getResponseHeader("x-csrf-token");

  });
```

To POST Data:

```
$.ajax({
     type: "POST",
     url: "/yourDestination/servicepath/service",
     headers:{"x-CSRF-Token": that.myToken},
     contentType:"application/json",
     dataType:"json",
     data:YOUR_DATA_To_Be_POSTED
  }).always(function(data, status,response){
     //Resopnse check


  });
```

5.3 *When to load Data*

The rule of thumb: models for globally available data should be instantiated in Component.js. If some data source is used in a single view, the corresponding model should be created in the controller of that particular view. Load data in init if you know that it should be immediately available. Load auxiliary and rarely used data only when needed.
If a model is initialized in 'init' and its 'attachRequestCompleted' or attachRequestSent' is added, don't forget to add 'attachRequestFailed' to ensure proper error handling.

5.4 *OData Vs REST*

1. OData is optimized for SAP CRUD operations, but REST is a generic architecture and not optimized for SAP.

2. OData mandates the following conventions that REST does not, which makes OData preferred for SAPUI5.
    i. Representation formats – JSON Entities
    ii. Exact verb meanings – CRUD is defined explicitly and not just POST, PUT GET etc
    iii. URL Parameters for sorting, filtering, and paging. $ORDERBY, $TOP, $FILTER,

6 Desktop and Mobile Applications

6.1 *Do not mix UI elements from Mobile and Desktop libraries.*

Use the UI libraries specific to the requirement. Do not mix UI elements from Mobile and "Desktop" library of UI5. Using mixed libraries for developing views could lead to incorrect rendering of the UI element at run time

6.2 *Create separate views for Mobile and Desktop devices.*

Manage device switching during the application initialization. Proper ui layout on different devices enhances the user experience, which is one of the key expectations of a UI5 application

7 Internationalization

The property file should be named as explained below
messageBundle_xx.properties [Here "xx" is the country code]
.

8 Standard FIORI Launchpad Config for ECC

8.1 *Create Tile and Target Mapping References*

1. For designer, CUST scope is suggested.
2. Whenever possible, Use the SAP technical catalog to create the Tile / Target Mapping Reference

8.2 *Naming Conventions for custom catalog and groups*

1.As for naming catalogs to use:
ZC<process area>_</area code/ shortened catalog description>

2.For groups:
ZG<process area>_</area code/ same shortened description as for catalog>

3.for example ZCP2P_PR for a catalog with description "P2P Purchase requisition processing."
ZGP2P_PR for a group with description "P2P Purchase requisition processing."

8.3 *Activating OData services*

1. Activate only those OData services which are required
2. Assign the system alias to the services as per the SAP fiori apps library.

8.4 *Activating SICF nodes*

1. Activate only those SICF nodes which are required

8.5 *To check inconsistency in system alias*

Fiori System Alias Check (transaction /UI2/FSAC) displays consistency issues for the selected system alias.

8.6 *No Reference should be broken from standard in our custom catalog*

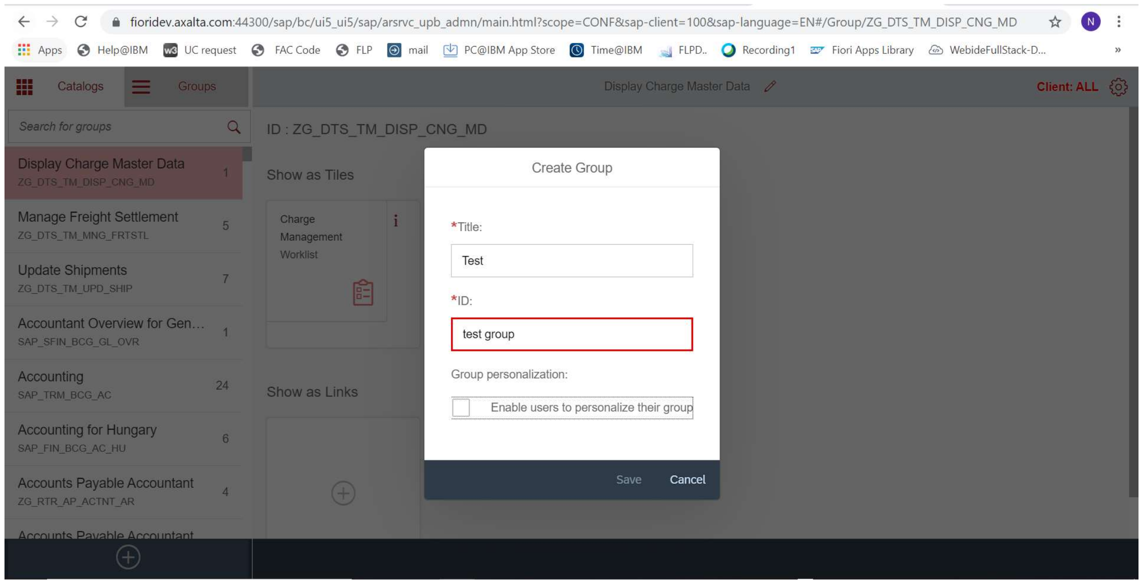
1. Reference should not be broken for custom catalog in which we have added tiles and target mapping taking reference from standard. During upgrade, changes present in standard catalog will not reflect in custom catalog if reference is lost.

2. We should always create reference from standard catalog for tile and target mapping in custom catalog.

8.7 *Group personalization should be disabled while creating group*

A group contains predefined content that you can view on the page. You can add tiles to groups from catalogs. You can specify that a group cannot be personalized by the end user.

We should uncheck checkbox for Enable Users to personalize their group so that users cannot make changes in the custom group (Allow personalization only when it is explicitly required). If they want to make some modifications, they can add to "My Home".


fioridev.axalta.com:44300/sap/bc/ui5_ui5/sap/arsrvc_upb_admn/main.html?scope=CONF&sap-client=100&sap-language=EN#/Group/ZG_DTS_TM_DISP_CNG_MD
Catalogs
Groups
Display Charge Master Data
Client: ALL
Search for groups
ID : ZG_DTS_TM_DISP_CNG_MD
Display Charge Master Data ZG_DTS_TM_DISP_CNG_MD 1
Manage Freight Settlement ZG_DTS_TM_MNG_FRTSTL 5
Update Shipments ZG_DTS_TM_UPD_SHIP 7
Accountant Overview for Gen... SAP_SFIN_BCG_GL_OVR 1
Accounting SAP_TRM_BCG_AC 24
Accounting for Hungary SAP_FIN_BCG_AC_HU 6
Accounts Payable Accountant ZG_RTR_AP_ACTNT_AR 4
Show as Tiles
Charge Management Worklist
Show as Links
Create Group
*Title:
Test
*ID:
test group
Group personalization:
Enable users to personalize their group
Save
Cancel

9 Standard FIORI Launchpad Config for S4 HANA

9.1 *Naming Conventions for custom catalog and groups*

1.As for naming catalogs to use:
ZC<process area>_</area code/ shortened catalog description>

For Parent and Child Approach

ZC<process area>_BC_<catalog_number>

ZC<process area>_</area code/ shortened catalog description>_CHLD

2.For groups:
ZG<process area>_</area code/ same shortened description as for catalog>

3.for example ZCP2P_PR for a catalog with description "P2P Purchase requisition processing."
ZGP2P_PR for a group with description "P2P Purchase requisition processing."

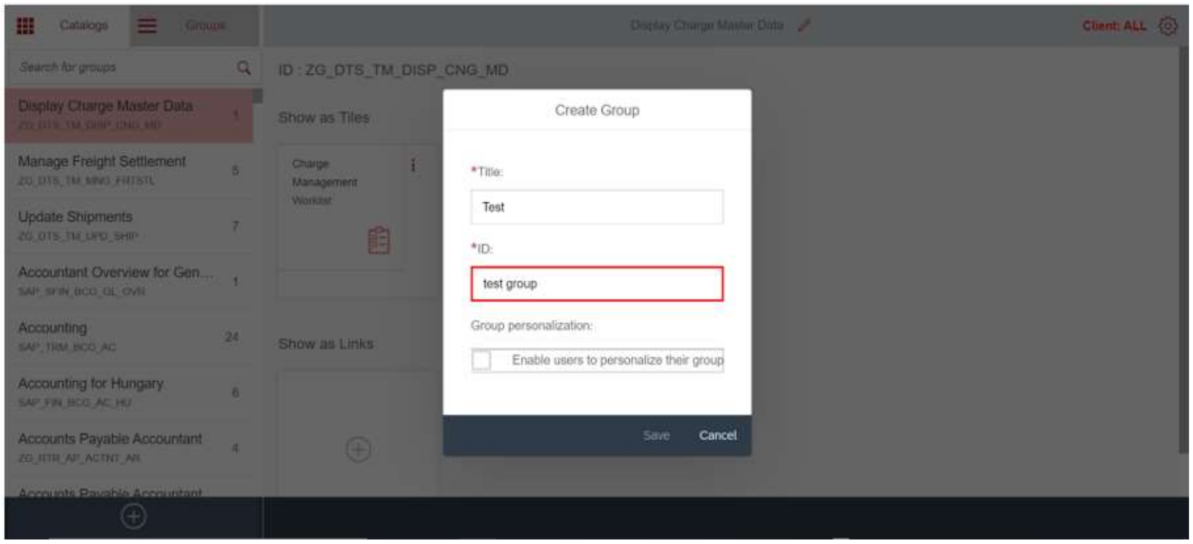
9.2 *Create Tile and Target Mapping*

1. For designer, CUST scope is suggested.

2. Whenever possible, use the SAP technical catalog to create the Tile / Target Mapping with reference
3. Try to avoid breaking reference from standard. During upgrade, changes present in standard catalog will not reflect in custom catalog if reference is lost.
4. It is recommended to create customer-specific catalogs rather than adjust catalogs shipped by SAP. Changing a catalog delivered by SAP on CUST layer, decouples the catalog from changes on CONF layer.
5. If SAP delivers new applications in the catalog, they cannot be integrated into the same catalog on CUST layer without losing the changes done by the customer.
6. Create a business group and fill it with content from the new catalog(s). It is recommended to create customer-specific groups rather than adjusting groups shipped by SAP.
7. Applications (extended SAP applications or custom applications) are integrated in the launchpad. These applications need to be registered in the target mapping or in transaction LPD_CUST.
8. It is advisable to use transaction /UI2/FLPCM_CUST and /UI2/FLPCM_CONF for copying catalog in mass instead of copying it manually.
9. Based on security task role design, you can follow parent and child approach of catalog creation. In this case, tiles should be in one catalog and target mappings should be in another catalog. This is a flexible design approach. For more details please refer to the deck of parent-child catalog approach.
10. If a good number of same apps are distributed amount multiple task role, then you can follow parent-child approach. It will reduce the development and maintenance effort.
11. If based on authorization, different number of apps need to need to display in same group, please follow parent-child approach.
12. For Fiori Designer changes, create TRs based on process area.
13. For Fiori Designer changes, maintain the sequence for TR movement.

9.3 *Create group*

It is advisable to disable option for group personalization – We should uncheck checkbox for Enable Users to personalize their group so that users cannot make changes in the custom group. If they want to make some modifications, they can add to "My Home". If we enable the option for personalizing the group for users, users might remove few of the important tiles required for them. Also, it will cause issues in troubleshooting as well. So, it is recommended to uncheck the checkbox until and unless it is very much required by the user.



ID : ZG_DTS_TM_DISP_CNG_MD

Display Charge Master Data	ZG_DTS_TM_DISP_CNG_MD	1
Manage Freight Settlement	ZG_DTS_TM_MNG_FRTSTL	5
Update Shipments	ZG_DTS_TM_UPD_SHIP	7
Accountant Overview for Gen…	SAP_SFIN_BCG_GL_OVR	1
Accounting	SAP_TRM_BCG_AC	24
Accounting for Hungary	SAP_FIN_BCG_AC_HU	6
Accounts Payable Accountant	ZG_RTR_AP_ACTNT_AR	4
Accounts Payable Accountant		

Create Group

*Title: Test

*ID: test group

Group personalization: [ ] Enable users to personalize their group

9.4 *System Alias Maintain*

1. System Alias should not be set for S/4HANA oData services unless it is using a separate gateway. For embedded S/4 gateway, 'co-deployed' should be the set as processing mode.
2. "S4FIN" system alias should be in the list of system alias otherwise some financial app will not work.
3. "S4PRC" should be also added for some procurement analytics app to work.
4. Sometimes, maintaining system alias is a part of cutover activity specially for custom app.

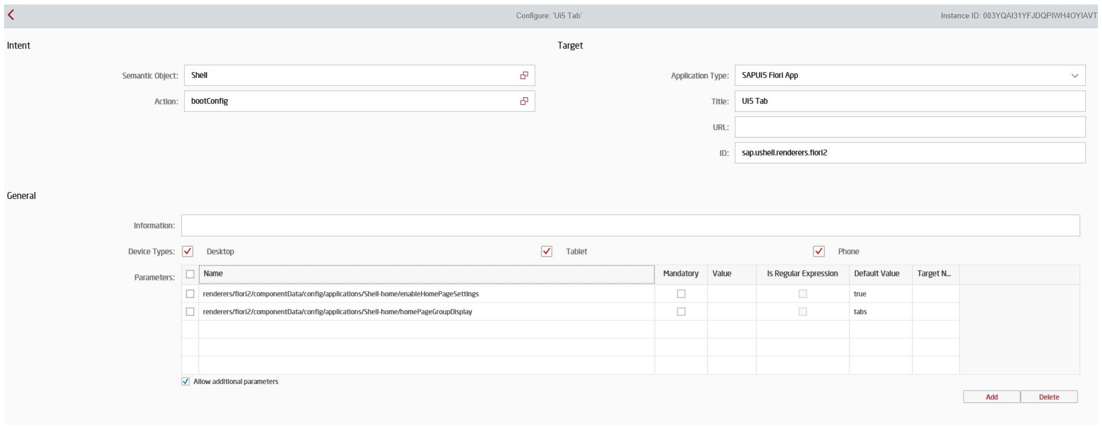
9.5 *Service Activation*

1. Only activate those service which are in scope of your project. Others it is suggested to keep deactivated.
2. There are some default Fiori services (i.e. ESH_SEARCH_SRV), that we need to activate, and access should be under default role.
3. Always do mass activation of OData service in a Sandbox system. Generally there will not be a transport path from Sandbox to Dev, so it is recommended that the required /identified Fiori apps are then separately activated and configured in Dev box and then move the changes to quality and further systems through transport request.
4. Some dependent oData service may need to activate manually in the systems. You can maintain a list and execute the task list for activating service to other systems.
5. Follow the same rule as point 4 for some services which are not activated during mass activation.
6. Catalogue the oData list, which are not activated, into process area and use different TR for each process area.
7. Use script/program to assign system alias for oData service for different systems.
8. If you have a smaller number of oData need to activate after mass activation, you can do with manual activation and moving through TR.

9.6 *Performance Role*

1. For improving the performance, it is recommended to add some specific roles. It should be under default role of all Fiori Users.

   a. Open GUI and Web Dynpro in same tab of the browser (create a role with catalog /UI2/CONFIG_NAVIGATION_MODE and assign to user)
   b. Display only one group at a time (Create a catalog with below target mapping and assign to default role to the user)



Configure: 'Ui5 Tab'   Instance ID: 003YQAI31YFJDQPIWH4OYIAVT

Intent

Semantic Object:	Shell
Action:	bootConfig

Target

Application Type:	SAPUI5 Fiori App
Title:	Ui5 Tab
URL:	
ID:	sap.ushell.renderers.fiori2

General

Information:

Device Types: ✓ Desktop ✓ Tablet ✓ Phone

Parameters:

Name	Mandatory	Value	Is Regular Expression	Default Value	Target N...
renderers/fiori2/componentData/config/applications/Shell-home/enableHomePageSettings	☐		☐	true	
renderers/fiori2/componentData/config/applications/Shell-home/homePageGroupDisplay	☐		☐	tabs	

✓ Allow additional parameters

   c. Implement Role based search – search will be based on process area / sub process area. For more details refer below link.

      https://gad5158842f.us2.hana.ondemand.com/dtp/viewer/#/tree/1775/actions/22992:22995

      https://help.sap.com/doc/saphelp_nw75/7.5.5/en-US/b4/f6b7313bf2455fa5bb3fa2abd52436/content.htm?no_cache=true

      https://help.sap.com/doc/saphelp_nw75/7.5.5/en-US/b4/f6b7313bf2455fa5bb3fa2abd52436/frameset.htm

   d. Implement CDN if required** - (for global implementation may require). Create a role with Catalog /UI2/CONFIG/UI5/CDN
   e. Catalogs and groups should be made based on task role instead of job roles.

      Note: Task role is the role made at granular level, job role is a combination of multiple task roles.

2. Don’t keep more tiles in a group – try to keep 30 tiles/app in a catalog/group for better performance.

9.7 *App Activation – SICF*

1. Mass activation of standard apps should be decided based on the number of apps to be activated. If the selected number of apps are less than 100 its is better to go for manual activation. If the total apps are more than 100 Mass activation can be considered.
2. Activation of custom app in further system is a part of cutover activity.
3. Only activate those apps, which is required for your project.
4. Execute task list for SICF mass activation and not do not do individual manual node activation.
5. For custom apps, after moving to other systems (QA, UAT, PRD etc.) need to activate either through execution of task list or do it manually. If you are having shorten list, can be done it manually. Either wise execute task list.

9.8 *App Dependencies that needs to be called out*

1. In standard Fiori apps are having navigation to other apps. We need to activate the depended apps. This has to be identified by functional folks in collaboration with UX consultant and listed down as dependent apps.
2. List of navigating apps should be mentioned by process owners/security team.
3. List of navigating apps are determined by business requirements and authorization.

9.9 *Enterprise Search*

1. Enterprise Search should be activated in each client using the Task List
2. Only relevant business objects should be activated as per the requirement. For this use the ESH Cockpit. After activating ent search using the task list, go to the ESH cockpit and disable all the business objects that are not used.
4. Search should be controlled using Roles and authorization for each functional area. Only the rel
5. Apps should be enabled for search followed by other functional objects as per requirement.
6. Client copy or refresh can disable the Enterprise Search configuration so every time a client copy or refresh happens make sure to run the required task list to clean up the client.

10 Fiori Launchpad Theming

1. Use theme designer tool for developing custom theme.
2. Please check "Theme Accelerator Tools" if you can utilize any custom theme for your project.
3. Custom theme is UI5 version dependent. You must build theme with same SAPUI5 version as FLP.
4. Don't use a large image as background of FLP. Image size should be below 200 KB for desktop.
5. For Mobile, use small image less than 50 KB.
6. Don't use large image as favorite icon.
7. For custom theme id, follow below naming convention.
   custom_<project_name>_<standard_theme>
8. If your system version got updated, you may need to build theme for higher SAPUI5 version.
9. Apply custom theme as default theme in FLP.
10. Validate your theme for GUI, WebDynpro, Fiori, Fact Sheet, Fiori Elements, KPI Tile based, Design Studio all types of Fiori apps including custom SAPUI5 apps.
11. Validate your custom theme for mobile layout.

12. Don’t use custom css for theme as css is depend on SAPUI5 version. After library upgrade, custom css may not work.
13. Use standard theme “SAP Belize” as a base for custom theme for Fiori-2.
14. User standard theme “SAP Quartz Light” as a base for custom theme for Fiori -3.
15. Theme should be based on client’s branding guidelines.

Theme Parameter Toolbox

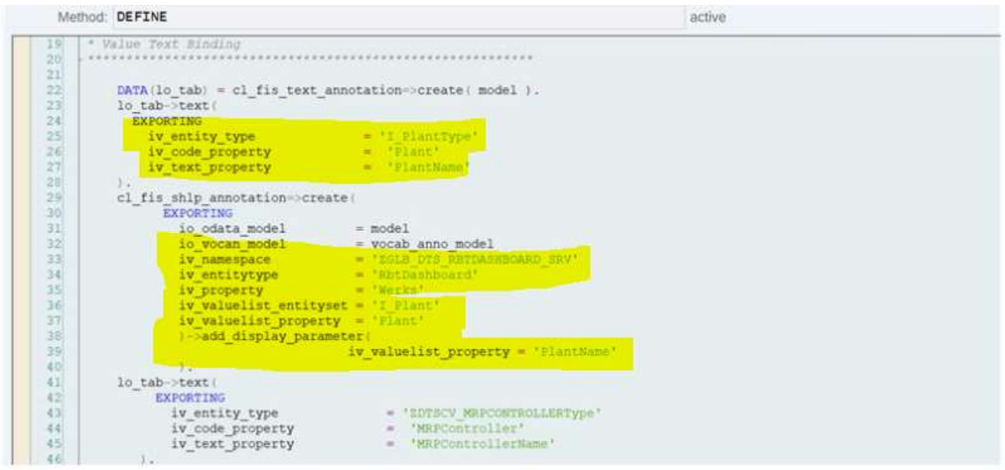
Theme parameter details of what parameter is used where is detailed in the blow theme parameter toolbox.

https://sapui5.hana.ondemand.com/test-resources/sap/m/demokit/theming/webapp/index.html

11 Fiori Elements

1. Annotations are suggested to add in the backend CDS service or OData service rather than local annotations. Annotations can be added for Filter bar, for data display in table and data display in chart format. Thus, we can avoid/minimize code maintenance form in one side.

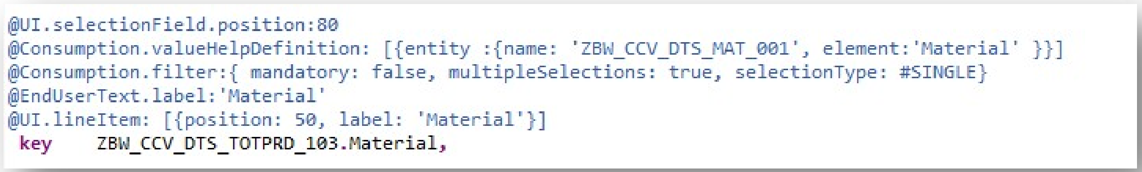
Example: OData Annotation



```
* Value Text Binding
*************************************************************

    DATA(lo_tab) = cl_fis_text_annotation=>create( model ).
    lo_tab->text(
      EXPORTING
        iv_entity_type             = 'I_PlantType'
        iv_code_property           =  'Plant'
        iv_text_property           =  'PlantName'
    ).
    cl_fis_shlp_annotation=>create(
          EXPORTING
            io_odata_model          = model
            io_vocan_model          = vocab_anno_model
            iv_namespace            = 'ZGLB_DTS_RBTDASHBOARD_SRV'
            iv_entitytype           = 'RbtDashboard'
            iv_property             = 'Werks'
            iv_valuelist_entityset  = 'I_Plant'
            iv_valuelist_property   = 'Plant'
          )->add_display_parameter(
                          iv_valuelist_property = 'PlantName'
          ).
    lo_tab->text(
          EXPORTING
           iv_entity_type              = 'ZDTSCV_MRPCONTROLLERType'
           iv_code_property            =  'MRPController'
           iv_text_property            =  'MRPControllerName'
      ).
```

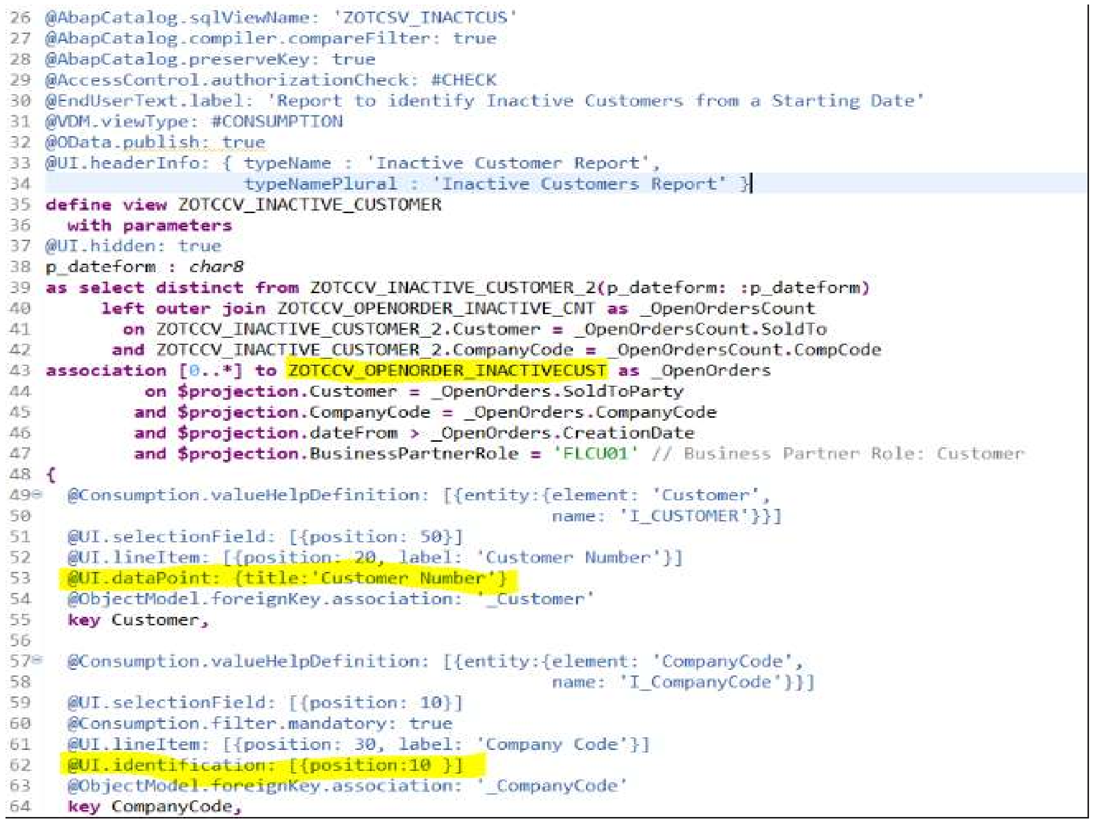
CDS Annotation Example:



```
@UI.selectionField.position:80
@Consumption.valueHelpDefinition: [{entity :{name: 'ZBW_CCV_DTS_MAT_001', element:'Material' }}]
@Consumption.filter:{ mandatory: false, multipleSelections: true, selectionType: #SINGLE}
@EndUserText.label:'Material'
@UI.lineItem: [{position: 50, label: 'Material'}]
 key    ZBW_CCV_DTS_TOTPRD_103.Material,
```

@UI.selectionField.position: 60: Annotation to add Filter
@UI.lineItem: [{position: 30, label: 'Material Group'}]: Annotation to add column in table.
@Consumption.valueHelpDefinition: [{entity :{name: 'ZBW_CCV_DTS_MATGRP_002', element:'MatGrp' }}]: Annotation to add value help for filter
@Consumption.filter:{ mandatory: false, multipleSelections: true, selectionType: #SINGLE}: Annotation to add properties into value help.
@EndUserText.label:'Material Group': Annotation to add end user label.

2. Detail page navigation or item navigation recommended to be handled through backend annotation.

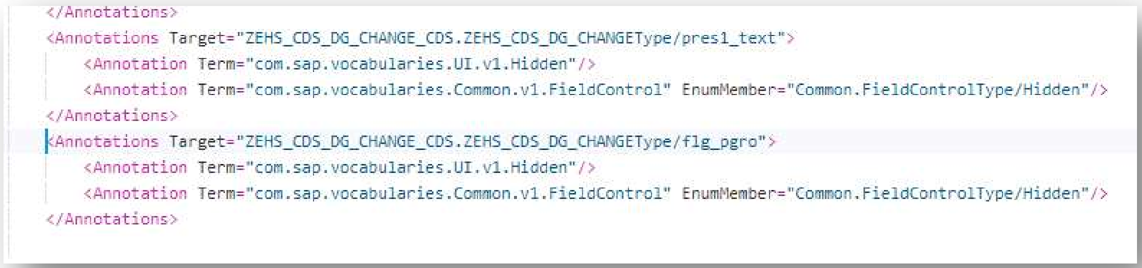


```
@AbapCatalog.sqlViewName: 'ZOTCSV_INACTCUS'
@AbapCatalog.compiler.compareFilter: true
@AbapCatalog.preserveKey: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'Report to identify Inactive Customers from a Starting Date'
@VDM.viewType: #CONSUMPTION
@OData.publish: true
@UI.headerInfo: { typeName : 'Inactive Customer Report',
                  typeNamePlural : 'Inactive Customers Report' }
define view ZOTCCV_INACTIVE_CUSTOMER
  with parameters
@UI.hidden: true
p_dateform : char8
as select distinct from ZOTCCV_INACTIVE_CUSTOMER_2(p_dateform: :p_dateform)
     left outer join ZOTCCV_OPENORDER_INACTIVE_CNT as _OpenOrdersCount
       on ZOTCCV_INACTIVE_CUSTOMER_2.Customer = _OpenOrdersCount.SoldTo
      and ZOTCCV_INACTIVE_CUSTOMER_2.CompanyCode = _OpenOrdersCount.CompCode
association [0..*] to ZOTCCV_OPENORDER_INACTIVECUST as _OpenOrders
         on $projection.Customer = _OpenOrders.SoldToParty
        and $projection.CompanyCode = _OpenOrders.CompanyCode
        and $projection.dateFrom > _OpenOrders.CreationDate
        and $projection.BusinessPartnerRole = 'FLCU01' // Business Partner Role: Customer
{
  @Consumption.valueHelpDefinition: [{entity:{element: 'Customer',
                                      name: 'I_CUSTOMER'}}]
  @UI.selectionField: [{position: 50}]
  @UI.lineItem: [{position: 20, label: 'Customer Number'}]
  @UI.dataPoint: {title:'Customer Number'}
  @ObjectModel.foreignKey.association: '_Customer'
  key Customer,

  @Consumption.valueHelpDefinition: [{entity:{element: 'CompanyCode',
                                      name: 'I_CompanyCode'}}]
  @UI.selectionField: [{position: 10}]
  @Consumption.filter.mandatory: true
  @UI.lineItem: [{position: 30, label: 'Company Code'}]
  @UI.identification: [{position:10 }]
  @ObjectModel.foreignKey.association: '_CompanyCode'
  key CompanyCode,
```

3. We should add UI annotations only if there is any limitation to put annotation in backend service.

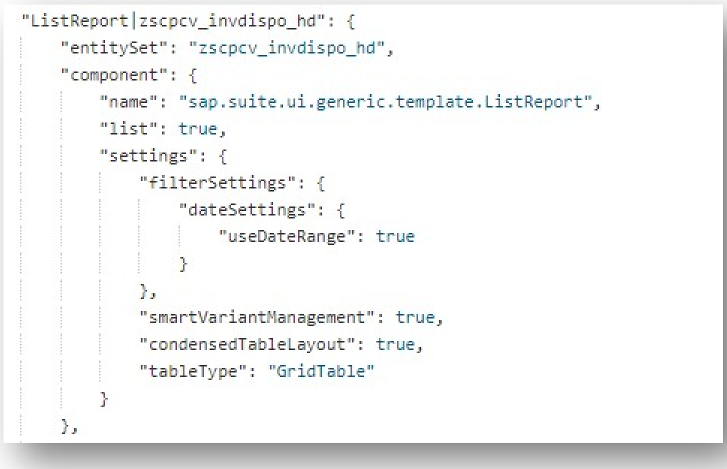
For example:
Case 1: To hide virtual elements from UI, use local annotations but always pass virtual elements in the service.
Case 2: UI annotation or local annotation should be there to hide the extra fields (formatted data field) exposed by the backend service due to analytical query.



```xml
</Annotations>
<Annotations Target="ZEHS_CDS_DG_CHANGE_CDS.ZEHS_CDS_DG_CHANGEType/pres1_text">
    <Annotation Term="com.sap.vocabularies.UI.v1.Hidden"/>
    <Annotation Term="com.sap.vocabularies.Common.v1.FieldControl" EnumMember="Common.FieldControlType/Hidden"/>
</Annotations>
<Annotations Target="ZEHS_CDS_DG_CHANGE_CDS.ZEHS_CDS_DG_CHANGEType/flg_pgro">
    <Annotation Term="com.sap.vocabularies.UI.v1.Hidden"/>
    <Annotation Term="com.sap.vocabularies.Common.v1.FieldControl" EnumMember="Common.FieldControlType/Hidden"/>
</Annotations>
```

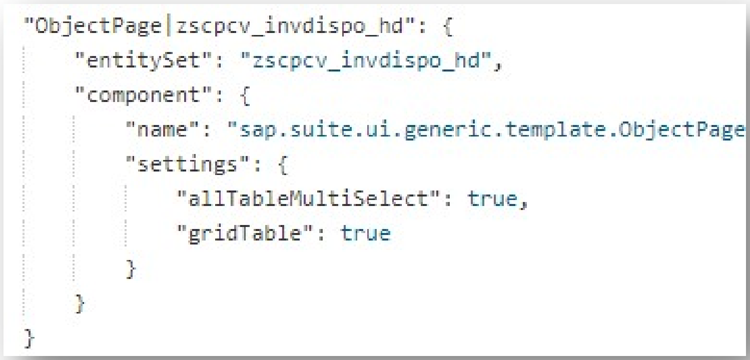
.

4. Extend list page to make any kind of change in the list page as per the customer requirement. i.e. Change the display format of data in Analytical table, UI table or to add any custom action.



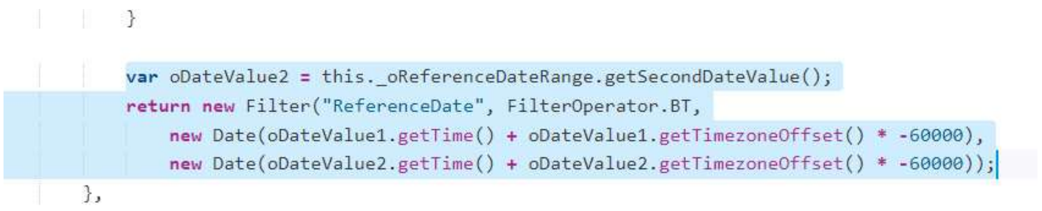
```json
"ListReport|zscpcv_invdispo_hd": {
    "entitySet": "zscpcv_invdispo_hd",
    "component": {
        "name": "sap.suite.ui.generic.template.ListReport",
        "list": true,
        "settings": {
            "filterSettings": {
                "dateSettings": {
                    "useDateRange": true
                }
            },
            "smartVariantManagement": true,
            "condensedTableLayout": true,
            "tableType": "GridTable"
        }
    },
```

5. Extend detail page to make any kind of change in the detail page as per the customer requirement. i.e. Change the display format of data in Analytical table, UI table or to add any custom action.



```
"ObjectPage|zscpcv_invdispo_hd": {
    "entitySet": "zscpcv_invdispo_hd",
    "component": {
        "name": "sap.suite.ui.generic.template.ObjectPage
        "settings": {
            "allTableMultiSelect": true,
            "gridTable": true
        }
    }
}
```

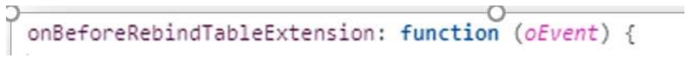
6. If there is any requirement to add custom filter please extend filter to add any custom filter in the filter bar.
7. If you add date in custom filter and your date format is ***Edm.DateTime***, please add ***Time Zone*** while generating date object.



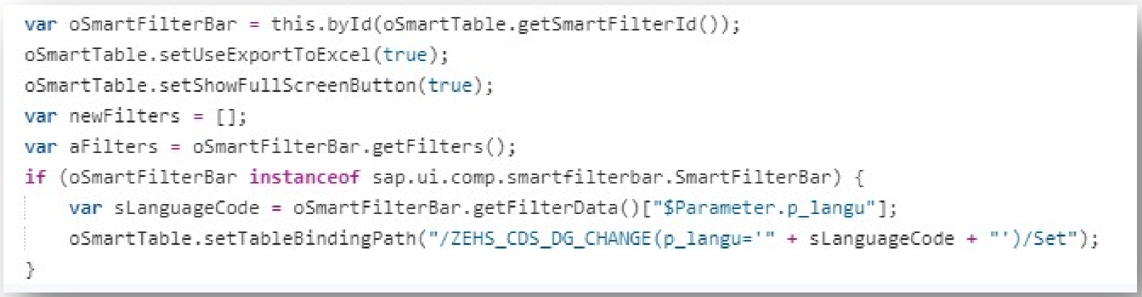
```
        }

        var oDateValue2 = this._oReferenceDateRange.getSecondDateValue();
        return new Filter("ReferenceDate", FilterOperator.BT,
            new Date(oDateValue1.getTime() + oDateValue1.getTimezoneOffset() * -60000),
            new Date(oDateValue2.getTime() + oDateValue2.getTimezoneOffset() * -60000));
    },
```

8. Whenever you want to pass date type as a parameter to any service, please use char8 type instead of **Edm.DateTime**.
9. Use ***onBeforeRebindTableExtension*** hook method to pass parameter into the parameterized service.



```
onBeforeRebindTableExtension: function (oEvent) {
```

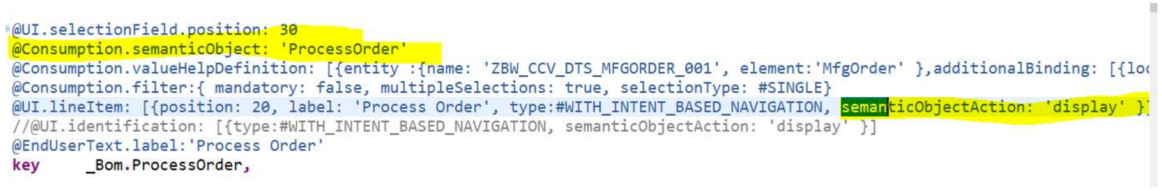


```
var oSmartFilterBar = this.byId(oSmartTable.getSmartFilterId());
oSmartTable.setUseExportToExcel(true);
oSmartTable.setShowFullScreenButton(true);
var newFilters = [];
var aFilters = oSmartFilterBar.getFilters();
if (oSmartFilterBar instanceof sap.ui.comp.smartfilterbar.SmartFilterBar) {
    var sLanguageCode = oSmartFilterBar.getFilterData()["$Parameter.p_langu"];
    oSmartTable.setTableBindingPath("/ZEHS_CDS_DG_CHANGE(p_langu='" + sLanguageCode + "')/Set");
}
```

10. Change/update all possible fields and values into manifest.json file first to change table format, remove unwanted navigation etc instead of writing code in the controller file. i.e. If details navigation is not required, please remove details navigation.

11. Add intent-based navigation annotation in the service to add hyperlink to any data for navigation to another application.

i.e. In CDS View, Semantic Object and Semantic Action is added.



```
@UI.selectionField.position: 30
@Consumption.semanticObject: 'ProcessOrder'
@Consumption.valueHelpDefinition: [{entity :{name: 'ZBW_CCV_DTS_MFGORDER_001', element:'MfgOrder' },additionalBinding: [{lo
@Consumption.filter:{ mandatory: false, multipleSelections: true, selectionType: #SINGLE}
@UI.lineItem: [{position: 20, label: 'Process Order', type:#WITH_INTENT_BASED_NAVIGATION, semanticObjectAction: 'display' }]
//@UI.identification: [{type:#WITH_INTENT_BASED_NAVIGATION, semanticObjectAction: 'display' }]
@EndUserText.label:'Process Order'
key     _Bom.ProcessOrder,
```

12. If there is any specific reason or specific requirement, we cannot add navigation annotation at service end, in that case we can add UI code to navigate to another application. i.e. If user want to navigate to only a particular application instead of related applications.

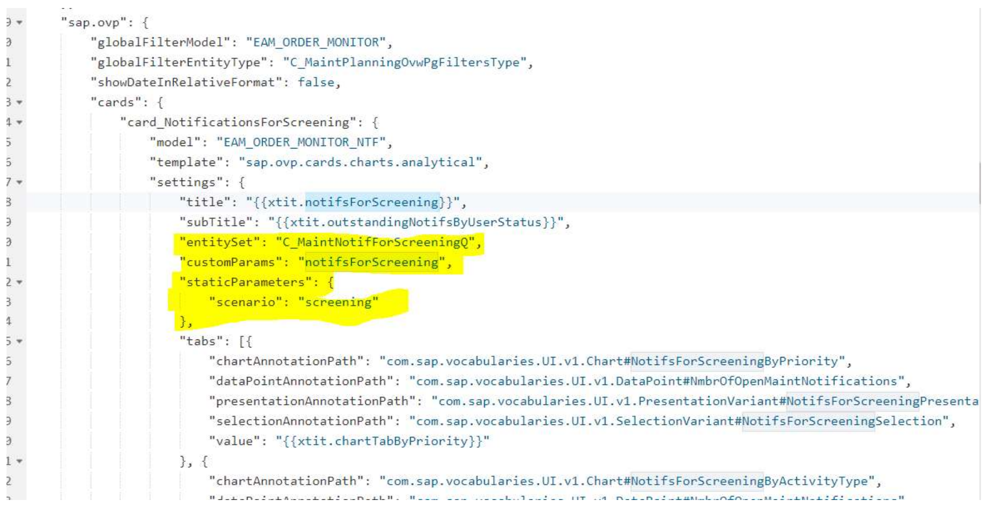
i.e. In FLP, if you have access to related application with same Semantic Object, you are given an option to navigate to those. If user don't want that facility and asked to show only one navigating application, then handle it from UI side.

13. For OVP, filter field name in all the services should be same, in case of development with cart. So that the selected filter will apply automatically in all the carts.

In OVP, you can multiple Entity Set for different carts. In OVP, one global filter is present, and it is applicable to all carts. If field names are not same, then unnecessary you need to handle through custom approach.

14. For OVP application, use same filter name in the target application otherwise you have to do custom coding for passing parameters to navigating application.

15. For OVP application, use annotation-based navigation to avoid parameter handling manually.



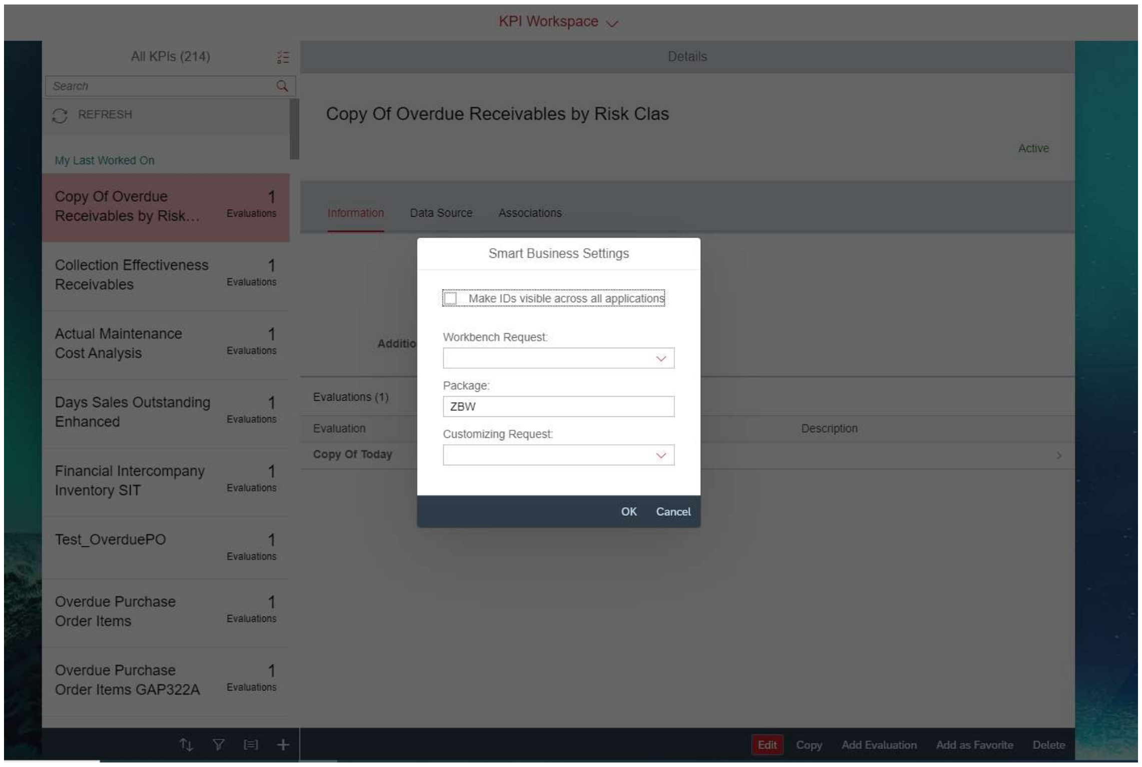
```
    "sap.ovp": {
        "globalFilterModel": "EAM_ORDER_MONITOR",
        "globalFilterEntityType": "C_MaintPlanningOvwPgFiltersType",
        "showDateInRelativeFormat": false,
        "cards": {
            "card_NotificationsForScreening": {
                "model": "EAM_ORDER_MONITOR_NTF",
                "template": "sap.ovp.cards.charts.analytical",
                "settings": {
                    "title": "{{xtit.notifsForScreening}}",
                    "subTitle": "{{xtit.outstandingNotifsByUserStatus}}",
                    "entitySet": "C_MaintNotifForScreeningQ",
                    "customParams": "notifsForScreening",
                    "staticParameters": {
                        "scenario": "screening"
                    },
                    "tabs": [{
                        "chartAnnotationPath": "com.sap.vocabularies.UI.v1.Chart#NotifsForScreeningByPriority",
                        "dataPointAnnotationPath": "com.sap.vocabularies.UI.v1.DataPoint#NmbrOfOpenMaintNotifications",
                        "presentationAnnotationPath": "com.sap.vocabularies.UI.v1.PresentationVariant#NotifsForScreeningPresenta
                        "selectionAnnotationPath": "com.sap.vocabularies.UI.v1.SelectionVariant#NotifsForScreeningSelection",
                        "value": "{{xtit.chartTabByPriority}}"
                    }, {
                        "chartAnnotationPath": "com.sap.vocabularies.UI.v1.Chart#NotifsForScreeningByActivityType",
```

Please check the standard app – EAM_ORD_MONS1 (1809 FS2) code for more details.

16. For OVP application, try to use standard cart as much as possible.

12 SAP Smart Business Application-KPI Workspace

1. Developer should have proper authorizations to work in Fiori Launchpad Designer as an administrator.
2. Developer should be assigned role and authorization for using the 'KPI Design' apps like KPI Workspace, Create KPI, Configure Drill-Down etc.
3. OData services SMART_BUSINESS_RUNTIME_SRV and SMART_BUSINESS_DESIGNTIME_SRV should be activated.
4. SICF node /sap/bc/ui5_ui5/sap/sbrt_appss1 should be activated.
5. Keep both the TRs customizing and workbench ready for both the system Gateway and development(backend) before starting this kind of application development. Please note, you need to move customizing TRs to each client of the system. As workbench TRs are client independent, no need to move to each client. But when you are moving the changes to further systems, need to move both the TRs and need to check if customizing changes if present in each client.



Copy Of Overdue Receivables by Risk Clas — Active

KPI	Evaluations
Copy Of Overdue Receivables by Risk…	1 Evaluations
Collection Effectiveness Receivables	1 Evaluations
Actual Maintenance Cost Analysis	1 Evaluations
Days Sales Outstanding Enhanced	1 Evaluations
Financial Intercompany Inventory SIT	1 Evaluations
Test_OverduePO	1 Evaluations
Overdue Purchase Order Items	1 Evaluations
Overdue Purchase Order Items GAP322A	1 Evaluations

Evaluations (1)

Evaluation	Description
Copy Of Today	

Smart Business Settings

☐ Make IDs visible across all applications

Workbench Request:

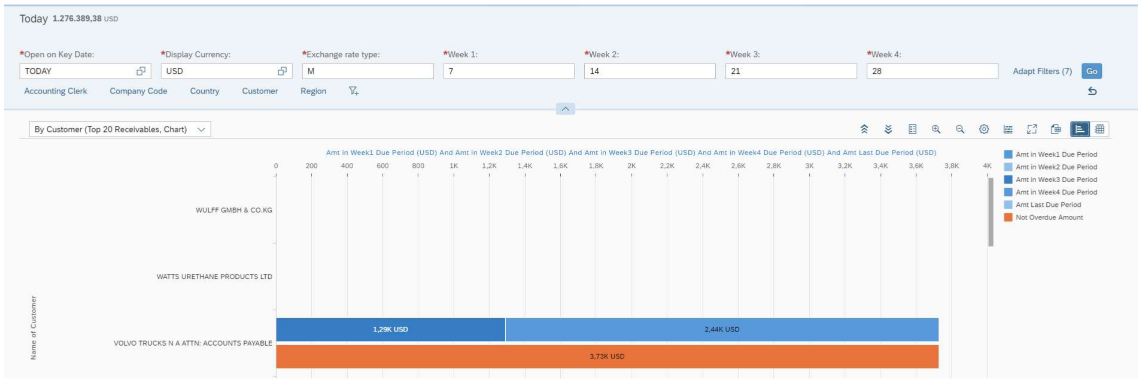
Package: ZBW

Customizing Request:

OK Cancel

6. Open Fiori Designer in ‘CUST’ scope and select the customizing transport request.

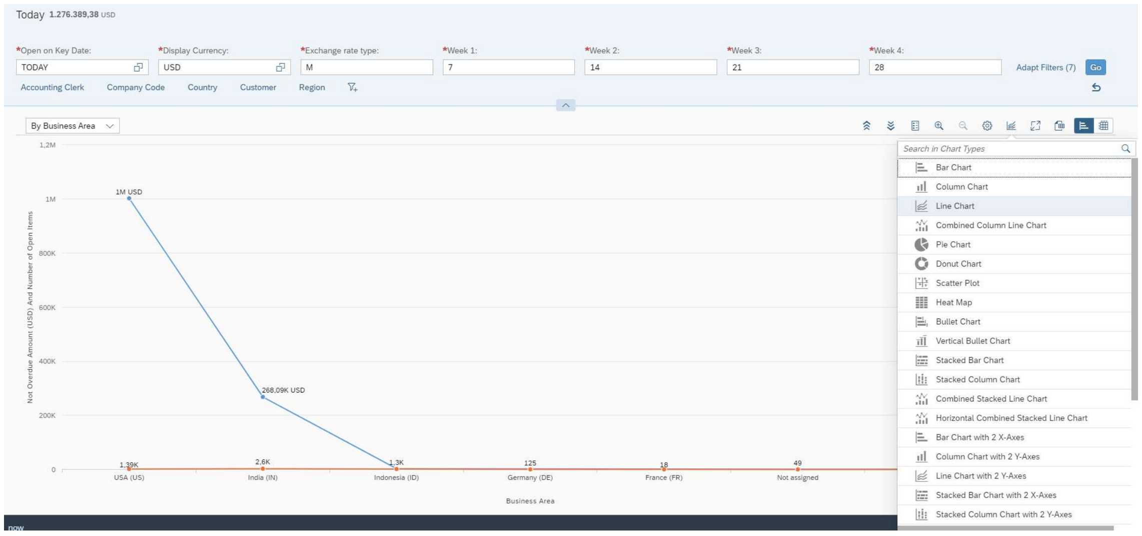
7. If there is requirement to create a report (non-transactional) with graphical interface, you can use SAP Smart Business Application development.



Today 1.276.389,38 USD

*Open on Key Date:	*Display Currency:	*Exchange rate type:	*Week 1:	*Week 2:	*Week 3:	*Week 4:
TODAY	USD	M	7	14	21	28

8. Multiple type of charts can be displayed in this approach with zero coding. No IDE is required in SAP Smart Business Application development. There is no requirement of service annotations.



Today 1.276.389,38 USD

*Open on Key Date: TODAY

*Display Currency: USD

*Exchange rate type: M

*Week 1: 7

*Week 2: 14

*Week 3: 21

*Week 4: 28

9. Extension/Enhancement: If there is any requirement to make some changes on standard application, copy the standard application and make the require changes on that.

10. Create separate catalog to add SAP Smart Business Application to avoid catalog Outdated error.

13 Troubleshooting commonly faced challenges

Troubleshooting

Troubleshooting Steps	Description	Possible Solution
Product Version	Product Version	Validate Product Version
Software Component Version	Software Component Version	Check on whether the front end and back end components are installed. Verification of the installation of Front-End / Back-End components from the corresponding section for the required Fiori-app. We verify in the SPAM transaction that the UIS4HOP1 200 UI component is installed for the Monitor Capacity Utilization Fiori application.
App Activation and Service Activation	SICF and ICF Node Activation	Verify in the SICF transaction that ICF nodes are active. We need to activate/validate the UI5 application in SICF. The default

		path for this is /default_host/sap/bc/ui5_ui5/sap/.
ODATA service activation	ODATA service activation	Verify in the OData service for the application is active.Use transaction /IWFND/MAINT_SERVICE to check details. The default path for this is /default_host/sap/opu/oData/sap/
Fiori-app catalog	Fiori-app catalog	Check whether business catalog/Technical catalog are available.
Roles	Roles	Check if the proper Roles assigned to the user/group. Use transaction PFCG to check detail if you have access to the Tcode.
Verifying in Launchpad Designer	Verifying in Launchpad Designer	In transaction /UI2/FLPD_CUST (Launchpad Designer), we can verify Fiori – group, and Fiori – catalog and Target Mapping, indicated in the IMPLEMENTATION INFORMATION section for the required Fiori application.
End-User Verification		In transaction SU01, we can add the created role to the end-user. Check detail if you have access to the Tcode.
App Verification in FLP	End-User Verification	In transaction /UI2/FLP (SAP Fiori Launchpad) we can check that the tiles of this Fiori application are available to the user.
Add system alias	Add system alias	Add system alias
List of navigating application	List of navigating application	Please check in business catalog for list of navigating applications
Troubleshooting		We can use transactions /IWFND/TRACES and /IWFND/ERROR_LOG

Challenges

Challenges/Issues	Description	Possible Solution
Troubleshooting	Check error	We can use transactions /IWFND/TRACES and /IWFND/ERROR_LOG

App could not be opened.	Failed to resolve navigation target "#XXXX-YYYY".	This is most likely caused by an incorrect SAP Fiori launchpad content configuration or by a missing role assignment. Assign the proper role to the user
No Authorization	No Authorization to access service or component	Assign require role
Cannot Load Tile	Cannot Load Tile	Add missing service, Add system alias
Unable to Load Groups	Unable to Load Groups	Clean cache
Missing Tile/App in FLP	Missing Tile/App in FLP	Assign require role, Readjust the role
Tile reference lost		Delete the tile add again and move TR
Global search in FLP not working		Add require authorization to service ESH_SEARCH_SRV
Catalog/Group outdated error		Click on outdated in fiori designer and make the require changes
Tile reference lost		Delete the tile add agait and move TR
Changes not reflecting in APP/FLP		Clear local cache and global cache(/UI2/INVALIDATE_GLOBAL_CACHES), execute app indexing(/UI5/APP_INDEX_CALCULATE)
Standard app is not working for any custom role		Standard app is not working for any custom role - Assign standard role and check
GUI and WebDynpro apps not working properly	TCode based application not working as expected	Get in touch with ABAP team
Wrong data in report		Assign to respective backend team
Duplicate data/record		Assign to respective backend team
Performance issue	App running Slow/Taking long time to load	Get in touch with Basis team
For GUI app search or buttons click is not working as expected		Please check in alternative browser i.e. IE, Microsoft Edge.
Authorization error		Please check SU53 log for a user in gateway and backend both the server
Data is not available for CDS based service		Please check authorization at backend system

ICF note is failed to activate		Please check the backend object
Standard transactions we cannot exposed as Tile		Few standard transactions (by coping reference from standard catalog) we cannot exposed as Tile specially those transaction which are having mandatory parameter
Some of the button not visible in application		Visibility of few buttons are based on authorization, please use list of standard authorization and find required application dependency
Few features are not available while exposing the transaction in FLP. But same is be available in backend		Please refer to the following SAP Note: 2709162 - MS Word editor not available for selection on WebGUI. One more known limitation of SAP GUI for HTML where office integration like excel file upload doesn't work
Catalog/Group outdated error		If you find outdated error in non-development system, please open the client and click on outdate. Issue will be resolved.

14 Guidelines for Commonly Used UI5 Controls

UI5 controls plays very vital role in any application. SAP has provided best practice guideline for each control. Here we have included few controls which are extensively used in applications.

14.1 *Table (sap.m.Table)*

A table contains a set of line items and usually comprises rows (with each row showing one-line item) and columns. Line items can contain data of any kind, but also interactive controls, for example, for editing the data, navigating, or triggering actions relating to the line item.

To display large amounts of data in tabular form, several table controls are provided. These are divided into two groups, each of which is defined by a consistent feature set:

Fully responsive tables

Desktop-centric tables

Guidelines

Use the responsive table if:

- A table is needed. The responsive table is the default table in SAP Fiori.
- The table content should be flexible and visually appealing. The **responsive table** offers the most flexibility in regard to its content because all SAPUI5 controls, and even

multiple controls, can be used. In addition, different rows can be based on different item templates.
- The focus lies on working on line items, not on individual cells.
- A main use case involves selecting one or more items, for which details are needed in order to choose the correct item.
- Line items are independent of each other and no operation across columns is needed.
- You want to have only one implementation for all devices.

Use the list if:

- You want to display a simple dataset.
- A table would be too complex.
- A list of actions is to be displayed.
- Simple two-level hierarchies are required (by using grouping or navigation).
- The main use case involves selecting one of several items with only a few details per item.
- A master list in a split-screen layout is needed.

Use the tree if

- You want to display a simple hierarchical dataset.
- You want to use a hierarchical master list in a split-screen layout.
- Using a tree table would be too complex.
- The main use case involves selecting one of several hierarchical items with only a few details per item.

Use the grid table if

- You need to display more than 1,000 rows at the same time.
- The cell level and the spatial relationship between cells are more important than the line item, such as if users need to recognize patterns in the data, like in waterfall charts.
- Comparing items is a major use case. The grid table layout remains stable irrespective of the screen width. In addition, a cell only ever contains one control.
- You need an analytical table, but you cannot provide an analytical binding.

Use the tree table if:

- Data needs to be displayed in a hierarchical manner.

Note that the tree table is not fully responsive. It is available only for desktops and tablets, so you will need to take an adaptive approach by offering an additional UI for smartphones.

Use the analytical table if:

- You need multilevel grouping as well as grand totals and subtotals.

Note that the analytical table is not fully responsive. It is available only for desktops and tablets, so you will need to take an adaptive approach by offering an additional UI for smartphones.

Use the smart table if:

- Any of the above tables should render themselves from the OData service, and almost no additional UI code should be written.

Note that the smart table supports only basic content layouts, unless you invest further in UI code. In addition, the smart table cannot be used within the chart container.

Do not use a table at all if:

- The main use case involves choosing one item from a very small number of items with no additional details. In this case, a select or combo box might be more suitable.
- You need an overview of large amounts of data. In this case, use a chart.
- You just need it for layout reasons. In this case, use a layout container, such as the grid layout.
- You need read-only or editable field-value pairs. In this case, use a form instead.

14.2 *Smart Table*

The smart table creates a responsive table, grid table, tree table, or analytical table based on an OData (Open Data Protocol) service and its annotations. The table toolbar comes with additional built-in features, such as personalization, export to spreadsheet, and variant management.

- Use Smart Table control only if you have OData version 2.
- Use the responsive table wherever possible.
- Don't use when you need more flexible content design.
- In-case of very complex data consider using chart instead of Smart table.
- Make sure that the sortProperty and the filterProperty are set (define p13nData via the aggregation CustomData). If you offer the export to spreadsheet option, ensure that it works as expected.
- Keep the number of initially visible columns to a minimum. Avoid pop-in behavior
- You can use the P13n dialog to let users show/hide the columns.
- Offer column totals by default for all columns where totals make sense
- If the page has a filter bar, don't offer filtering for the table
-

14.3 *Form / Simple Form / Smart Form*

(sap.ui.layout.form.Form | sap.ui.layout.form.SimpleForm | sap.ui.comp.smartform.SmartForm (smart form since version 1.36)

A form is used to present data to the user and to allow users to enter data in a structured way. The form acts as a container for other UI elements (such as labels, input fields, checkboxes, and sliders), while structuring these into a specific layout.

In SAPUI5, forms can be built using three different controls:

Form – sap.ui.layout.form.Form (API)

Simple form – sap.ui.layout.form.SimpleForm (API)

Smart form – sap.ui.comp.smartform.SmartForm (API)

With a **form**, you can easily layout a list of properties and input fields. A form is structured into form containers. Each form container consists of form elements. And each form element consists of a label and an input field.

The **simple form** control gives you the possibility to achieve the same result as with the form control, but in a much easier way. Inside a simple form, a form control is created along with its form containers and form elements:

The layout and structure are defined by the content that is entered.

Form containers and form elements are created automatically according to the content type.

A title (sap.ui.core.Title (API)) automatically starts a new form group (form container), and a label (sap.m.Label (API)) automatically starts a new row (form element).

All other controls following this label will be assigned to its row (form element).

The **smart form** control belongs to the new set of smart controls. Smart controls give developers the possibility to build UIs without the need to build a complex UI code (the term "smart" refers to the annotations that add semantics and structures to the provided data). The goals are to ensure design consistency, keep apps up to date with evolving design guidelines, and reduce the amount of front-end code for building SAP Fiori apps. Compared to the corresponding standard controls, however, the settings may be limited.

There are three types of forms:

1. Display-only: the data is presented only as label-value field pairs without editable fields.
2. Editable: the data is presented as label-input field pairs, so users can enter data.
3. Mixed: some fields are editable and some are not.

Guidelines

- Order the form logically from a user's perspective. For example, ask for a user´s name before asking them for their address.
- Group related information by using form and group titles.
- Try to arrange form groups (especially in size L and XL) in a way that the form:
- Is easy to read and understand.
- Does not contain too much white space (split groups if necessary).

- A label is not a help text. Give each field a meaningful label. Labels should be succinct, short and descriptive.
- If you have combined fields that contain, for example, a postal code and the name of a city, you can provide one combined label (postal code and city) for this group.
- The label of a required field is marked with an asterisk (*). There is a corresponding property in the API for this. Do not write the asterisk manually in the label text. Just use the corresponding property and the asterisk will be inserted automatically.
- At the end of the label, the form container automatically inserts a colon (:), which is triggered by the stylesheet. Do not write the colon manually in the label text.
- Use default settings for labels. (For example, labels are not supported for manual bold formatting.)
- Less is more: try to minimize the number of labels and their corresponding fields as much as possible.
- If an input element is in an error or warning state, provide a meaningful message for the user. There is a corresponding property valueStateText in the sap.m.Input API.

14.4 *Date*

This section describes the international rules for date formats. The SAPUI5 date formatters will help you to comply with these rules.

Types

You can use five different types of date formats: short, medium, long, full, and relative. The formatting and order of the values differ based on the locale settings that have been configured in the browser.

Short Format:

Dates in the short format are displayed as digits for the day, month, and year. Use the correct formatting to avoid errors.

English (US): 8/7/17
Spanish (ES): 7/8/17
Chinese (CN): 17/8/7
German (DE): 07.08.17

Medium Format

In general, you should opt to use the medium date format, which usually shows an abbreviation of the month as text.

English (US): Aug 7, 2017
Spanish (ES): 7 de ago. de 2017
Chinese (CN): 2017年8月7日
German (DE): 07.08.2017

Long Format

Use the long format if you need to display the full names of months. For some languages there may be no difference between the medium and long formats.

English (US): August 7, 2017
Spanish (ES): 7 de agosto de 2017
Chinese (CN): 2017年8月7日
German (DE): 7. August 2017

Full Format

If you need to display the day of the week, use the full date format.

English (US): Friday, August 7, 2017
Spanish (ES): viernes, 7 de agosto de 2017
Chinese (CN): 2017年8月7日星期五
German (DE): Freitag, 7. August 2017

Relative Format

If it suits your use case, such as ongoing events within a period of six days, use the relative format. Relative dates are displayed as text, for example, today, 6 days, and so on.

The default range for relative dates is between -6 and 6 days relative to the current date.

English (US): 2 days ago, yesterday, today, tomorrow, in 2 days
Spanish (ES): hace dos días, ayer, hoy, mañana, dentro de 2 días
Chinese (CN): 2 天前，昨天，今天，明天，2 天后
German (DE): vor 2 Tagen, gestern, heute, morgen, in 2 Tagen

Guidelines

Use date formatting if:

- You need to display dates based on the user's locale settings.
- The application is used in an international context

14.5 *Button*

Buttons allow users to trigger an action – either by clicking on or tapping the button, or by pressing certain keys such as Enter or the space bar.

Guidelines

- Don't combine an icon and text within one button.
- Choose a button text that is short and meaningful.
- Use imperative verbs for all actions; for example: Save, Cancel, Edit.
- Keep in mind that the text can be up to 300% longer in other languages.
- We don't recommend using tooltips since they are only visible on desktop devices. However, you can use tooltips for icon buttons.
- For icon buttons, make sure the default accessibility text for the icon is correct for your use case. If not, define app-specific accessibility text.
- If an action button is temporarily inactive, use the disabled status.
- If you need to show the number of items that will be affected by the action of the button, you can add the number in brackets. For example, Edit (3).

14.6 *Busy States*

Most of the time we need to show a busy state for on going background activity so that user can not perform modification.

You can set a busy indicator locally at control level (for example, on a page or for a button) using a busy state, or set it globally using the busy dialog. In SAP Fiori, the aim is to keep the blocking of UIs to a minimum, and to unblock areas where user interaction is possible. Because response time depends on available bandwidth and server performance, unblocking can take a second or more. In this case, we need to inform the user that the process is ongoing.

- Only use the busy dialog if you do not want to allow the user to perform any action in whole page.
- Use control's busy state to perform control specific activity blocking

- If multiple busy indicators overlap, the SAPUI5 framework ensures that only the one at the uppermost level is shown.
- Do not use the busy dialog for app or page loading. Set the busy state at app level.

14.7 *Barcode Scanning*

The general approach for dealing with native app capabilities is to use native user interfaces triggered by an SAPUI5 control whenever possible. Only use the barcode button to read barcodes. For OCR, RFID, and other scanning methods, use independent controllers.

- Fiori Client should not be used or proposed for mobile capabilities
- Usage of SAP approved 3rd party libraries for barcode scanning should be considered.
- Always open such kind of app in same Tab of browser. Opening App in different tab other than root can cause malfunctioning of Native feature
- Show the barcode button as a standalone button. Do not bundle it with input fields.
- On the button, show an icon. Do not show a text.

14.8 *PDF Viewer*

The PDF viewer control displays PDF documents within your app. It can be embedded in your page layout, or you can set it to open in a popup dialog. In addition, this control allows you to print and download the PDF documents it displays.

- To avoid the risk of performance issues, do not embed more than three instances of the PDF viewer per page.
- The embedded mode of the PDF viewer can be used on the Object Page if the container is as wide as the object page. If this is not the case, use the popup mode of the PDF viewer instead.
- Use a Flexbox container to wrap the embedded mode of the PDF viewer inside the application.
- To ensure accessibility options are supported, you need to have Adobe PDF Reader installed.

14.9 *Chart (VizFrame)*

Charts are used to visually represent how numeric values relate to each other. Therefore, it's crucial to define the type of relationship you want to illustrate when choosing the correct chart type. There are several types of charts available inside VizFrame) A typical use case may fall under below mentioned lists. As a developer it should chooses against the mentioned chart type

Uses	Chart Type
Rank items from highest to lowest, or vice versa.	Bar chart, column chart
Compare values of items in a list that has no particular order	Bar chart, stacked bar chart, bullet chart, heatmap
Show the variation of values over time.	Line chart, column chart, stacked column chart, bullet chart.
Display the contribution of individual values to the whole	Pie chart, bar chart, line chart, stacked bar chart
Show the deviation, difference, or gap between two sets of values	Bar chart, bullet chart, column chart, line chart.

Show the distribution within a set of values.	Bar chart, column chart, stacked column chart, line chart.
Show the correlation between two or three sets of values	Scatter chart, bubble chart, bar chart
Show the accumulation of successive values.	Waterfall chart

15 Reusability Concept

When developing SAPUI5 applications, the developer should have reusability in mind. This means reusing existing development such as generic components for new development as well as generalizing new development to enable it to be reused for other developments. The concept of reusability can be applied both within the scope of an application as well as across multiple applications.

Reusable Elements within the Application Scope
Within the scope of an application (or component), custom controllers and Fragments can be reused. Custom controllers can contain reusable business logic, whereas Fragments can contain reusable UI element definitions.

Reusable Elements Across Applications
Reuse of code across applications can be done through the concepts of Componentization and Modularization.

Componentization
Components are independent and reusable parts used in SAPUI5 applications. An application can use components from different locations from where the application is running. Thus, components can be developed by different development teams and be used in different projects. Components also support the encapsulation of closely related parts of an application into a particular component. This makes the structure of an application and its code easier to understand and to maintain. Generic components can be developed, which can be reused by different applications. The generic components can be made extensible through the use of extension points to allow for flexible reusability. The concept of Componentization means to compose applications based on building blocks of prebuilt generic components, which could be either UI components or faceless components.

Modularization
Modularization is the concept of splitting up comprehensive SAPUI5 applications into smaller parts. That means, instead of defining and loading one large bundle of JavaScript code, an application can be split into smaller parts which then can be loaded at runtime at the time when they are needed. These smaller individual files are called modules. Cross-App Modularization can be achieved through sharing of modules (JavaScript files) from a shared repository.

To summarize:
In-App reusability can be achieved through the use of Fragments and custom controllers.
Cross-App reusability can be achieved through sharing of components and modules (JavaScript files) from a shared repository.
Generic components can be implemented with extension points to allow for (flexible) reusability.

16 Testing

The application can be tested through the following

- SAP Systems: UI5 is quite flexible in terms of the server it needs for running the application. Provided the server has the required library, it can run on almost any SAP server, namely, SAP NetWeaver AS ABAP, SAP NetWeaver AS Java, SAP NetWeaver Cloud, SAP HANA XS, SAP Mobile Platform and open source platforms such as Apache, Tomcat, etc.

Although SAP provides the flexibility, it is recommended to do the testing in an environment closest to the eventual production system.

- Unit Test Case: Write QUnit test cases for testing individual modules. QUnit can be written in WebIDE. If eclipse is used, QUnit test cases need to run on local server.

- Use IBM Asset "SourceCodeAnalyzer" tool (WEBIDE Plugin) for Custom SAPUI5 application.

- Integration Testing: OPA5 is an API for SAPUI5 controls. It hides asynchronicity and eases access to SAPUI5 elements. This makes OPA especially helpful for testing user interactions, integration with SAPUI5 , navigation, and data binding.

- Behaviour-driven development (BDD): With behaviour-driven development (BDD), you as a developer start with a user story that defines the business value that the developed app should have. Next, you write a test that verifies the new functionality (this test initially fails). Finally, you write the needed functionality and your test passes. *Gherkin* is a test framework that supports this approach.

    - Gherkin: Gherkin is written in JavaScript and is fully compatible with SAPUI5, OPA, and QUnit. It is based on the "cucumber" tool.

17 Debugging

SAP UI5 being a "client" technology, runs on the browser itself. This means the debugging has to be done on the browser using the development tools of the browsers. Breakpoint can be set on the JavaScript code to check the run time flow and data of the view.

Usually, the SAP delivered UI5 apps uses minified version for performance reasons. In such cases add the sap-ui-debug=true parameter to the application URL. This would load the "regular" version making it easier to debug.

Sometimes, there could be a need to debug also the backend code i.e. Gateway or ABAP system. In such cases, an external breakpoint can be set on the backend ABAP system and this would make the debugging possible from the browser to the backend.

Apart from these UI5 Inspector is good tool debug and Analysis the UI page on browser, It a opensource Chrome plugin which gives more convenient way to debug and analyse the UI5 control.

Key features of UI5:

- Inspect SAPUI5 controls and review their properties, bindings, and data model
- Modify control properties on the fly and see how this affects rendering and behaviour

- Find relevant framework information for your SAPUI5 app

For more details on the debugging of UI5, OData or ABAP, please refer to the publicly available help sites on these technologies.

More Points:-
- Use Ctrl+Shift+Alt+'S' to enabling the Debug Sources options in respective SAPUI5 application when running in Google Chrome

18 Performance considerations

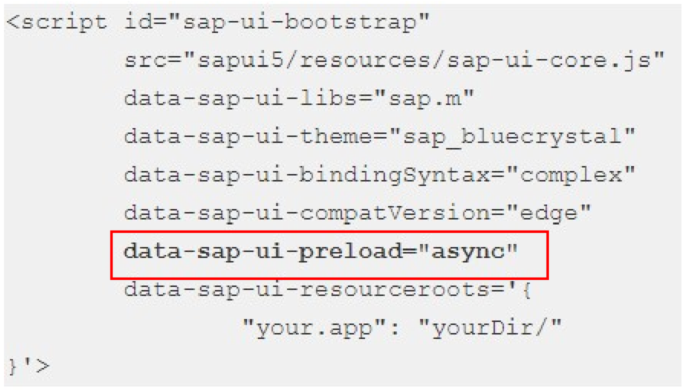
SAPUI5 apps are basically JavaScript files sent to a client by a server and interpreted by the browser. If a web app has performance issues, finding the cause can be both a time-consuming and nerve-consuming task. So, it's not only the coding of the app that can cause slow performance. It often turns out, for example, that the configuration is wrong. Slow networks or servers may also have a heavy impact on the performance of a web app. To help you avoid and solve performance issues in your app, here are some good practices while dealing with SAPUI5 apps.
- Enable Asynchronous Loading in the Bootstrap by setting parameter in bootstrap script data-sap-ui-async="true"
- Ensure that Root View and Routing are Configured to Load Targets Asynchronously
- Make Use of Asynchronous Module Loading by define
- Use manifest.json Instead of the Bootstrap to Define Dependencies
- Ensure that Library Preloads are Enabled
- Use the OData Model Preload
- Use OData Metadata Caching:To ensure fast loading times for SAP Fiori applications started from the SAP Fiori launchpad, the OData metadata is cached on the web browser using cache tokens. The tokens are added with the parameter sap-context-token to the URL of metadata requests. Please check via the developer tools of your browser (e.g. the Network tab in the Google Chrome developer tools) if the token has been appended to the request URL.
- Use a $select Query when Binding an Aggregation in the OData V2 Model: $select query you enable your application to fetch only necessary properties of an entity from the collection.
- Avoid the usage of setTimeout()
- Optimizing the application resources: Such as image file sizes which are heavy to load over network.
- Use Deferred batch call while dealing with data Services using model.
- Don't do frequent Dom query
- Check OData service performance using browser Dev tool and Use sap-statistic=true parameter to check OData service performance if there any response delay
- Cache dom object in variable if it is need more than one time, and then use it.
- Remove junk code
- Do not modify objects or values inside the model manually; always use the provided API to change data in the model, or use two-way binding
- Use sap-statistic=true parameter to check OData service performance if there any response delay
- Don't refresh model very frequently use only when necessary.
- Try to use OData navigation instead of $expand, As it adds cost in performance

18.1 *Asynchronous Module Definition*

Asynchronous Module Definition (AMD) syntax allows to clearly separate the module loading from the code execution and greatly improves the performance of the application. The browser can decide when and how the resources are loaded prior to code execution.

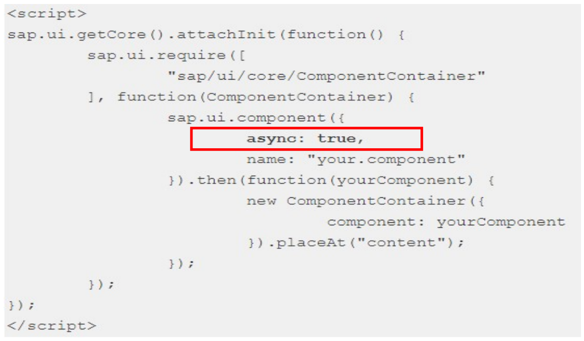
The following practices are recommended to ensure better performance when developing UI5 apps.

1. Config parameter in bootstrap – Preload



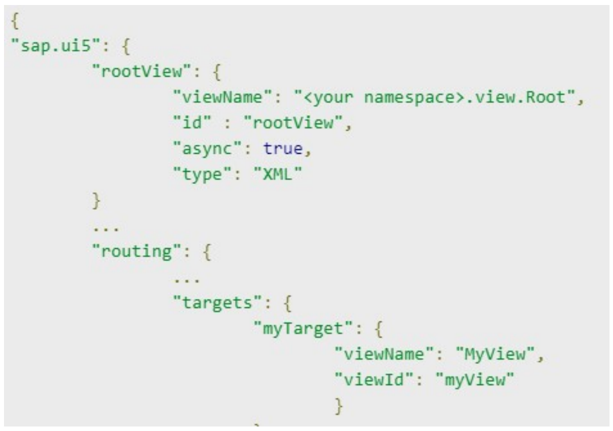
```
<script id="sap-ui-bootstrap"
        src="sapui5/resources/sap-ui-core.js"
        data-sap-ui-libs="sap.m"
        data-sap-ui-theme="sap_bluecrystal"
        data-sap-ui-bindingSyntax="complex"
        data-sap-ui-compatVersion="edge"
        data-sap-ui-preload="async"
        data-sap-ui-resourceroots='{
                "your.app": "yourDir/"
}'>
```

2. Asynchronous loading of component



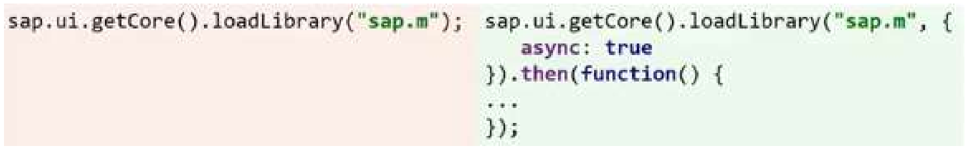
```
<script>
sap.ui.getCore().attachInit(function() {
        sap.ui.require([
                "sap/ui/core/ComponentContainer"
        ], function(ComponentContainer) {
                sap.ui.component({
                        async: true,
                        name: "your.component"
                }).then(function(yourComponent) {
                        new ComponentContainer({
                                component: yourComponent
                        }).placeAt("content");
                });
        });
});
</script>
```

3. Asynchronous loading of view in manifest.json



```
{
"sap.ui5": {
        "rootView": {
                "viewName": "<your namespace>.view.Root",
                "id" : "rootView",
                "async": true,
                "type": "XML"
        }
        ...
        "routing": {
                ...
                "targets": {
                        "myTarget": {
                                "viewName": "MyView",
                                "viewId": "myView"
                        }
```

4. Asynchronous loading of Library



```
sap.ui.getCore().loadLibrary("sap.m");
```

```
sap.ui.getCore().loadLibrary("sap.m", {
    async: true
}).then(function() {
...
});
```

19 Transport and Versioning

Before starting the development, please make sure that the backend SAP system hosting the UI5 applications has the proper version controlling Add-on installed.

1. Create a common package (ZUX*) for all UX related changes.
2. Use separate TR for each custom SAPUI5 application.
3. Activate ODATA with same TR of custom SAPUI5 application.
4. Which you want to transport, don't keep in $TMP package.

19.1 *Using Git Repository*

Git is the most used source code repository and version control system and is recommended to be used for SAPUI5 developments including Standard Fiori extensions. Following set of rules must be adhered to use it efficiently.

- Use Branches: Master branch should be used for day to day work. Always create a branch and work on your own branch. Make sure to merge from other branches when you are collaborating with a team and one the complete code is tested then only merge it to master.
  The GIT repository of a project should contain the following branches to optimally leverage the advantages of the GIT branching capabilities.
    - **Master**

The core branch that all other branches are sourced upon, represents the commit that is in production

- **Develop**

First child of tier main and used to support development teams feature development and release cycles

- **WO-N**

The purpose of a Work order branches is to isolate code to allow for well-controlled development cycle, this branch will be more volatile than the Master branch

- **Sustain**

For production issues where the manipulation of the current release version of the code is required, forked off of Master

- **Sustain Release**

A set of production defects that are ready for testing and have a release window. This branch is a child of the Sustain branch. Once a release version has been push to production then create a pull request to merge this branch with Master.

- **Release**

A set features that are determined to be ready for release forked off the develop branch. Once a release version has been push to production then create a pull request to merge this branch with Master.

- Commit Related Changes: fixing two different bugs should produce two separate commits
- Committing often keeps your commits small and, again, helps you commit only related changes
- Commit messages should be clear and meaningful so that the version history can be easily identified of what change has been made.
- Update **.**gitignore file to avoid unwanted or temporary file to be pushed into repository
- Don't git push straight to master
- Don't leak secrets into source control: Secrets, or secret keys or secret credentials, include things like account passwords, API keys, private tokens, and SSH keys. You should not check them into your source code.
- Always have the most current version of your project in the master repository. Before working on new features

20 Securing SAPUI5 Applications

SAPUI5 is a client-side JavaScript library, so while the library itself is designed and tested to be secure, it cannot ensure the application to be secure. Unlike WebDynpro, where the application is built against an abstract programming model and the framework handles the HTML rendering, JavaScript code and communication with the browser, in SAPUI5 the application controls the HTML output, it provides own JavaScript code which is executed on the client and it handles client/server communication.

While this brings a lot of freedom and possibilities for the application, it comes with a lot of responsibility with regards to security. Application developers need to understand the security threats and actively prohibit exploitation. Also, the correct configuration of the HTTP server, which is used, is important.

This also means common security mechanisms, which are taken for granted, like user authentication, session handling, authorization handling, or encryption are not part of SAPUI5 and need to be handled by the server-side framework and/or custom code of the application.

Different areas where we can secure the application:

- Browser Security
- Transport Security
- Server Security
- Third-Party Libraries
- Secure Programming Guide

- Security Concepts

Please check the below reference link for more details:
https://sapui5.hana.ondemand.com/sdk/#docs/guide/91f3d8706f4d1014b6dd926db0e91070.html

SAP UI5 does **NOT** handle the below aspects of the application
- User Authentication
- Session Handling
- Authorization handling
- Encryption, etc.

These must be handled either by the server and/or by writing custom code.

21 Review Check Points

Focus Area	**Concern About**	**Comment**
MVC	Maintainability	Do not put view logic in the controllers and do not put controller and model logic in the views.
		The event handlers, which is part of controller logic, should go in the controller.
Components	Stability	Use Component.js as the root component serving as the entry point for any application.
		The allows the UI5 component to be reused later.
Models	Performance	Do not overload the client models - JSON and XML – And do not add user data to client models.
Models	Security	Use SAP OData based data sources to access server data.
		OData is the standard and preferred data access protocol for SAPUI5. SAPUI5 provides built-in session security for the oData model. SOAP web services is an example of Non OData based server data access.
Views	Maintainability	Use XML views. JSON and HTML views should be avoided entirely.
		Although UI5 gives the flexibility to use several view types, XML views are recommended for the following reasons:
		• It helps following MVC pattern by not allowing any controller method within the view itself

		• Adopted by standard Fiori apps, making it consistent
		• Its declarative, which makes it look closer to the final HTML
		Use of JavaScript views is allowed only on approval explaining why an XML view cannot be used.
Views	Maintainability	Use Formatter functions in custom controllers to format data in XML views
Views	Stability	The UI Elements of a view must only be manipulated by the view's own view controller. Event listeners can be used to trigger manipulation or update of UI elements from other controllers.
Views	Stability	Do not use the createContent function in the Component to instantiate the root (XML) view, unless dynamic view enhancement is required. Instead specify the view using the rootView metadata parameter.
Views	Stability	Keep the root view as simple as possible, ideally with a single container control (for example a SplitApp). Fill the container of the root view with UI content defined in separate views.
Views	Stability	Implement simple client side validation (such as type check, number range check, etc.) for UI Controls before sending data to the backend.
		This is to minimize server round trips and ensure data consistency.
Views	Stability	Do not use the sap.ui.core.HTML in the views. Request approval for if HTML is required.
		If approved, use XHTML elements to embed HTML in the views.
Controllers	Maintainability	Use custom controllers for reusable or generic functions.
Data Binding	Stability	Do not use 2-way bindings for OData models.
		2-way data binding for OData models is currently an experimental feature from SAP. Until this becomes a permanent feature, Use 1-way binding and let the application manage the write operations through explicit calls to the ODataModel methods for Create, Update and Delete.
Bootstrapping and Initialization	Stability	Use enforced check of duplicate UI element ID's in the bootstrap. (data-sap-ui-noDuplicateIds="true").
		This is to prevent unintentional use of duplicate UI element ID's which could lead to errors.

Bootstrapping and Initialization	Maintainability	The name of the root HTML file should always be index.html
		Index.html should only Include the header, SAPUI5 bootstrap and application startup (attachInit).
Navigation and Routing	Stability	Do not use the EventBus for navigation. Use the Routing mechanism for navigation within an SAPUI5 application.
Event Handling	Stability	Use the Event Bus to handle non-navigation events and to pass information between views. For navigation events use the Router mechanism.
Navigation and Routing	Usability	Include a 'catchall' route in the routing configuration.
		This is to ensure that a sensible message is shown if navigation to an invalid resource is attempted.
Fragments	Maintainability	Use Fragments to define reusable view elements that may be used in several views.
		Use Fragments to break down complex views into smaller blocks, thereby decreasing complexity.
Fragments	Stability	Always use the FragmentDefinition element as a container in the fragment definition file (even if not syntactically required).
		This allows modifications and later additions of additional root elements to the fragment.
Fragments	Stability	Specify a unique fragment ID when instantiating the fragment.
		This is to avoid duplicate IDs when using a fragment multiple times within a view.
Unique ID's	Stability	Use own defined ID’s for UI elements only if really needed.
Dialogs and Popups	Maintainability	Always use Fragments to define dialogs and other pop-up controls
UI Controls	Stability	In the XML view, always explicitly specify a UI component's default aggregation rather than implying it.
UI Controls	Maintainability	Do not create and use custom controls. If needed, request an Approval

		The standard controls contains encoding to protect from cross site scripting issues, which the custom control may lack.
CSS and Theming	Usability	Use Project approved themes
CSS and Theming	Usability	Do not use local style sheets or adapt the standard theme without approval.
General Coding Practices	Stability	Do not use global variables in your code.
General Coding Practices	Performance	Use appropriate hook functions (onInit, onBeforeRendering, onAfterRendering, onExit) to add the required logic.
General Coding Practices	Performance	Do not call *.byId(someId) several times in a row, for the same ID.
		Store the result in a local variable to avoid repeated element searches.
Libraries	Stability	Ensure version consistency. The UI5 core libraries version of the development environment should be same as the server it is going to be executed on.
		Add a component descriptor to every developed SAPUI5 application and component. The component descriptor should be a JSON-file named "Component.json" placed in the same folder as the Component.js file. Add the design-time metadata, including the versioning information, to the component descriptor.
Libraries	Stability	The controls/API's used in the application should be based on the library version of the production server application. Any API/Control that is not available, is deprecated or experimental in the production environment should not be used.
Libraries	Maintainability	Use only SAP delivered libraries. Avoid the use of third party library unless it is an approved one.
Libraries	Stability	Do not mix Mobile and "Desktop" libraries within a view.
		The application should use sap.m library to enable the built-in responsiveness.
File Names and Locations	Maintainability	Follow proper project structure and file naming convention. Refer to section 1.6 for details.
Namespaces	Stability	All the objects (eg views) within the application should be created and used with same namespace.

Modularization	Maintainability	Split larger SAPUI5 developments into multiple modules and/or components.
		By default, modules are loaded on demand (when they are needed) and synchronously (the loading of the module is completed before any subsequent script is executed).
Performance Considerations	Performance	Have at most 2 sequential round trips per user interaction step.
Performance Considerations	Performance	Enable the Cache Buster mechanism to cache resources on the client side and the Application Cache buster to cache application files.
Extensions	Maintainability	When modifying custom applications or components which are either of a generic nature or used for multiple purposes, aim to insert extension points in the original application and implement the modification as an extension rather than modifying the original application.
Screen Adaptation	Usability	Use a Device Model (client side JSON model) to store information about the device.
		This is to facilitate responsiveness in the UI.
Screen Adaptation	Runtime Errors	Use One Way binding mode for the Device Model.
		This will prevent unintentional modification of data.
Screen Adaptation	Usability	Create separate views for mobile and desktop devices. Manage device switching during the application initialization.
Image Handling	Performance	Don't use generic Images within the application. Instead use a img model that has the links to the images stored on the server
Image Handling	Performance	Only use Image Formats supporting compression e.g. png, .jpg, .gif depending on the features required. Avoid using "oversized" images.
Text and Message Handling	Usability	Use only ResourceModel for any text or message to be displayed in the application even if internationalization is not required. Also, do not use resource bundles directly within the UI control.
		Use only one global properties file per language that contains all texts used in the application.
		Use of resource model allows using translation keys instead of (hard-coded) actual texts. The use of resource bundle directly means extra processing to retrieve any text.

Text and Message Handling	Usability	Follow the below guidelines on notifications and error messages in UI5 applications
		• Provide short and crisp error messages to the user.
		• A message should always contain a 'Call for Action'.
		• To achieve the above you need to map error messages from a backend system.
		• Focus on the most common error situations and improve the messages there.
		• For the rest simply state that a 'Backend' or 'Database' error has occurred. The user cannot handle this anyway and no further details are required.
		• It is a must to detect all problems related to network connectivity and indicate them as such.
Browser Compatibility	Usability	Applications must be tested and fully functioning in a Project approved browser
Development Environment	Development Process	Do not use upload/download practice for developing UI5 application. The development environment should be linked to the backend system hosting the UI5 code. The team provider addon has all the necessary features required by the developer

22 Libraries and References

To understand the libraries available for coding in UI5, please refer to SAP documentation

Fiori design guidelines
https://experience.sap.com/fiori-design-web/

sap.m.table
https://experience.sap.com/fiori-design-web/table-overview/

sap.m.form
https://experience.sap.com/fiori-design-web/form/

API Reference
https://sapui5.hana.ondemand.com/sdk/#docs/api/symbols/sap.ui.html

Version Info
https://sapui5.hana.ondemand.com/sdk/#versioninfo.html

Since the API’s in UI5 change relatively faster, special attention is needed before using them. Version compatibility flags may be added to the bootstrap to ensure the application’s compatibility with the runtime libraries.

SAP aims for SAPUI5 to be a Release Independent Component (RIC). This means that while the API will evolve over time, backward compatibility will be retained within a major release. Incompatible API changes will only be implemented with new major releases. If any feature changes its behaviour between two minor releases, SAP will introduce a compatibility version flag to allow the application to control whether to apply the new or the old behaviour.