Contents

[1. Abstract 1](#_Toc118981962)

[2. Basic capabilities of the Tool 2](#_Toc118981963)

[3. Quality aspects of the tool 2](#_Toc118981964)

[4. IT Architecture of the Recipe App 3](#_Toc118981965)

[5. Cap Model 3](#_Toc118981966)

[6. Define the OData service API endpoint in ui5.yaml file 4](#_Toc118981967)

[7. Scripts and dependencies in Package.json 5](#_Toc118981968)

[8. Define the models in Manifest.json 9](#_Toc118981969)

[9. Define the root view in Manifest.json 10](#_Toc118981970)

[10. Define the router in Manifest.json 11](#_Toc118981971)

[11. Index.html 12](#_Toc118981972)

[12. Component.js 13](#_Toc118981973)

[13. Program flow 14](#_Toc118981974)

[14. App.view.xml 14](#_Toc118981975)

[15. App.controller.js 15](#_Toc118981976)

[16. Login 16](#_Toc118981977)

[17. Create a recipe 21](#_Toc118981978)

[18. Display a list of items 24](#_Toc118981979)

[19. Display an Item 24](#_Toc118981980)

[20. Edit an Item 25](#_Toc118981981)

[21. Save an Item 25](#_Toc118981982)

[22. Delete an Item 25](#_Toc118981983)

[23. Copy an Item 25](#_Toc118981984)

[24. Display Hierarchy of items: 25](#_Toc118981985)

# Abstract

The ‘Crunchy Crowd’ Recipe web application is used to create and manage the recipes. The user creates the recipe with the list of ingredients and method of preparation. User can edit and clone a recipe. Only the author of the recipe can be able to edit/delete the recipe. The other users can only be able to view the recipe or clone the recipe. Value addition of the developed Recipe App enables the user to create different variants of recipes. The recipe relationships can be visualized through table and Tree view. The recipe variants can be compared to determine the key changes of the recipe. Through this web application, the user can be able to change the recipe instantly based on the taste buds of the customer.

# Basic capabilities of the Tool

1. Light weight Login functionality
2. Create a Recipe
3. Display a list of Recipes
4. Filter functionality to filter the recipes
5. Detail page about the Recipe
6. Edit and Save Recipe to database
7. Inbuilt Authorization
8. Delete a Recipe
9. Clone/Copy a Recipe
10. Display the relationship between Recipes
11. Compare the Recipes

# Quality aspects of the tool

* 1. Validation of Recipe data during creation of Recipe and while editing the recipe
     1. Mandatory fields check
     2. total time validation
     3. image validation
     4. minimum ingredient validation
     5. minimum prep step validation
  2. UI/UX:
     1. Display only the relevant data in the Detail page of Recipe. Remove Add/Delete.
     2. Enabling and displaying the controls based on user actions
  3. Standards in Text formatting and display
  4. Clean Coding principles such as separating the functionalities like adding an item/Deleting an item
  5. Following MVC principle by setting the model in the controller and accessing the UI Controls through the model
  6. Maintaining i18n Model to display text in Message boxes and Control texts
  7. Inbuilt Authorization: Only the author of the recipe can be able to edit/delete a recipe
  8. project leverage CI/CD for the backend repo through scanning the code for vulnerabilities by MEND and the deployment in our cloud foundry space

# IT Architecture of the Recipe App

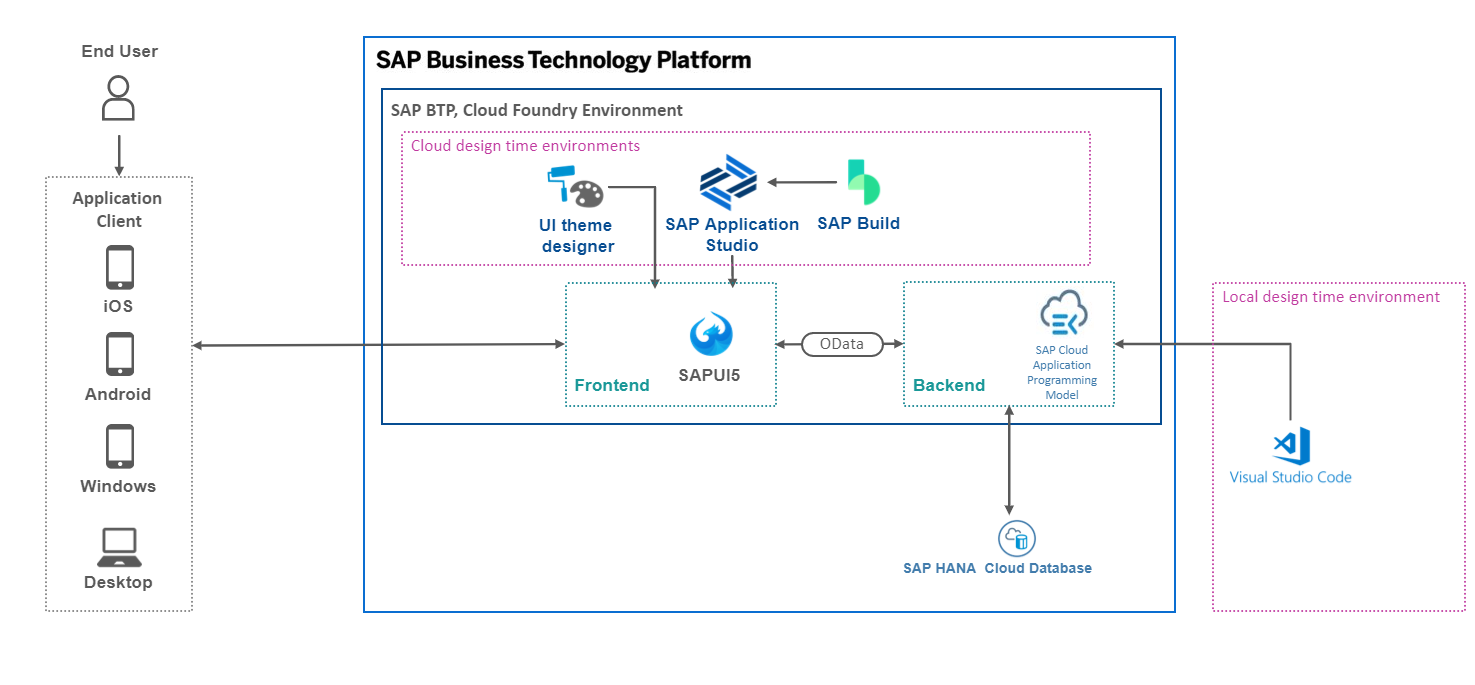


Figure IT Architecture

# Cap Model

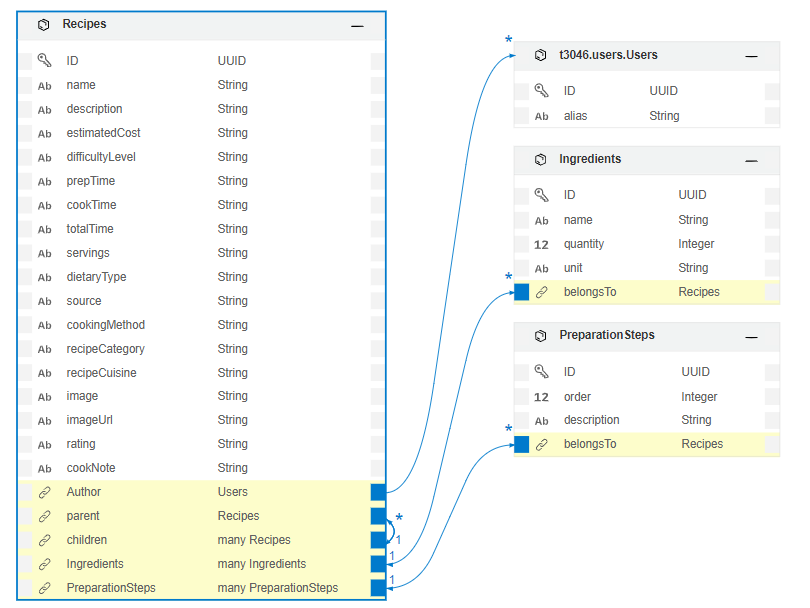


Figure CAP Data model in the backend

# Define the OData service API endpoint in ui5.yaml file

The UI5 Tooling Configuration must be done in ui5.yaml file located in the root directory as shown in Figure 3 Root Directory.

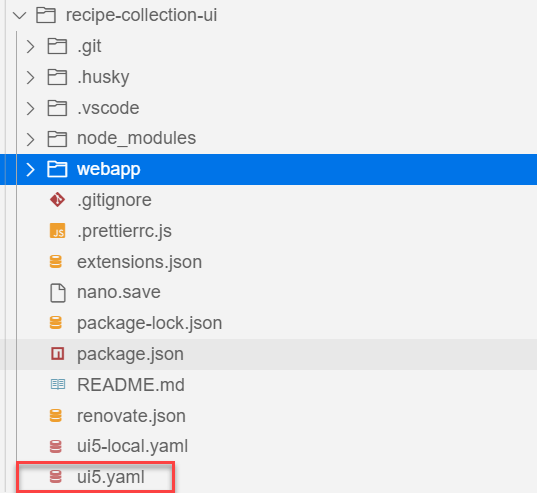


Figure Root Directory

The key parameters which needs to be set with values are type of project, name of the project and basic configurations for the backend. The name of the project defined is ‘sap.ui.ic2022’ and the type of the project can be application, library, theme-library or module. The type defined in this example is application. The backend parameters are defined under server->customMiddleware->configuration->backend. The parameters are path and url as shown in Figure 4 Configuration of UI5 yaml file.

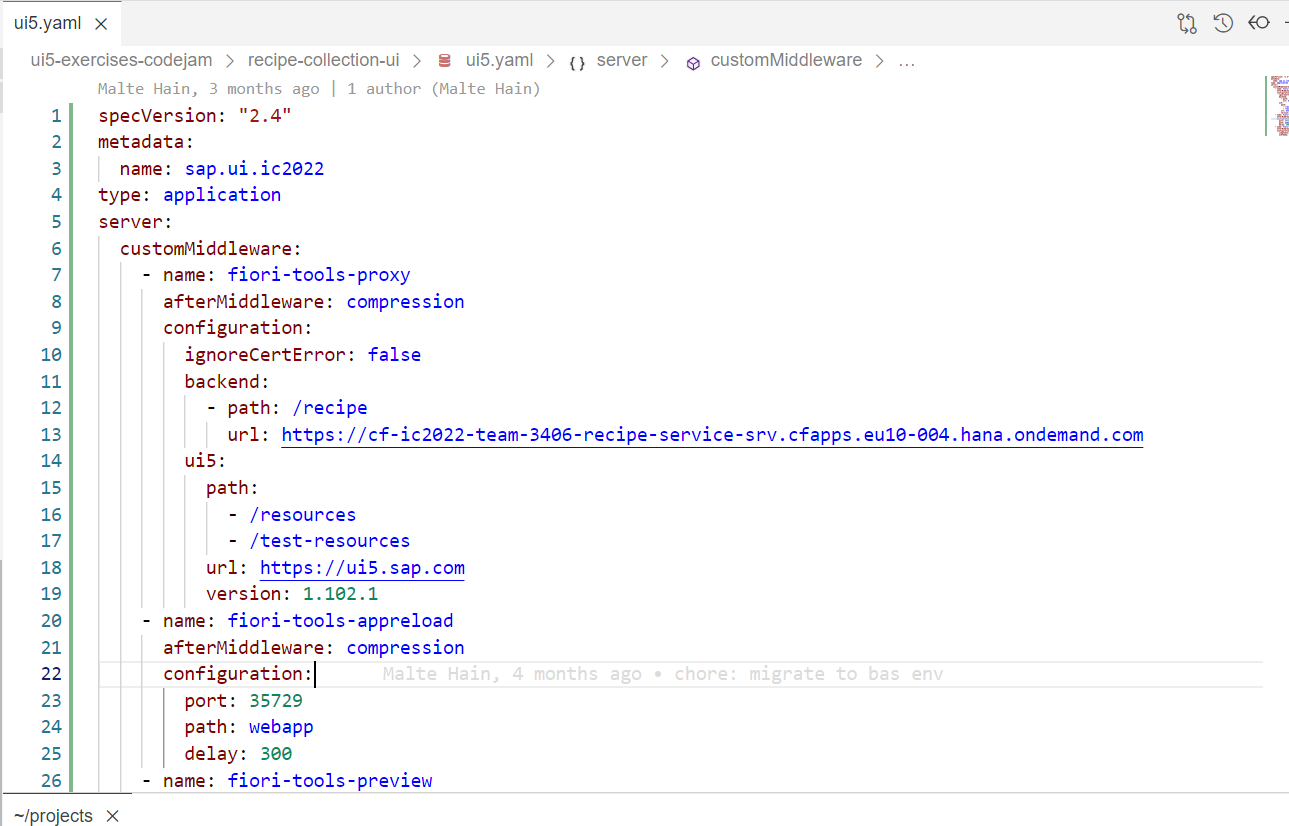


Figure Configuration of UI5 yaml file

Url: the url highlighted in the picture above take us to the Odata service url. We can see the different entities created for the OData service as shown in Figure 2 Entities.

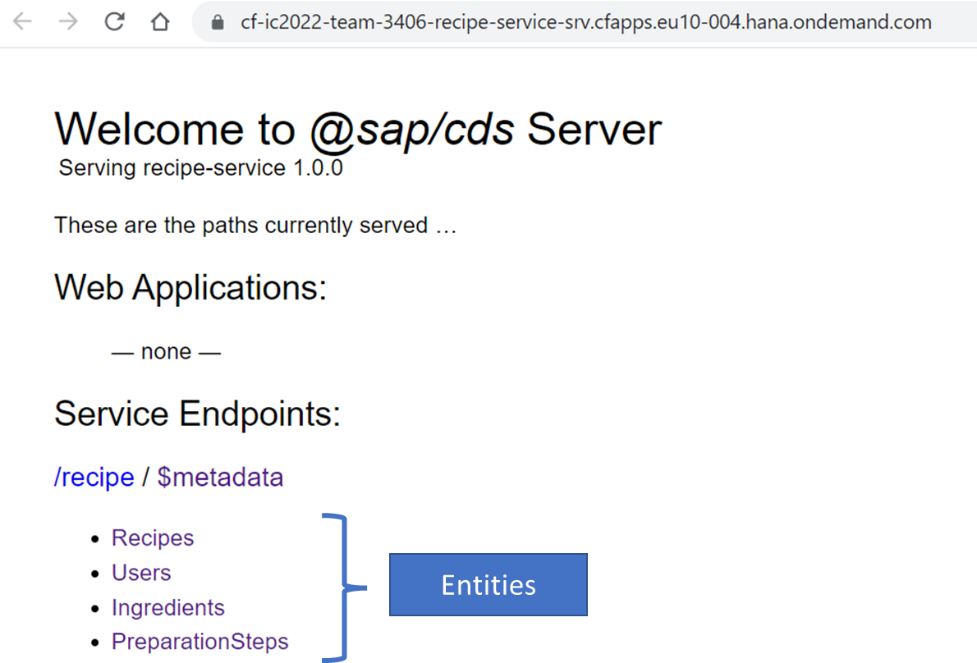


Figure Entities

By clicking further on the Recipes entities, it will open the webpage with the list of all data that belong to the Recipes entity. It is the list of recipes created.

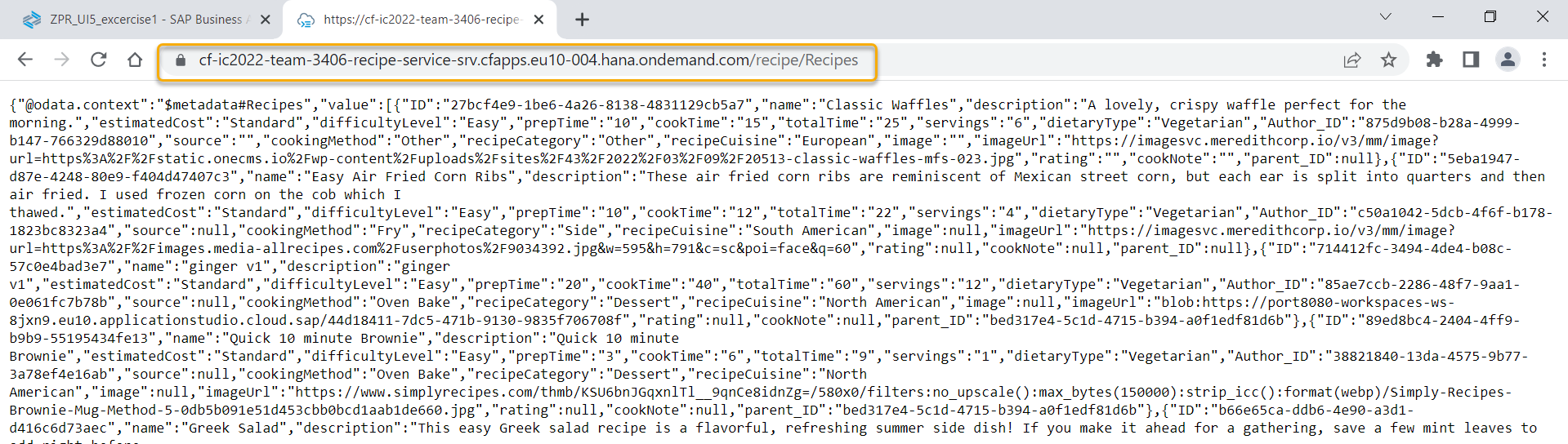


Figure List of created recipes under Recipes Entity

# Scripts and dependencies in Package.json

The package.json file must include the scripts and dependencies. If the scripts and dependencies are not defined properly in package.json file, then errors such as ‘middlewareRepository: Unknown Middleware fiori-tools-proxy’ will occur. Please also ensure that the file location and file types are properly mentioned in scripts. For example “start” script has the following line of code. It mentions about the index.html page which is the starting page of the SAP UI5 Application."start": "fiori run --open \"/index.html?sap-ui-xx-viewCache=false\"",

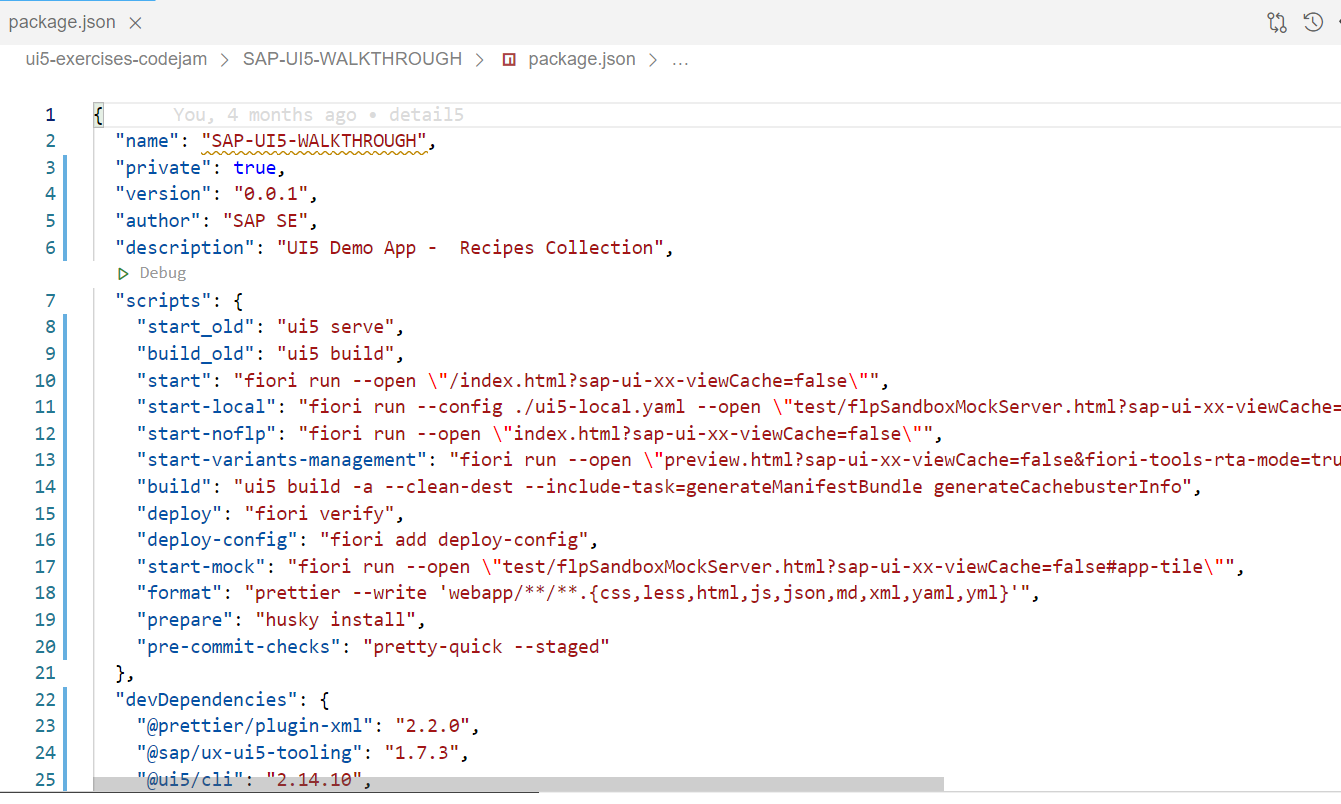


Figure Scripts in package.json

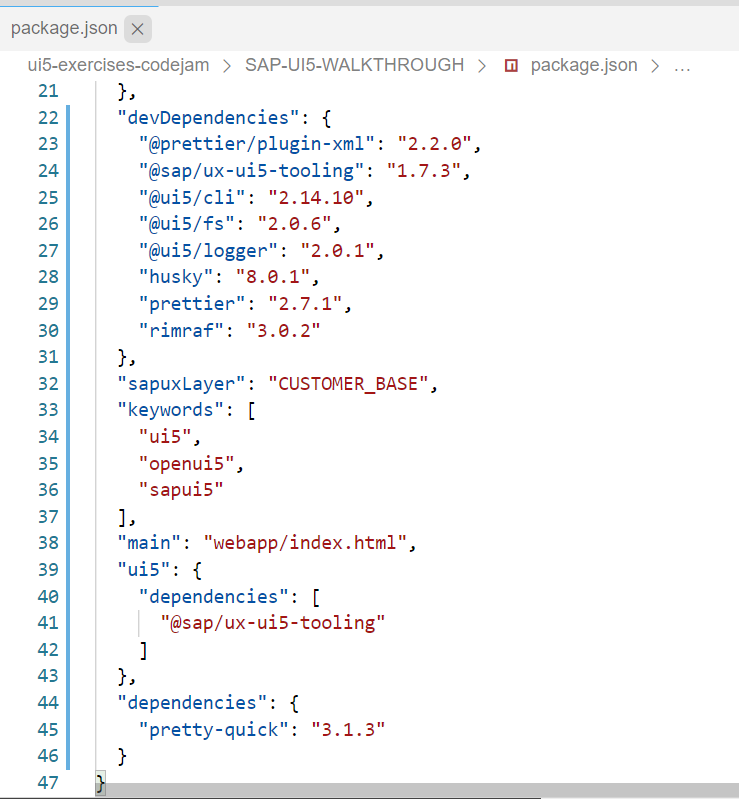


Figure devDependencies in Package.json file

**Exercise1:** What will happen if the scripts of package.json is not complete. For example, the script “start” is ignored as shown in below figure.

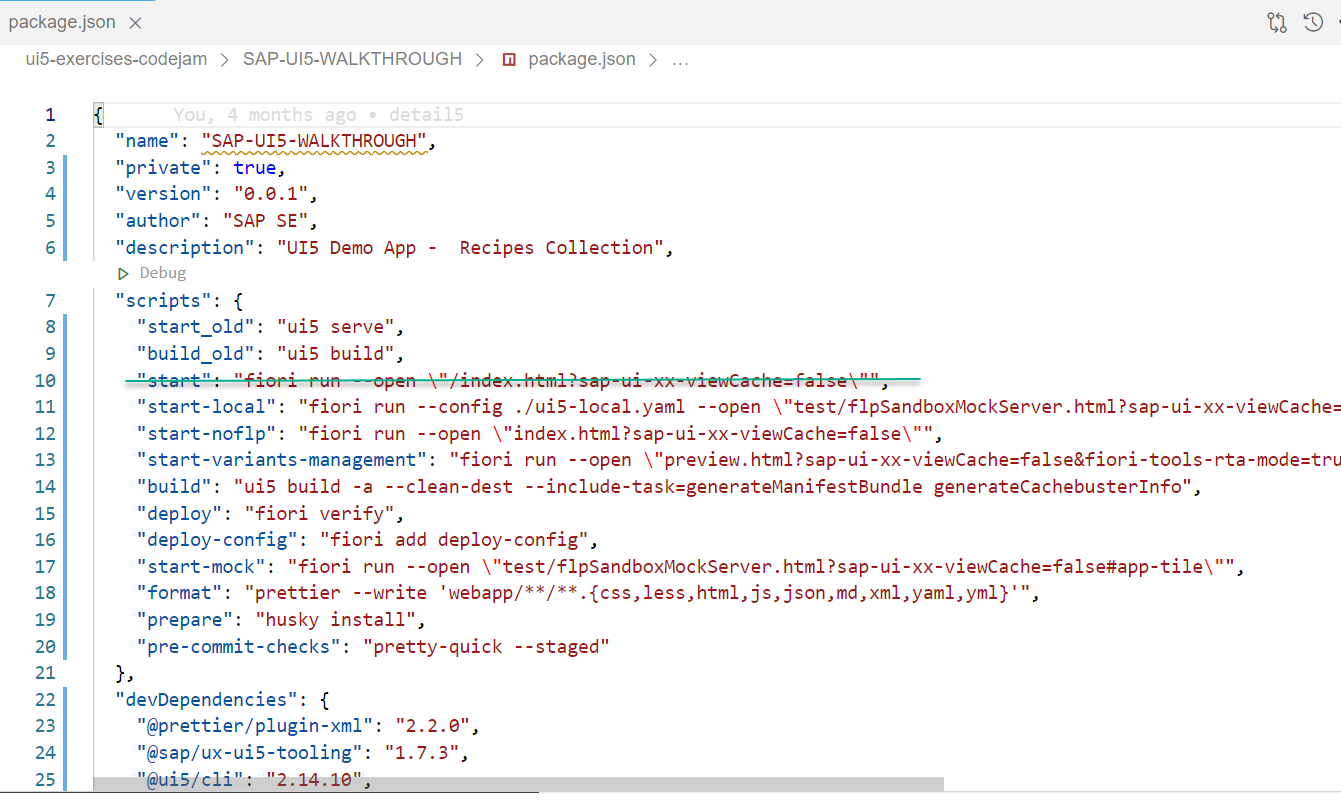


Figure Start script is removed from package.json

**Answer:** command ‘npm start’ in the terminal will result in error as shown below.

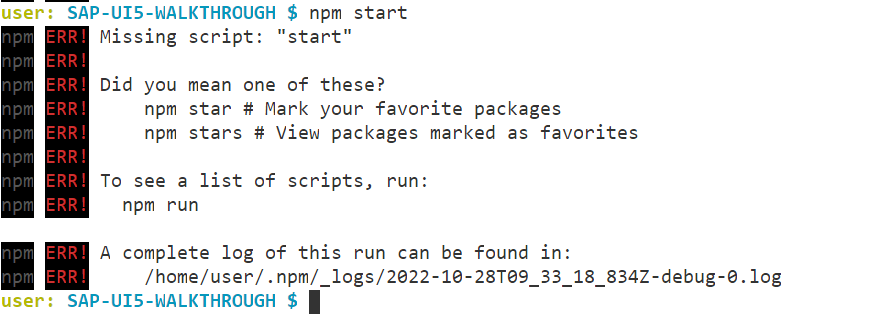


Figure Start script not found

**Exercise2:** what will happen if UI5 Tooling is not mentioned in package.json as shown below.



Figure UI5 Tooling info in package.json

**Answer:** It will result in the error ‘Unknown Middleware Fiori-tools-proxy’



Figure 'Unknown Middleware Fiori-tools-proxy' Error

**Exercise 3:** What will happen if the devDependencies are not filled completely in package.json as shown below.



Figure devDependencies not filled properly in package.json

**Answer:** It will result in error ‘npm dependencies missing’ as shown below.

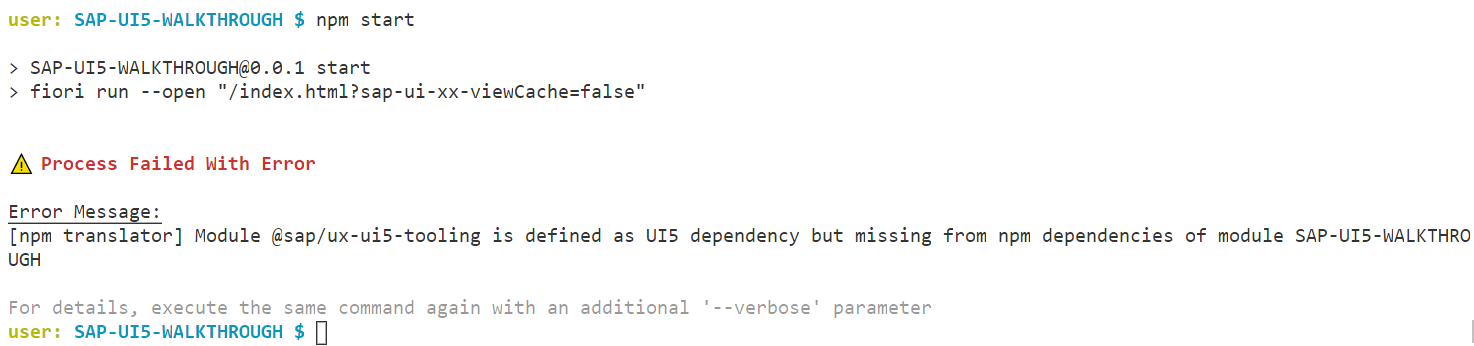


Figure npm dependencies missing error

# Define the models in Manifest.json

Manifest.json file is called as the descriptor file which is defined in JavaScript Object Notation(JSON) format. It defines the static information of the application which includes application name, file locations. As you can see in Figure 6 OData Service model, the “models” key contains three values which are again key value pairs such as i18n, recipe-service and user-management. The Odata service is referred with the parameters such as datasource, settings etc. The i18n model for language independency is also defined with parameters such as type,settings, bundlename. Finally a local JSON Model such as user-management is referred with type JSON model.



Figure Data source with type 'OData'



Figure OData Service model

# Define the root view in Manifest.json

The rootView key which is the initial or first view of the application when it is loaded. It includes parameters such as viewName, type, async and id.



Figure Root View

# Define the router in Manifest.json

The routes key in the manifest.json includes the different views of the application under a list and targets with the view id and its name. For example, the application contains views such as overview, detail, create recipe, treetable and compare views. The values which can be passed between different views are mentioned under the parameter ‘pattern’. It includes a name and target.

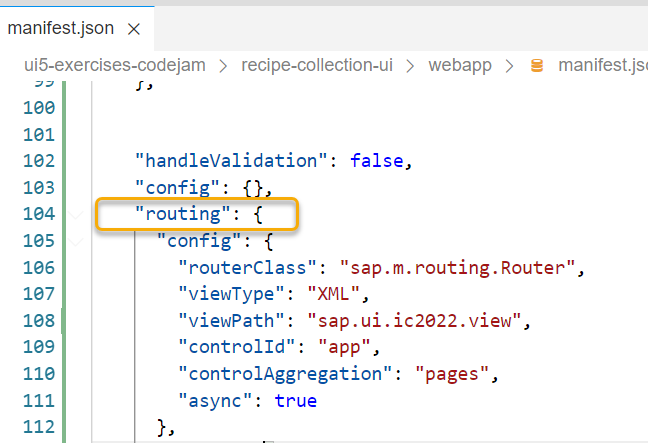


Figure Routing class definition



Figure List of routes with targets

The targets are further defined with the view id and view name.



Figure List of targets with the view id and the name

# Index.html

In the index.html file under the webapp folder, the standard ui5 libraries are defined. It also includes references to other files such as manifest. json,component.js etc. the index.html file name cannot be changed.



Figure Index.html

# Component.js

The component.js file includes details about the manifest.json and the router is initialized.

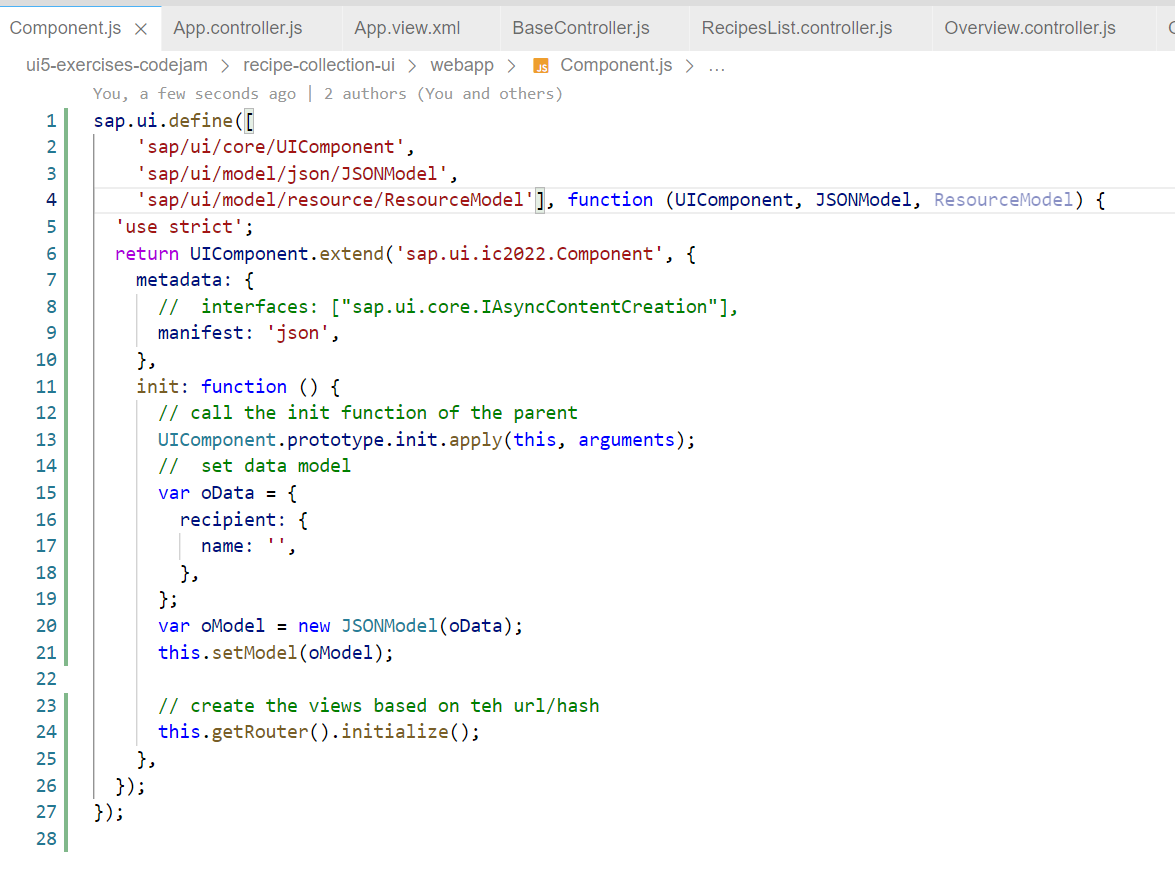


Figure Component.js

# Program flow

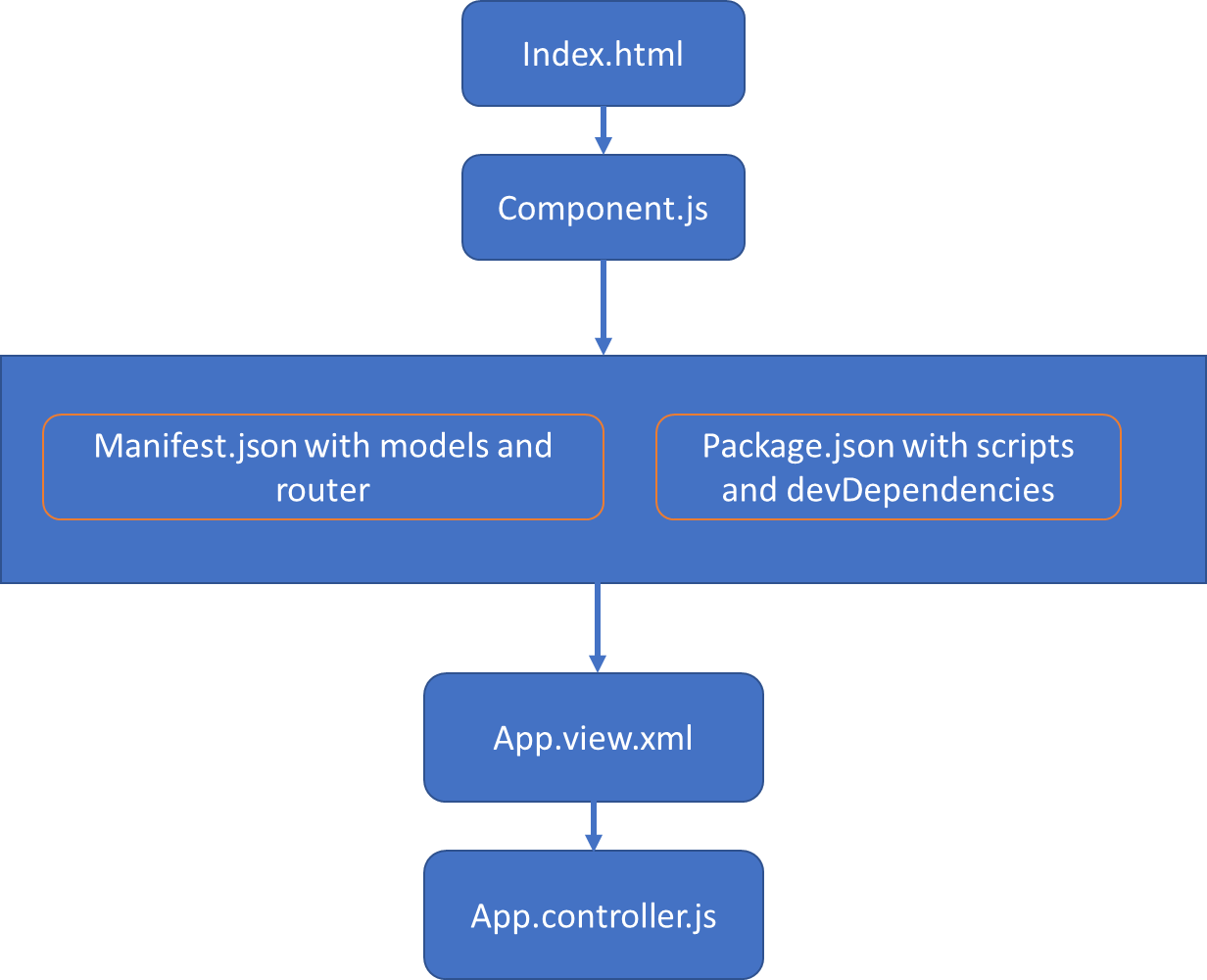


Figure Program flow in UI5 application

# App.view.xml

It is the first view of the UI5 application. It contains only the Shell control. A Shell contains an App or a SplitApp which is wrapped in a view. It is the parent container which helps to limit the app width for large screens as shown in Figure 19 Shell Control (Parent Container).

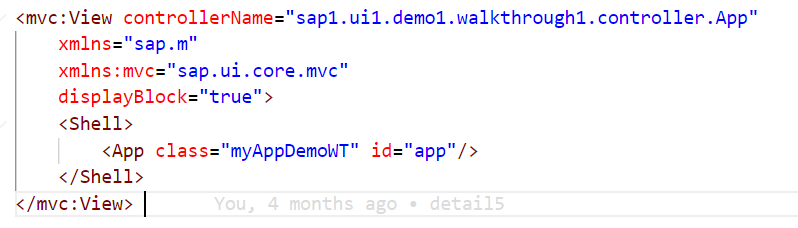


Figure Shell Control

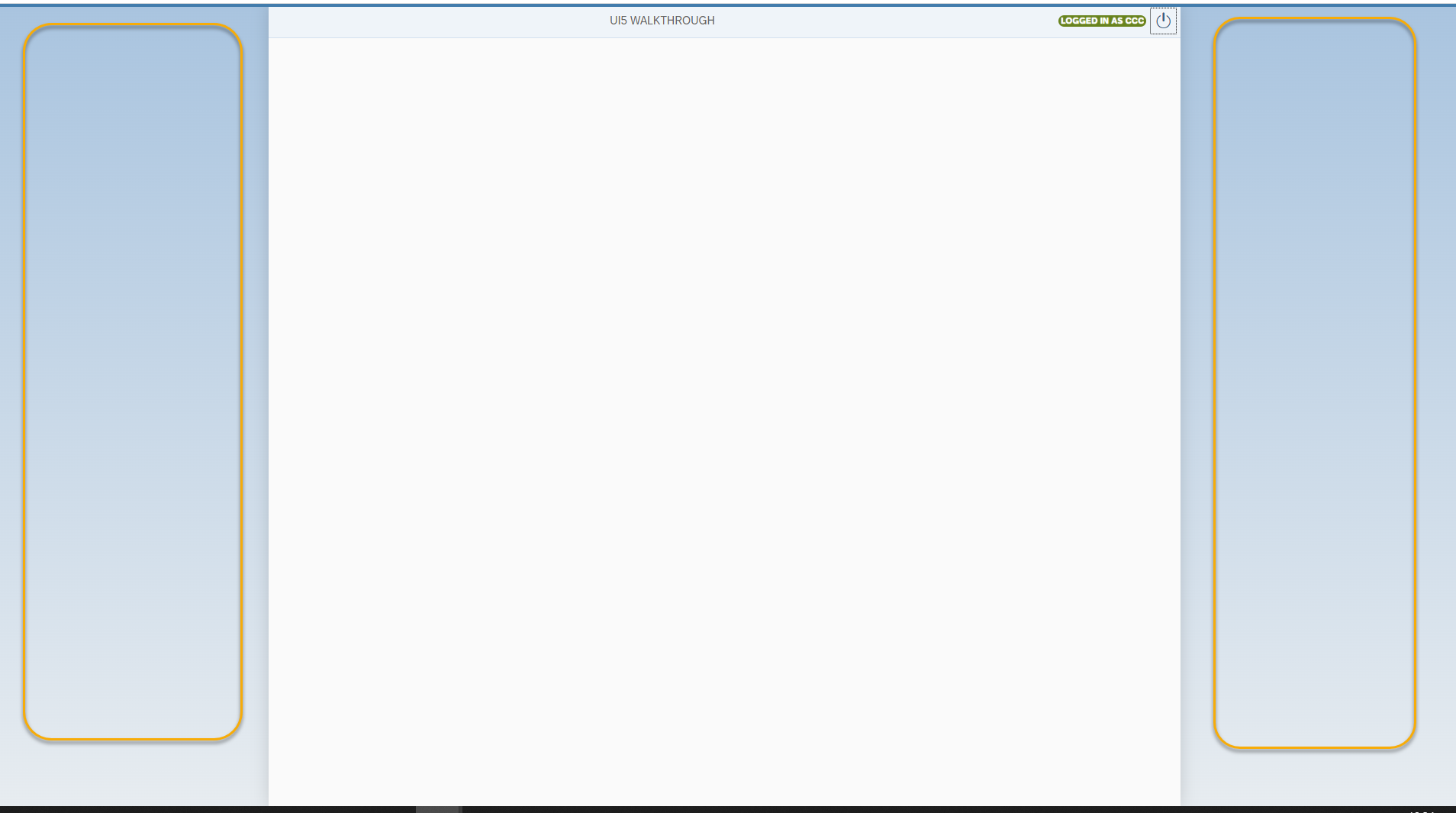


Figure Shell Control (Parent Container)

# App.controller.js

It is the initial Controller. In the onInit function of the controller, we set the ‘alias’ property of the JSON Model ‘user-management’ with the value from the localStorage(‘alias’). If the ‘alias’ property of the localStorage is empty, then the null value is set. The ‘alias’ property of the JSON Model is bound to the InfoLabel control(sap.tnt.InfoLabel) in Overview.view.xml which is shown in Figure 22 InfoLabel Control bound to alias of user-management.



Figure App.controller.js

# Login

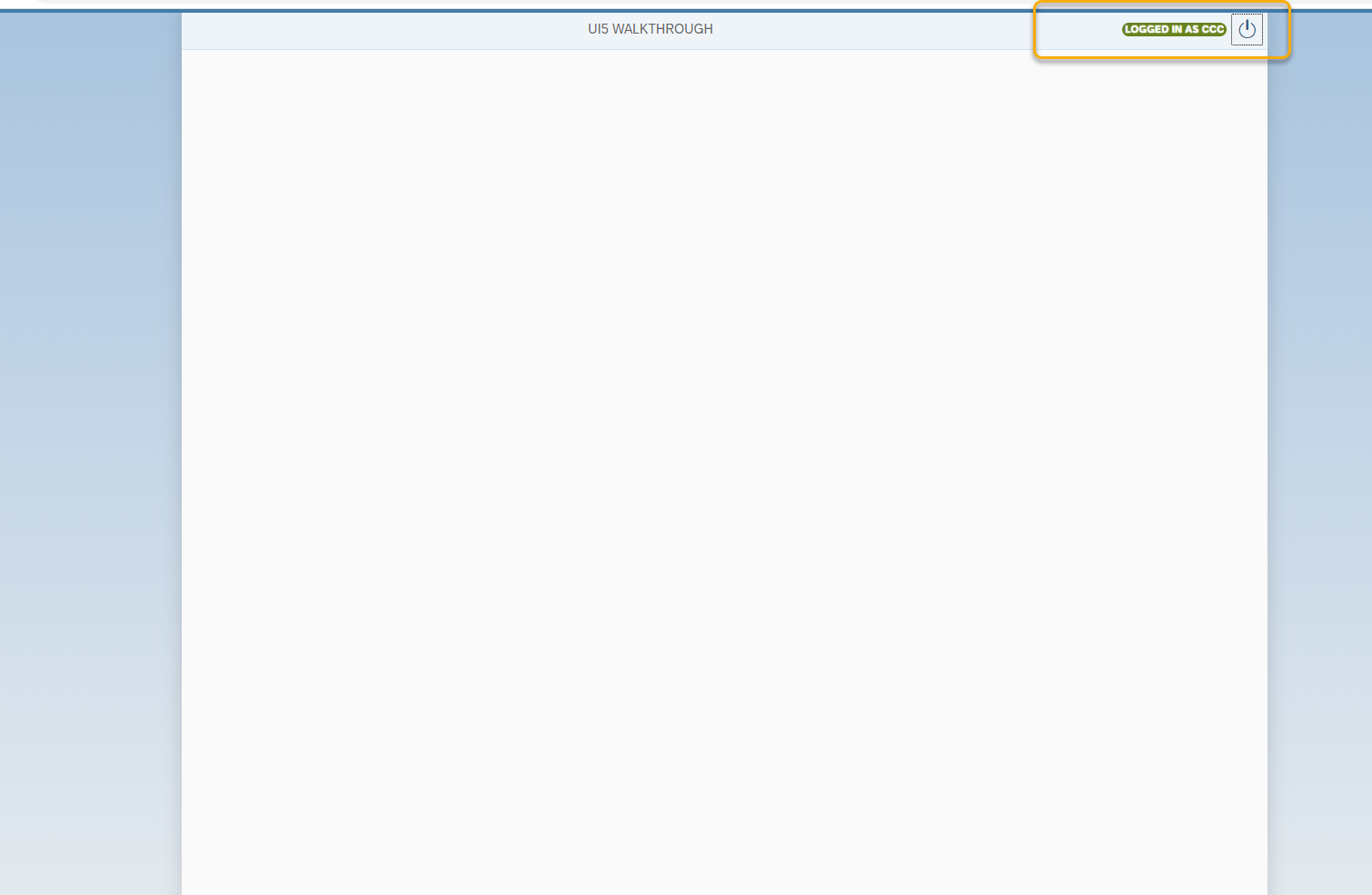


Figure Login functionality



Figure InfoLabel Control bound to alias of user-management

**Login Dialog box**

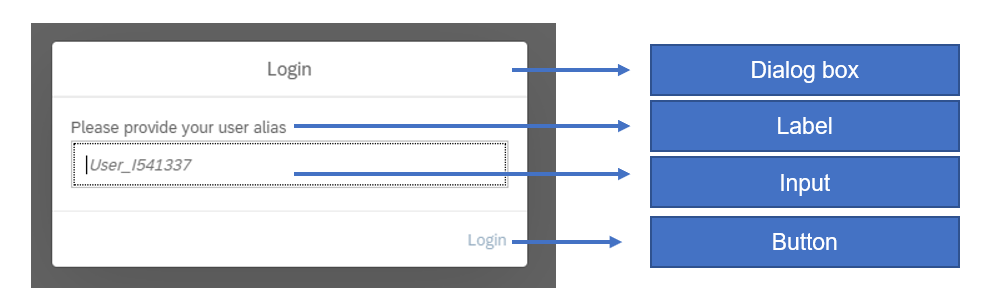


Figure Login Dialog Box and its controls

Login Dialog box is created within the Overview Controller file under aliasDialog1 function. So the Dialog box with the controls such as label, input and button instances as shown in ( *Figure 23 Login Dialog Box and its controls* ) are created inside the alias Dialog1 function. The properties are set in the constructor of the instances and the events are bound to the respective instances and function. Bind() method is used on the event liveChange, so that the press event is bound to the button object using the Bind() method.



Figure Login Dialog function

**Login Button:** It sets the value for the localStorage and ‘alias’ property for the JSON Model (user-management). After closing the dialog box, it set the Input control ‘userAlias’ value to null and the fireLiveChange event of the input con



Figure Button instance with press event

**Logout Button:** It sets the null value to JSON Model ‘user-management’ and removes the item ‘alias’ from local storage.



Figure Logout function ( Clears localStorage and JSON Model )

Instances used in Overview.controller.js are shown in Figure 26 Instances used in Overview Controller.js



Figure Instances used in Overview Controller.js

**OnAfterRendering:** This method is called after the View has been rendered. So the HTML is already a part of it. In this method, we call the alias Dialog function defined, which will open up the Login Dialog box if the user has not logged in. If the ‘alias’ property of the localStorage is null, then the login Dialog box opens as shown in Figure 27 Login Dialog box called from aliasDialog1 function.

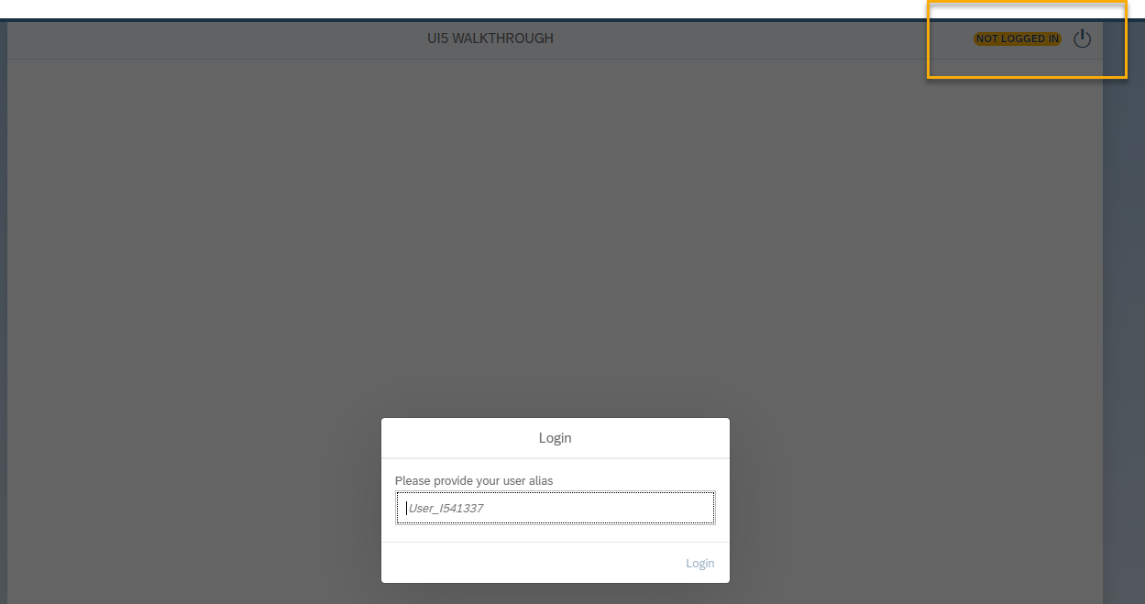


Figure Login Dialog box called from aliasDialog1 function

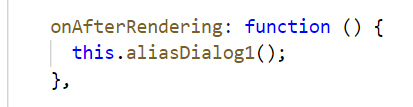


Figure Calls the aliasDialog function

**Exercise 1:** What will happen if the ‘alias’ property of the localStorage is null and the lines inside the function onAfterRendering is commented as below?

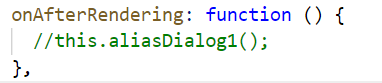


Figure Lines commented in onAfterRendering function

Answer: The login Dialog box will not popup as shown in Figure 30 Login Dialog box does not popup.

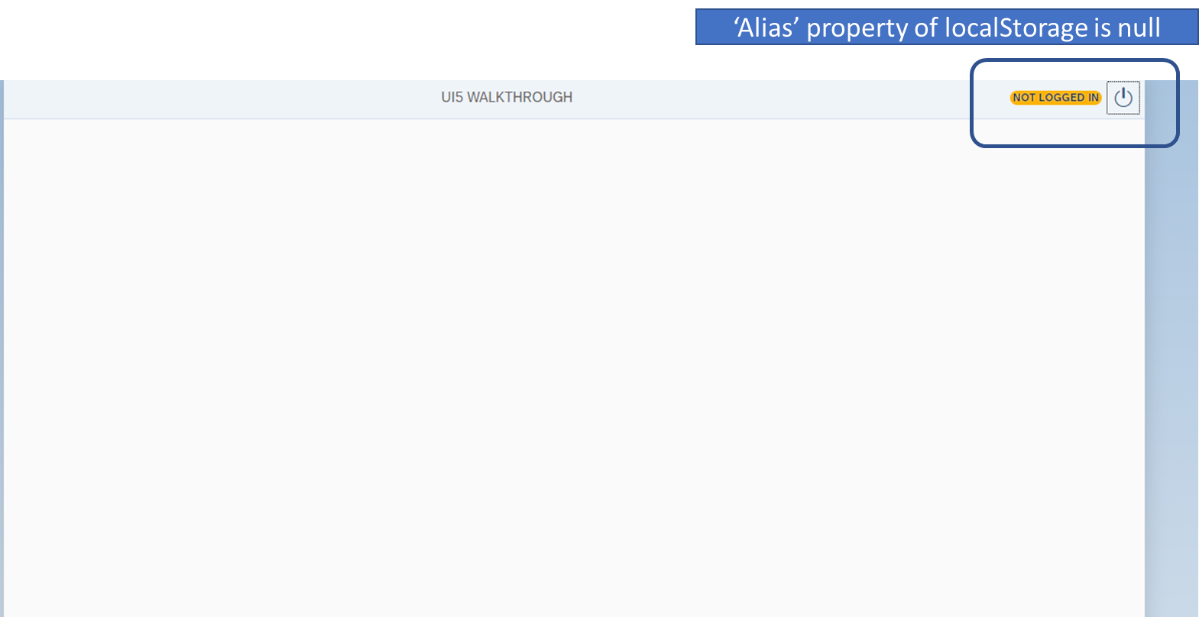


Figure Login Dialog box does not popup

**Exercise 2:** What will happen if the bind(this) is not used for the ‘liveChange’ event of the Input Control as shown in Figure 32 bind(this) not used in liveChange event of Input Control.



Figure bind(this) not used in liveChange event of Input Control

**Answer:** The liveChange event will not be bound to the Input Control instance. Therefore, any change in the input value will not trigger the event. Although the Input Value changed to ‘User\_changed’, the ‘Login’ Button is not enabled as shown in Figure 33 Login button not enabled as liveChange event not bound

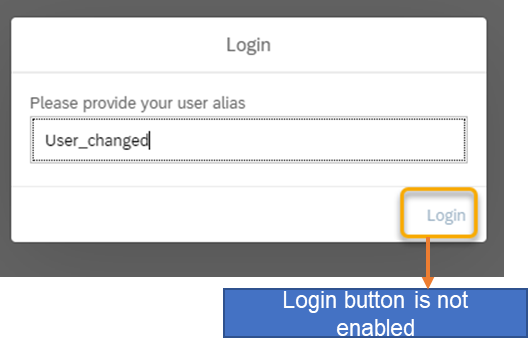


Figure Login button not enabled as liveChange event not bound

# Create a recipe

The Recipe is created in the CreateRecipe.view.xml page. The overview page will contain the button to navigate to the Create Recipe as shown below.

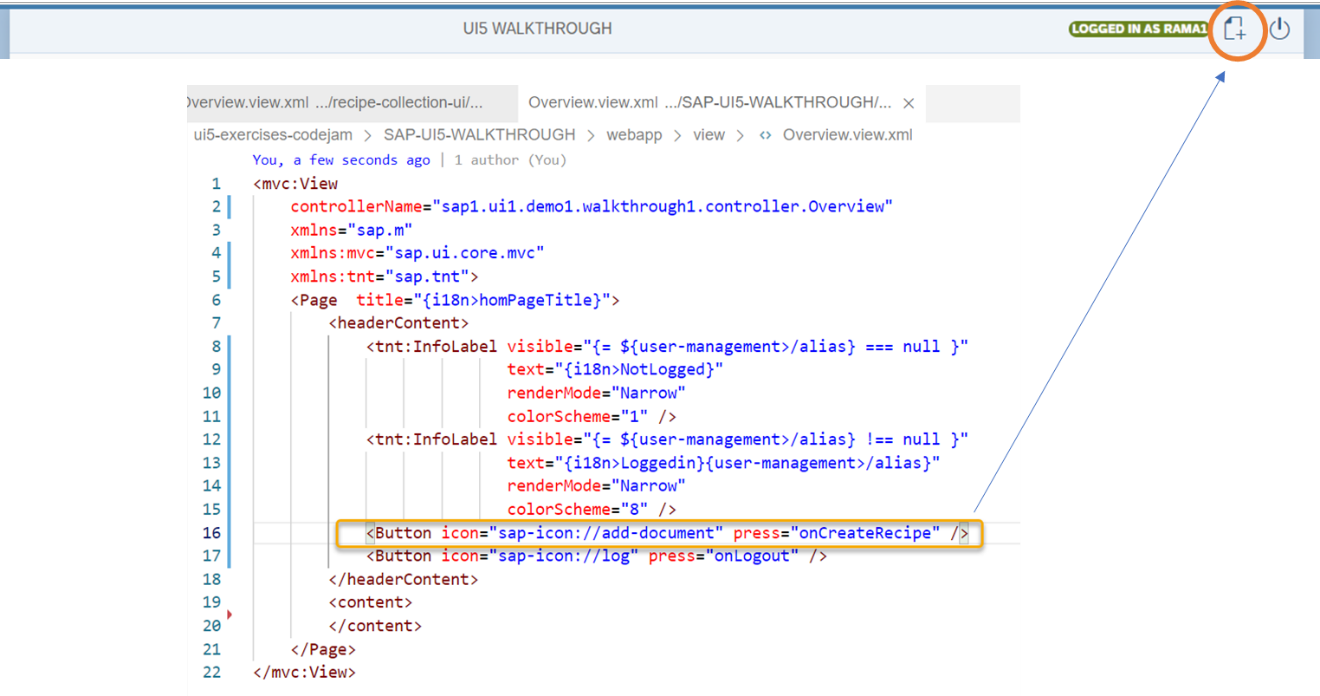


Figure CreateRecipe button inserted in Overview page

Let us create a CreateRecipe.view.xml web page with the Page Control. The page control contains <headerContent>,<content> and <footer>. The header content contains InfoLabel control. The body part of the createrecipe page is wrapped up in the <content> control. The <content> control contains three forms. First form contains the general details of the Recipe. Second form contains details about the ingredients. Third form contains details about the method of preparation. The footer contains the Create Recipe button.

**Headercontent:** Login information is written in the header content of the page control.

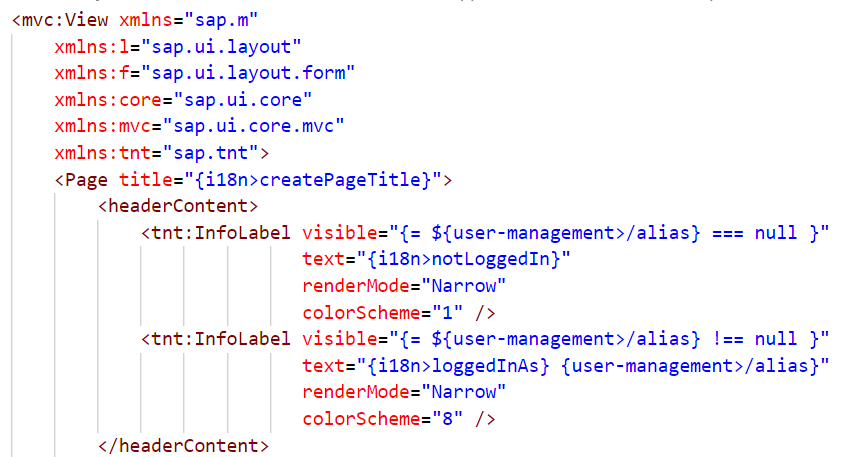


Figure Header content of Create Recipe View

**Content:** The below picture shows the first form with the field Author.



Figure content of Create Recipe View

The view is created as shown above. Now the view must also be mentioned in the manifest.json. so that the router instance can be used to navigate to the Create Recipe web page as shown in Figure 19 List of routes with targets and Figure 20 List of targets with the view id and the name.

The router instance is retrieved from the UIComponent class of the Basecontroller as shown in below figure.



Figure getRouter function in Base Controller

Then in overview controller, the navTo method of getrouter instance is called to navigate to the Create Recipe web page as shown in below figure.

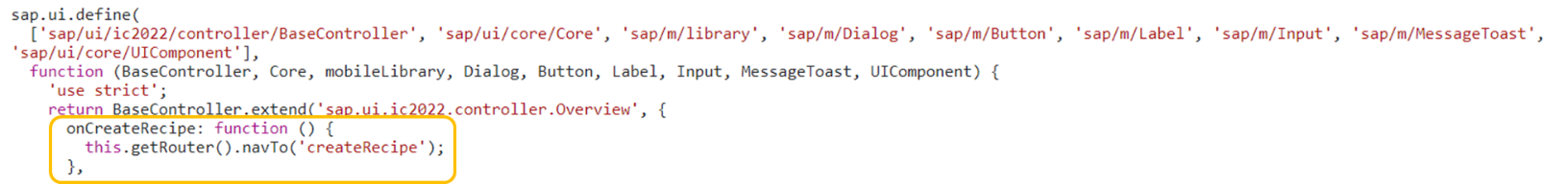


Figure Navigation to Create Recipe Web page

Now let us create all the fields for the Create Recipe web page. Please refer the GitHub page to see the complete code for the CreateRecipe.view.xml. The field values are linked to the local JSON Models of the Create Recipe web page.



Figure Field values linked to Local JSON Models

The Local JSON Models are defined in the onInit function of the createrecipe controller file. The general details of recipe such as recipe name, description, author, preparation time, recipe category, cuisine etc. are stored in newRecipeModel. Ingredients,Method of preparation are referred in ingredientModel and prepStepModel. The newRecipeModel contains the ingredientModel and prepStepModel as nested JSON data. ‘this.setModel’ refers to the setModel function of the Basecontroller.

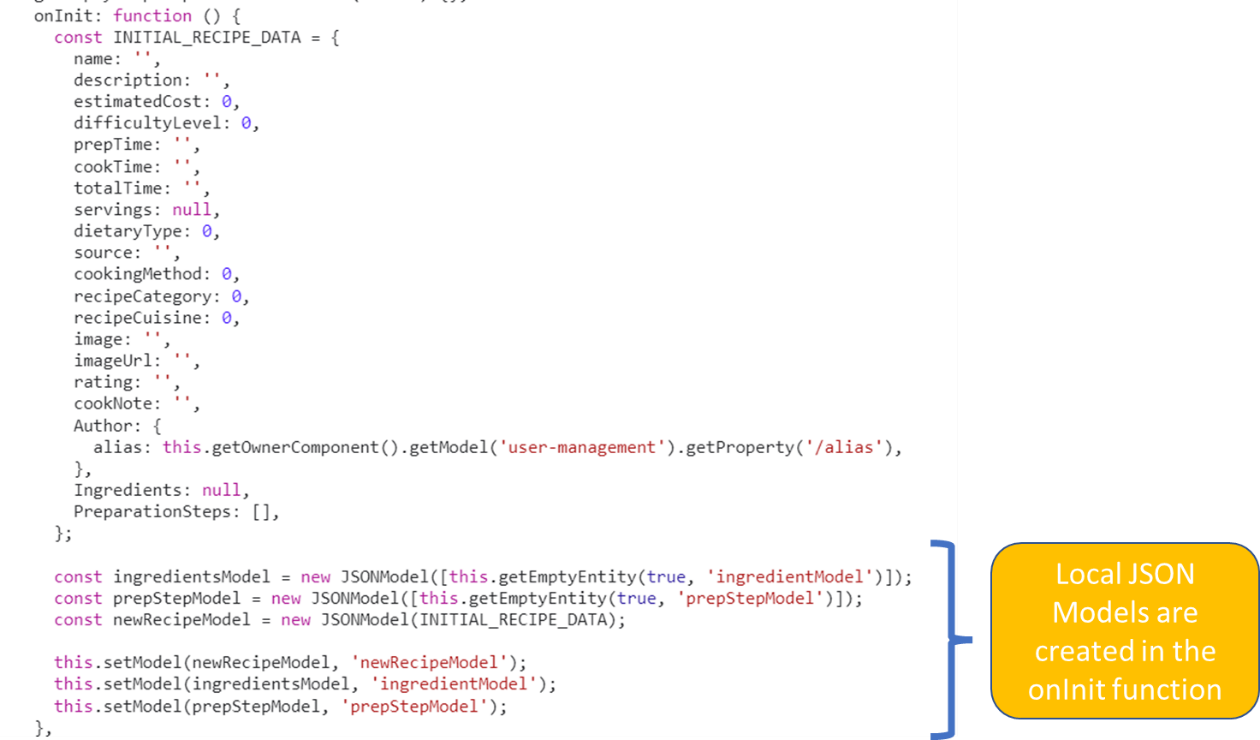


Figure Local JSON Models in onInit function

The Local JSON Models are filled with initially null values. This functionality is implemented in the getEmptyEntity function as shown below.



Figure getIndex and getEmptyEntity functions

**Adding Ingredients and Method of Preparation:**

**Deleting Ingredients and Method of Preparation**

# Display a list of items

**Model:** Data is available through the O-DATA service ‘recipe-service’

**View**: List control

Key learnings:

* How to connect the odata service directly to the list controller in view
* What are the key parameters maintained in the items property of list controller
* What are the aggregations to be maintained for the items
* Difference between object list item,first status, object status,second status, object attribute

Controller: Just a basic onInit function declaration

Key learnings: Do we need a oninit function for the display of items

# Display an Item

**Controller of Recipe List Page:**

**Controller of Detail page:**

**View of Detail page:**

# Edit an Item

***Key learnings:***

* **How to add an item to a list under an object**
  + **Define the model with initial values**
  + **Accessing the model in the view**

**How to delete an item from a list under an object**

# Save an Item

**Key Learnings: how to save the data in database available in all controls at one go**

# Delete an Item

# Copy an Item

# Display Hierarchy of items:

**Controller of Tree page:** get the parent child models using the parent id received from detail page. Set the parent child models to the Tree and Tree Table.

**View of Tree table:**

**Controller of Treetable:**

**Controller of Compare:**

**Going back to the Tree Controller:**

**View of the Compare page**



