**Dynamic From Setup**

This document gives information on how to implement the shared dynamic forms in your page.

As an example we are using ReasonType Maintenance page > Reason child grid Add popup.

A popup can be with or without tabs:

**I. Without tabs**

**II. With tabs**

**I. Without tabs (simple form)**

The setup will be done these files:

**1)Add Html (reason-add.component.html)**

**2)Add Typescript (reason-add.component.ts**

**3)Config Typescript (reason-config.service.ts)**

**4)Grid Typescript (reason-grid.component.ts)**

Each of these files are described below:

**1) Add Html (reason-add.component.html)**

fig: reason-add.component.html

In the html we have below code to call the dynamic-form shared component:

<dynamic-form [dialogRef]="dialogRef" [fields]="fieldConfig" [formConfig]="formConfig" (submit)="submit($event)">

</dynamic-form>

When calling the dynamic-form we need to pass required configuration to it:

\* [dialogRef]="dialogRef"

This contains the dialog related information.

\* [fields]="fieldConfig"

This is used to pass all field configuration.

\* [formConfig]="formConfig"

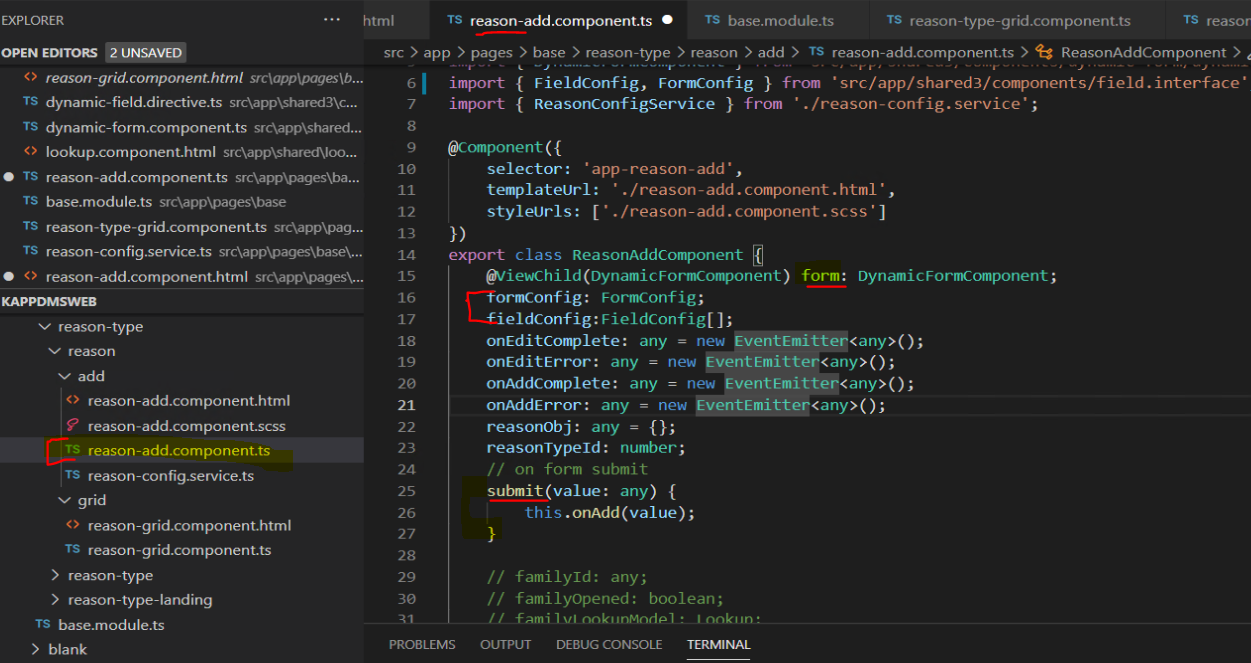
This is used to pass any form configurations.

It will also have an event which has to be handled:

\* (submit)="submit($event)"

These configurations and event handling are done on the Add Typescript side as shown in next section.

**2) The Add Typescript (reason-add.component.ts)**

fig: reason-add.component.ts

The configurations are declared here namely:

  @ViewChild(DynamicFormComponent) form: DynamicFormComponent;

    formConfig: FormConfig;

    fieldConfig:FieldConfig[];

And also the event is handled here:

 // on form submit

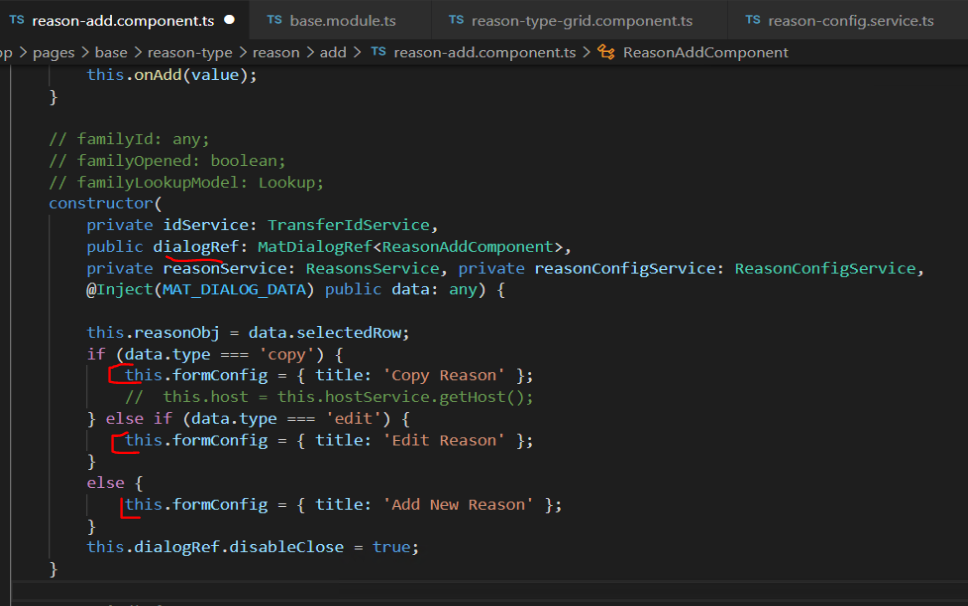
    submit(value: any) {

        this.onAdd(value);

    }

**a) formConfig**

The formConfig is used to pass any form related information. Currently we are using it to set the form title. This is done as below in the constructor.

fig: reason-add.component.ts constructor

**b)dialogRef**

You can also see the dialogRef that is also one of the things that is passed to the dynamic form control. This contains the dialog information.

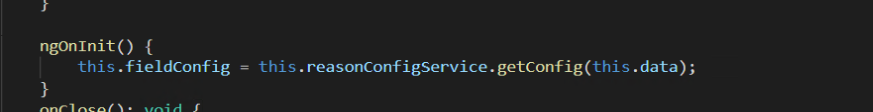
**c) The grid selected row object:**

  this.reasonObj = data.selectedRow;

The selectedRow will have grid’s selected row data. We will use this to set the id(primary key) while saving.

**d) fieldConfig**

The field configurations are written in a separate file ‘The Config Typescript file’ explained in next section. The configurations are fetched as below:

fig: reason-add.component.ts ngOnInit hook

this.data parameter has the grid selected row and the type of action (Add/Edit/Copy)

**e) submit event**

As mentioned in section 1) we have the event handling done as below:

This event happens on clicking the save icon in the form header and the ‘value’ parameter will have all the values as an object that will be passed to the backend. In this case we are calling the onAdd function with that ‘value’.

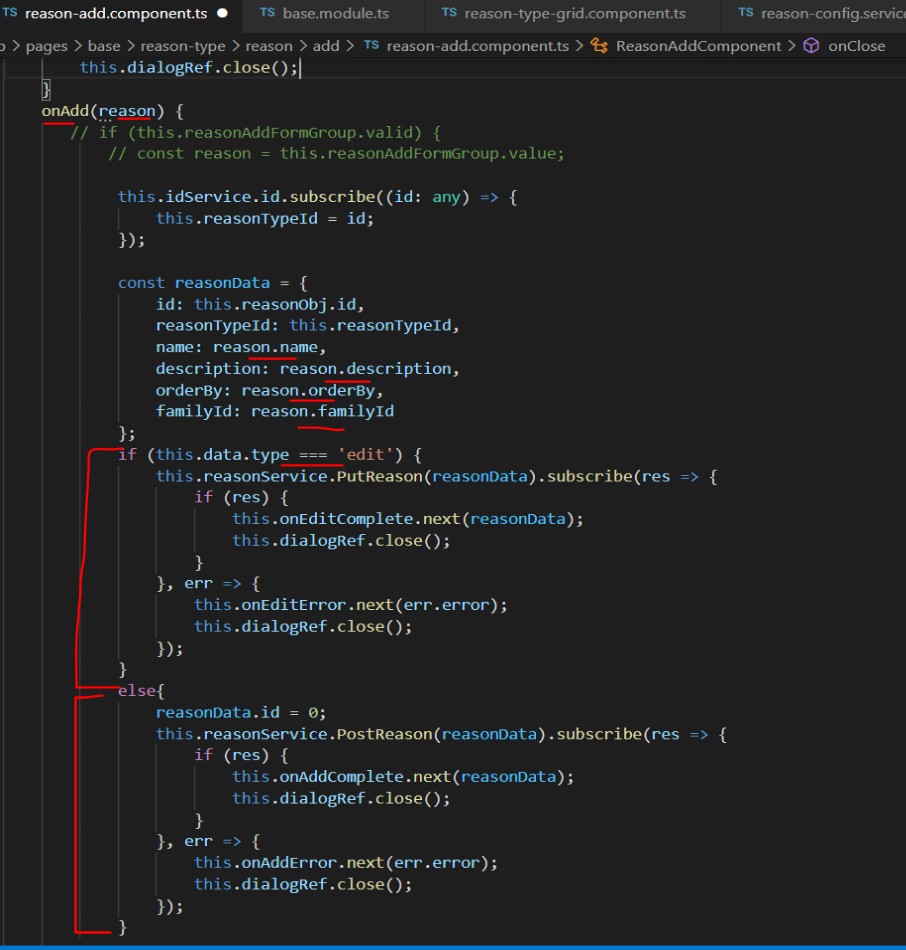
 // on form submit

    submit(value: any) {

        this.onAdd(value);

    }

**f)The onAdd function:**

fig: reason-add.component.ts onAdd function

The onAdd function receives all the needed input control values.

The id will be obtained in the constructor which was mentioned above in 2.c.

Depending on edit or add/copy appropriate service function is called which will call the PostReason(Add/Copy) or PutReason(Edit).

**3)The Config Typescript (reason-config.service.ts)**

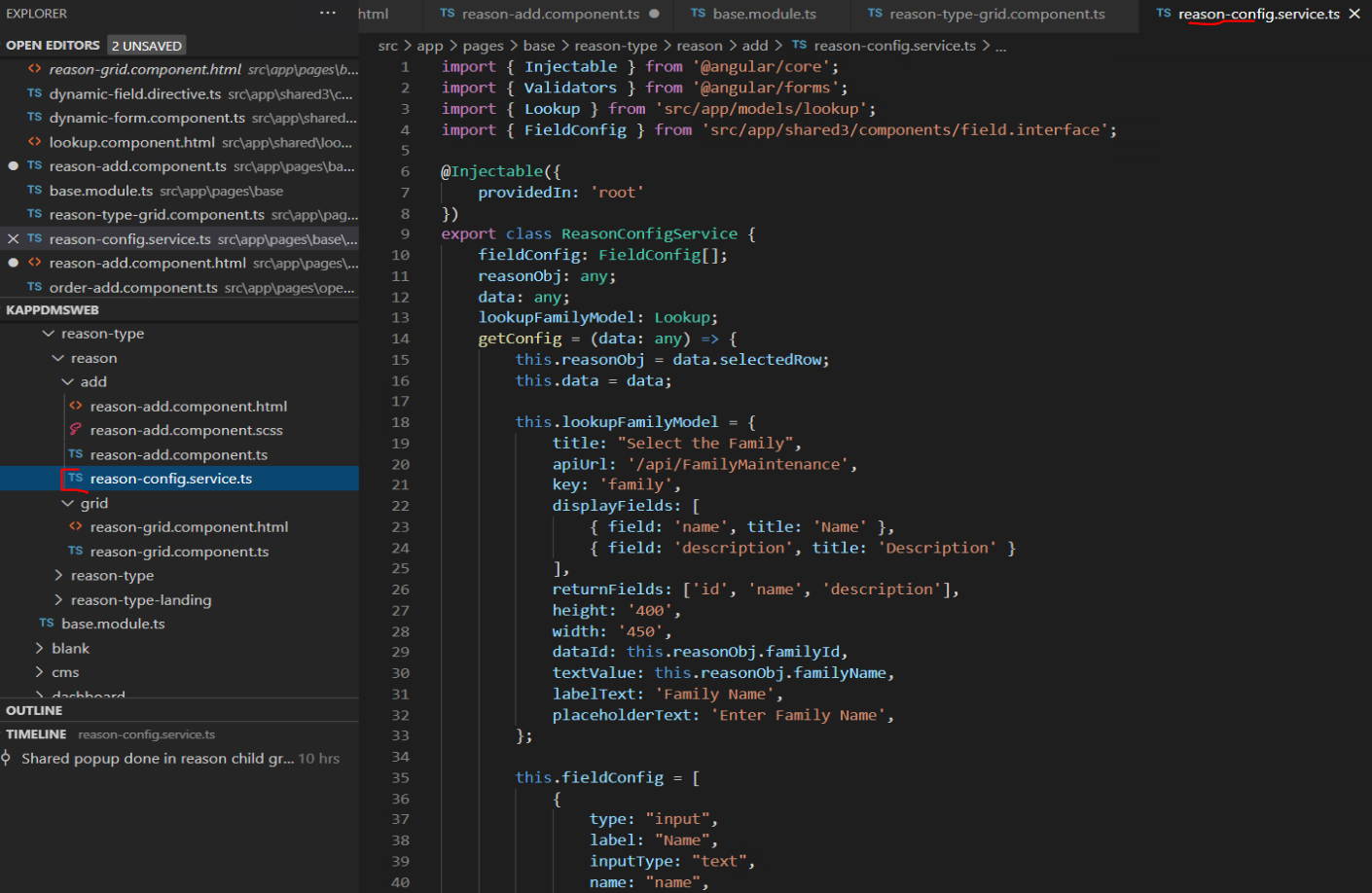
 fig:reason-config.service.ts



fig: reason-config.service.ts fieldConfig variable

The field configurations are written in separate file (reason-config.service.ts) for readability.

These field config are fetched into the Add Typescript in the onInit hook mentioned in 2.d.

FieldConfiguration explanation:

**a) For input type = text**

{

                type: "input",

                label: "Name",

                inputType: "text",

                name: "name",

                value: this.reasonObj.name,

                validations: [

                    {

                        name: "required",

                        validator: Validators.required,

                        message: "Build Code Required"

                    },

                    {

                        name: "maxlength",

                        validator: Validators.maxLength(100),

                        message: "Only 100 characters allowed"

                    }

                ]

            },

**b) For input type = number**

{

                type: "input",

                label: "Order BY",

                inputType: "number",

                name: "orderBy",

                value: this.reasonObj.orderBy,

                validations: [

                    {

                        name: "maxlength",

                        validator: Validators.maxLength(100),

                        message: "Only 100 characters allowed"

                    }

                ]

            }

**c) For date (from Order page)**

{

        type: "date",

        label: "End Effective Date",

        name: "endEffectiveDate",

        value: this.orderObj.endEffectiveDate,

      }

**d) For datetime(from Order page)**

{

        type: "datetime",

        label: "Planned Start Date Time",

        name: "plannedStartDateTime",

        value: this.orderObj.plannedDateTime,

      }

**e) For lookup**

{

                type: "lookup",

                label: "Family Name",

                name: "lkFamily",

                lookupModel: this.lookupFamilyModel,

            }

Notice we also need to pass a lookup model defined as:

 this.lookupFamilyModel = {

            title: "Select the Family",

            apiUrl: '/api/FamilyMaintenance',

            key: 'family',

            displayFields: [

                { field: 'name', title: 'Name' },

                { field: 'description', title: 'Description' }

            ],

            returnFields: ['id', 'name', 'description'],

            height: '400',

            width: '450',

            dataId: this.reasonObj.familyId,

            textValue: this.reasonObj.familyName,

            labelText: 'Family Name',

            placeholderText: 'Enter Family Name',

        };

**f) For image (from Order Page)**

{

      type: 'image',

      label: 'imageA',

      name: 'imageA',

      value: this.orderObj.imageA,

    }

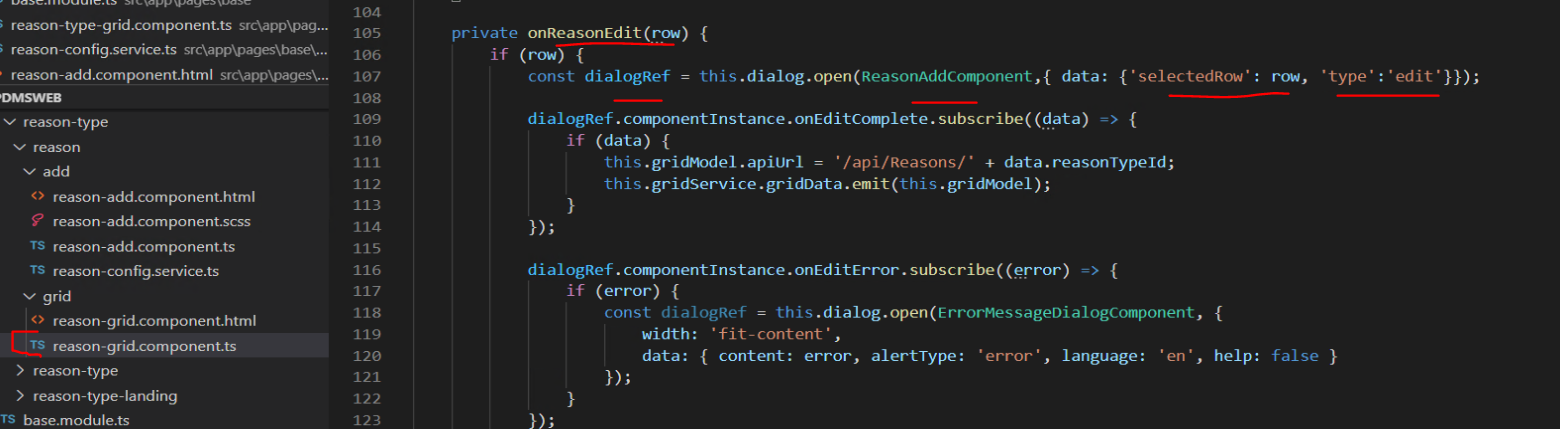
**4) Grid Typescript (reason-grid.component.ts)**

This file has minimum changes:

We already have the event handling for this namely (Add/Edit/Copy) obtained from the shared grid:

Need to make single line modifications:

**a) For edit:**

fig: reason-grid.component.ts edit

**b) Similarly for Add:**

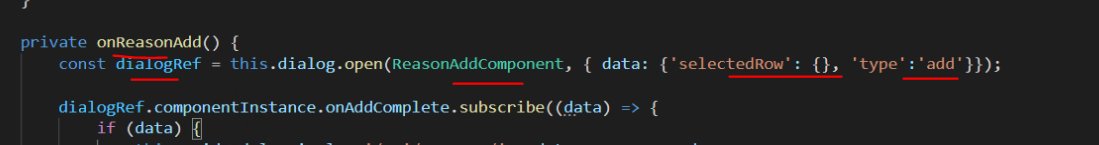


fig: reason-grid.component.ts add

**c) For Copy:**

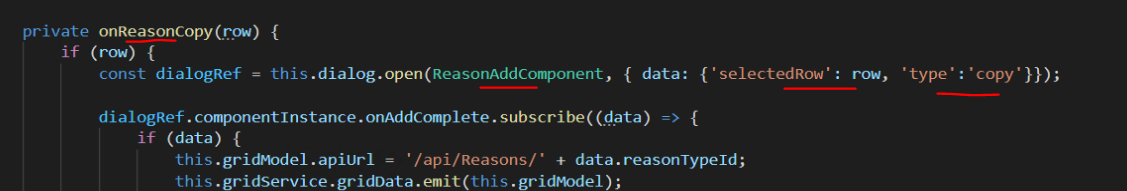


fig: reason-grid.component.ts copy

**II. With tabs**

The change is minor. In case of tabs instead of passing the fieldConfig to the dynamic-form component, we pass the tabConfig. The tabConfig is an array of fieldConfigs.

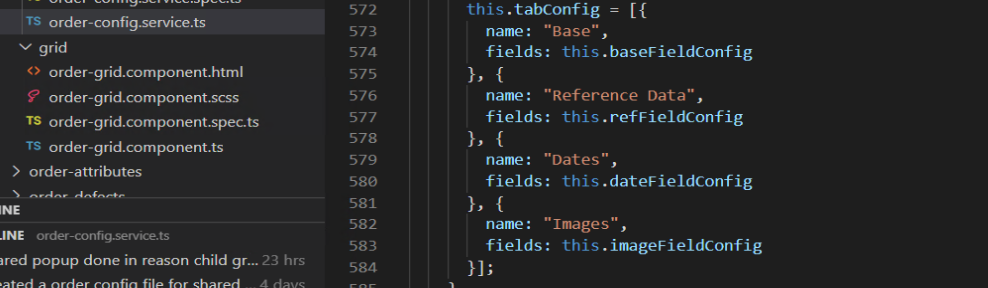
In the Add Html we will have below code.

<dynamic-form [dialogRef]="dialogRef" [tabs]="tabConfig" [formConfig]="formConfig" (submit)="submit($event)">

</dynamic-form>

**a) tabConfig**

The tabConfig is also set in the Config Typescript(order-config.service.ts ) file as shown below:

fig: order-config.service.ts tabConfig variable

The fieldConfigs are added to the tabConfig array and passed to the dynamic-form component.