**Layout column with detail view, tab and group function**

1. As Bang and Hoang suggestion, I fixed the column layout by the second way. Now the **maxColumn is removed**, the attributes is layout from top to bottom, if you need to add 2 columns in a row, call the function **newColumn()**. And the size of component is fixed, that means the size of component not changed when the window is resized. I will add function setWidth() later comeover this problem.
2. The group function is added. Please use **startGroup()** and **endGroup()**
3. The tab is also implemented. Please use **tab()** function.
4. Tab and group function is just support for 1 level .

See the example on EditSupplierView:

    @Override

**public** **void** initialPresentationView(DetailDataModel detailDataModel, Supplier entity) {

        detailDataModel

                .**tab**("We are S3S", "This is a tooltip", **new** ImageIcon(EditSupplierView.**class**.getResource("/icons/user.png")));

        detailDataModel.addAttribute("code", FieldTypeEnum.*TEXTBOX*);

        detailDataModel.addAttribute("name", FieldTypeEnum.*TEXTBOX*);

        detailDataModel.**startGroup**("Group 1");

        detailDataModel.addAttribute("representer", FieldTypeEnum.*TEXTBOX*);

        detailDataModel.addAttribute("sex", FieldTypeEnum.*RADIO\_BUTTON\_GROUP*).referenceDataId(*SEX\_ID*);

        detailDataModel.addAttribute("position", FieldTypeEnum.*TEXTBOX*);

        detailDataModel.**endGroup**();

        detailDataModel.addAttribute("address", FieldTypeEnum.*TEXTAREA*);

        detailDataModel.**tab**("We make it work!", "Tab 2", **null**);

        detailDataModel.addAttribute("phone", FieldTypeEnum.*TEXTBOX*);

        detailDataModel.**startGroup**("Group 2");

        detailDataModel.addAttribute("fax", FieldTypeEnum.*TEXTBOX*).**newColumn**();

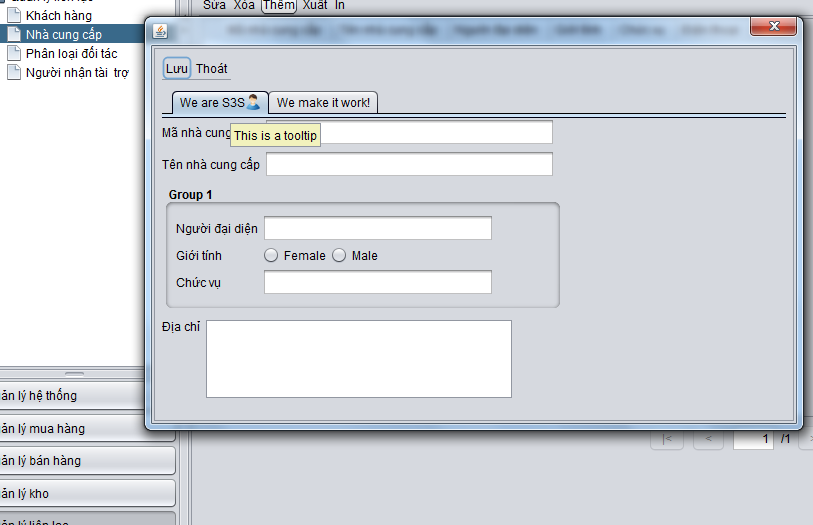
        detailDataModel.addAttribute("email", FieldTypeEnum.*TEXTBOX*);

        detailDataModel.**endGroup**();

        detailDataModel.addAttribute("isActive", FieldTypeEnum.*CHECKBOX*);

        detailDataModel.addAttribute("comment", FieldTypeEnum.*TEXTAREA*);

    }



**Multiview for edit view**

To flexible for the edit view. I changed a little bit  the design of AbstractDetailView:

1. AbstractDetailView is renamed to AbstractSingleEditView
2. From the List View. Open the Edit View by AbstractEditView
3. The AbstractEditView has 2 kinds:
   1. AbstractSingleEditView: the edit screen with 1 view
   2. AbstractMultiEditView: the edit screen with multi views. Changed by the tree on the left
4. Implement the AbstractMultiEditView like the way implement AbstractDomainView

Just implement the constructTreeView() like this:

    @Override

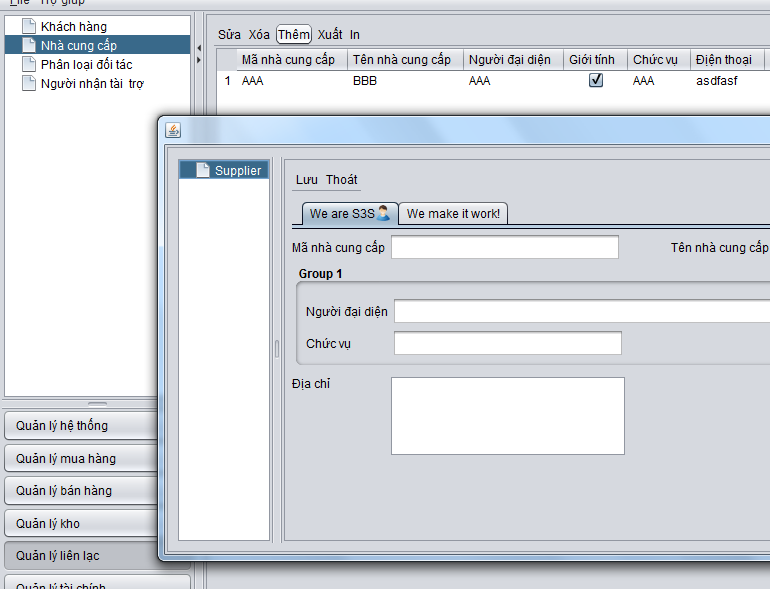
**protected** **void** constructTreeView(TreeNodeWithView root, Supplier entity) {

        TreeNodeWithView node = **new** TreeNodeWithView("Supplier", **new** EditSupplierView(entity));

        root.add(node);

}

See the EditMultiSupplierViewTest for example.



P/s: It is not good if the toolbar Luu Thoat inside the right view. I will move the toolbar to the top as the context toolbar later.

**Customize the component on the screen.**

I’ve just added function **protected** **void** customizeComponents() to customize the components on the screen. You can get the component from the map parameter.

Please see the example in EditSupplierView:

    @Override

**protected** **void** customizeComponents(Map<String, AttributeComponent> name2AttributeComponent) {

**super**.customizeComponents(name2AttributeComponent);

**final** JTextField email = (JTextField) name2AttributeComponent.get("email").getComponent();

        JCheckBox cb = (JCheckBox) name2AttributeComponent.get("isActive").getComponent();

        cb.addItemListener(**new** ItemListener() {

            @Override

**public** **void** itemStateChanged(ItemEvent e) {

                email.setVisible(e.getStateChange() == ItemEvent.*DESELECTED*);

            }

        });

    }

**Validation**

The validation on FW is almost done.

1. Validation on field:

* The FW automatically validate on the field marked by constraint validation include: @NotNull, @Max, @Min, … with the default message.

1. Manual validation:

* You need you write the method to validate the entity. The method must be **public**, return **boolean** type, and mark with annotation **@Validation.** The method return true if validation success, false if validation fail.
* Add the error message in i18n as the convention: “**error.{entity class name}.{method name}**”

Example in Suplier entity:

    @Validation

**public** **boolean** validateTest1() {

**return** **false**;

    }

    @Validation

**public** **boolean** validateTest2() {

**return** **false**;

    }

    @Validation

**public** **boolean** validateTest3() {

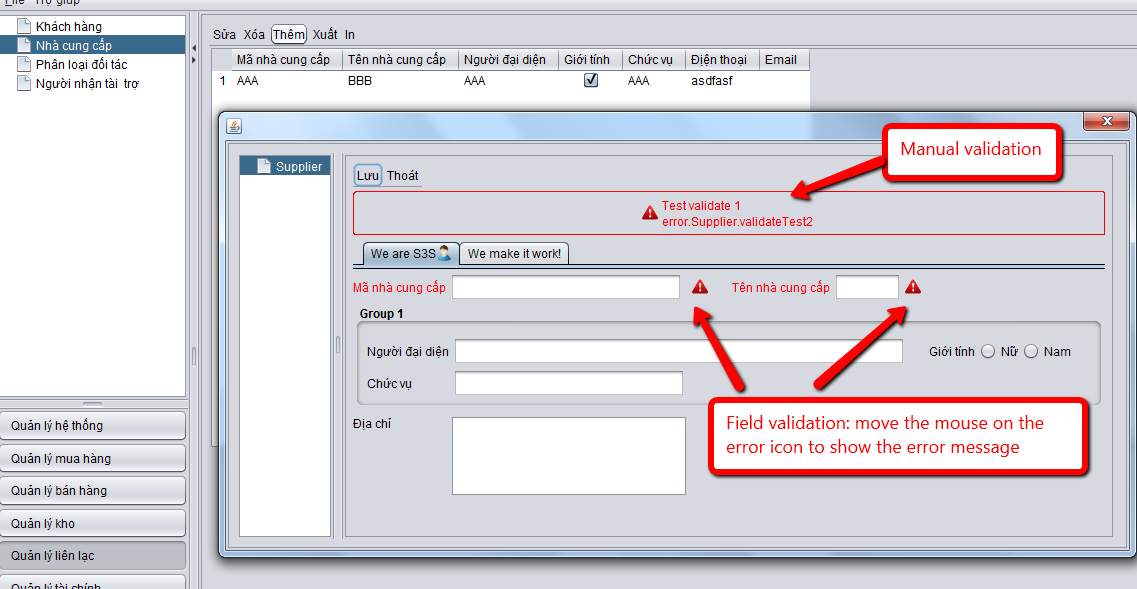
**return** **true**;

    }

In the i18n file:

error.Supplier.validateTest1=Test validate 1

On the screen:



P/S

@Bằng: Could you please add 2 kind of i18n .properties files on each module: message and error?

@Hoàng: Currently I used Supplier to test. Sorry for any inconvenience :D

Do not hesitate to show your idea!

Regards

Phúc