Template

Template file

The template file is the backbone of the whole widget. It is built on the simple JavaScript function returning vue Instance object. Each Vue instance has a unique id. It serves this purpose for easy saving of the user settings, user's data, and preferences. In the runtime "CustomModuleName_DynamicID" is being replaced with proper widget's id for further actions.

```
(function (internalName) {
    var internalID = internalName;
    return {
        moduleName: internalID,
        component: Vue.component(internalID, {
            props: [],
            data:,
            mounted() {},
            activated() {},
            deactivated() {},
            methods: {},
            computed: {},
            template: ``
        }),
    }

}("CustomModuleName_DynamicID"));
```

Working with database

Every Customer Panel launch system checks if the configuration for a specific customer already exists in the database. If it doesn't exist, it is being created, and then return to the frontend, or in the case, it already exists, simply return.

The system creates the configuration with the following pattern:

- <module_name>_<module_id>_<user_login>_hkrOUgHKUi_<invoking_module_name>_Settings

- <module_name>_<module_id>_<user_login>_hkrOUgHKUi_<invoking_module_name>_Data

Saving with Settings prefix is used for storing widget's configuration taken from the global widget's settings.

Saving with Data prefix is used for storing user's saved data. [settings configuration/user's actions/notes]

Set

Setting a user's data can be resolved with the following endpoint:

`POST <OTRS_URL>/customer.pl?Action=CustomerFrontend;Subaction=SetWidgetData;ID=<moduleName>`

```
sendNote: _.debounce(function () {
        axios.post(this.$store.getters["globalConfig/OTRS_URL"] +
"customer.pl?Action=CustomerFrontend;Subaction=SetWidgetData;ID=" + this.moduleName, {
            note: this.note
        }).then((response) => {
            console.log("Send note!")
        }).catch((error) => {
            console.log("ERROR [setWidgetConfig] ", error)
        });
    }, 300),
```

Get

Getting a user's data can be resolved with the following endpoints:

`GET <OTRS_URL>/customer.pl?Action=CustomerFrontend;Subaction=GetWidgetData;ID=<moduleName>`

```
getNote() {
        axios.get(this.$store.getters["globalConfig/OTRS_URL"] +
"customer.pl?Action=CustomerFrontend;Subaction=GetWidgetData;ID=" +
this.moduleName).then((response) => {
            if (response.data.success) {
                this.note = response.data.data.note
            }})}}
```

> These operations easily describe the whole idea of the template mechanism.

Global Storage

Global storage contains the most important information about OTRS and current CustomerPanel instance.

You can refer to its values by the keyword `this.$store` and retrieve properties with `getters`. Example of useful values:

```
[
	"account/getSessionId": "lkDT71THevxijoiKprX9LoHmehkoWu4emCa",
	"globalConfig/BACKEND_VERSION": "1.2.54",
	"globalConfig/FRONTEND_VERSION": "1.1.40",
	"globalConfig/OTRS_DEFAULT_LANG": "en",
	"globalConfig/OTRS_FQDN": "http://customotrsinstance.com",
	"globalConfig/OTRS_ScriptAlias": "/otrs/",
	"globalConfig/OTRS_URL": "http://customotrsinstance.com/otrs/",
]
```

These are very useful while working on your widgets. They allow you to connect to OTRS API our display information about currently previewed tickets.

```
getContractInfo() {
  axios.get(this.$store.getters["globalConfig/OTRS_URL"] +
`customer.pl?Action=ContractWidget;Subaction=GetContractInfo;Tn=${this.$route.params.ticketNum
ber}`).then((response) => {
      if (response.data.success === false) {
          this.contract_info = null
          this.contractLoading = false
      } else {
          this.contract_info = response.data.information
          this.contractLoading = false
     }
  })
}
```

Accessing for nested objects in global storage can be easily realized by referencing other elements eg. `this.$store.getters["ticketPreview/ticketArticles"][0]['Age']`

> You can display whole global storage content by simply printing this.$store.getters in the console.

Language support

To use predefined language strings, depending on the user's default language settings you can simply reference them in the template file like this:

```
 widgetTitle() {

     return this.$t('Widget name')

}
```

or directly in the HTML template:

```
<span> {{$t('Contract loading')}}</span>
```

Revision #8
Created 13 July 2020 14:19:51 by editor
Updated 11 August 2020 13:37:21 by editor