

Writing a first customer-panel widget

Main assumptions

Our goal is to create a simple widget that can be added to the Dashboard view. It will include the number of new tickets assigned to the specific customer user.

Module Structure

The widget will be built as a standard OTRS package, easy to build and deploy. It will contain all necessary configuration needed for proper running. We will create a separate hidden module in which we will put our widget. Our whole package's structure will look like this:

```
otrs-ticket-information/  
├─ install.sopm  
├─ Kernel  
│   ├─ Config  
│   │   └─ Files  
│   │       └─ XML  
│   │           └─ TicketInformationWidget.xml  
│   └─ Language  
│       └─ pl_TicketInformationWidget.pm  
└─ Modules  
    └─ TicketInformation.pm  
└─ var  
    ├─ httpd  
    │   └─ htdocs  
    │       └─ customer-panel  
    │           └─ modules  
    │               └─ WIDGET_NewTickets.js  
    └─ intalio-customer-panel  
        └─ ticket_widget_translation_strings.json
```

XML Configuration

What we do here is registering the OTRS module - TicketInformation, which will be referenced when calling our API for retrieving the ticket's information. Furthermore, we register our CustomerPanel module and assign 'Open tickets counter' to him.

```
<otrs_config version="2.0" init="Application">
  <Setting Name="CustomerFrontend::Module###ContractWidget" Required="1" Valid="1">
    <Description Translatable="1">FrontendModuleRegistration for TicketInformation
module. </Description>
    <Navigation>Frontend::Customer::ModuleRegistration</Navigation>
    <Value>
      <Item ValueType="FrontendRegistration">
        <Hash>
          <Item Key="Group">
            <Array>
              <Item>users</Item>
            </Array>
          </Item>
          <Item Key="Description"
Translatable="1">TicketInformation. </Item>
          <Item Key="Title"
Translatable="1">TicketInformation</Item>
          <Item Key="NavBarName">TicketInformation</Item>
        </Hash>
      </Item>
    </Value>
  </Setting>
  <Setting Name="OTRSFrontendModule::TicketInformation" Required="0" Valid="1">
    <Description Translatable="1">This is module for a small ticket's informations
widgets</Description>
    <Navigation>CustomerFrontend</Navigation>
    <Value>
      <Hash>
        <Item Key="id">TicketInformation</Item>
        <Item Key="type">hidden</Item>
        <Item Key="text">Contract</Item>
        <Item Key="to">/module/TicketInformation</Item>
        <Item Key="priority">110</Item>
        <Item ValueType="Textarea" Key="widgets">
          [
            {
```

```

        "props": {},
        "id": "20",
        "widgetFile": "WIDGET_NewTickets.js",
        "compiled": false,
        "view" : "Dashboard",
        "name": "Open tickets counter"
    }
]
</Item>
</Hash>
</Value>
</Setting>
</otrs_config>

```

OTRS Module

In the previous subthread, we defined OTRS module ['TicketInformation'] which will be our bridge to OTRS API. We will define there whole logic connecting to the database and returning it to the web caller. In this case, we will retrieve number of new tickets.

```

package Kernel::Modules::TicketInformation;

use strict;
use warnings;
use utf8;
use JSON::PP;
use Data::Dumper;
use Kernel::System::VariableCheck qw(:all);

our $ObjectManagerDisabled = 1;

sub new {
    my ( $Type, %Param ) = @_ ;

    # allocate new hash for object
    my $Self = { %Param };
    bless( $Self, $Type );
    $Self->{ParamObject} = $Kernel::OM->Get( 'Kernel::System::Web::Request' );

    return $Self;
}

```

```

}

sub Run {
    my ( $Self, %Param ) = @_;

    if($Self->{Subaction} eq 'GetNewTicketsCount') {

        my $UserLogin = $Self->{UserLogin};

        my $ResultJSON = {
            success => 0,
            message => "An unknown error occurred. Please contact the
administrator.",
        };

        my $DBObject = $Kernel::OM->Get('Kernel::System::DB');

        $DBObject->Prepare(
            SQL => "SELECT
                    COUNT(t.id)
                FROM
                    ticket t
                JOIN
                    ticket_state ts ON ts.id =
t.ticket_state_id
                JOIN
                    ticket_state_type tst ON tst.id =
ts.type_id
                WHERE
                    customer_user_id = ?
                AND
                    tst.id = 1",
            Bind => [ \ $Param{UserLogin}],
        );

        while( my @Row = $DBObject->FetchrowArray() ) {
            $ResultJSON->{data} = $Row[0];
        }
    }
}

```

```

        if (exists $ResultJSON->{data}) {
            $ResultJSON->{success} = \1;
            $ResultJSON->{message} = "OK";
        }

        return $ResultJSON;
    }
}
1;

```

You can read more on creating frontend modules in [the official OTRS developer guide](#). Basically, based on the passed UserLogin we retrieve tickets with the 'new' state and return a simple JSON response.

Template File

After adding backend logic we are ready to create our template file. This contains a simple syntax built on vue instance.

```

(function (internalName) {
    var internalID = internalName;
    return {
        name: internalID,
        component: Vue.component(internalID, {
            props: [],
            data: function () {
                return {
                    moduleName: internalID,
                    count: "0",
                    lastUpdate: "",
                }
            },
            mounted() {
                console.log("WIDGET_NumberOfNewTickets loaded!")
                axios.get(this.$store.getters["globalConfig/OTRS_URL"] +
`customer.pl?Action=TicketInformation;Subaction=GetNewTicketsCount`).then((response) =>
{
                    this.count = response.data.data.toString()
                })
                this.lastUpdate = new Date().toLocaleTimeString().slice(0,5)
            }
        })
    }
})(internalName)

```

```

    },
    activated() {
    },
    deactivated() {
    },
    template: `
      <v-flex xs12 md3>
        <material-stats-card
          style="margin-top: 0px !important;"
          color="success"
          icon="new_releases"
          :title="$t('Number of new tickets') "
          :value="count"
          sub-icon="calendar_today"
          sub-icon-color="secondary"
          :sub-text="$t('Last update' )+'
+lastUpdate"

          sub-text-color="text-primary"

        />
      </v-flex>
    `,
  }},
}
}("CustomModuleName_DynamicID"));

```

What we do here is returning **vue instance** object with its own ID, which is assigned to every CustomerUser. It allows us to easily manipulate and save the user's own personal settings (If we provide such options).

Here, we used a simple call to our earlier defined frontend module and retrieve the new ticket number. Then, we displayed it in the material-stats-card template as a simple card.

Language support

Note, that we used here a mechanism to include a multilanguage translation. This will depend on the user's default language's setting. We will create now two different files needed to be included in the module to properly translate your labels.

ticket_widget_translation_strings.json

```

[
  ["Number of new tickets",

```

```
[ "Last update"
]
```

pl_TicketInformationWidget.pm

```
package Kernel::Language::pl_TicketInformationWidget;

use strict;
use warnings;
use utf8;

use vars qw(@ISA $VERSION);

sub Data {
    my $Self = shift;

    $Self->{Translation}->{"Number of new tickets"} = "Liczba nowych zgłoszeń";
    $Self->{Translation}->{"Last update"} = "Ostatnia aktualizacja";

    return 1;
}

1;
```

Note you can use every language you need - it's completely up to you!

Installation file

Now on the top of our project, we create installation file - **install.sopm**. It will be responsible for building our .opm package and merging our translation strings to the global list.

```
<?xml version="1.0" encoding="utf-8" ?>
<otrs_package version="1.0">
  <Name>Ticket Information</Name>
  <Version>1.0.0</Version>
  <Framework>6.0.x</Framework>
  <Vendor>CustomCompany</Vendor>
  <URL>https://www.custom_company.pl</URL>
  <License>GNU AFFERO GENERAL PUBLIC LICENSE Version 3, November 2007</License>
  <Description Lang="en">Module for Customer Panel.</Description>
  <IntroInstall Type="post" Lang="en" Title="Thank you!">Thank you for choosing the
```

```

Customer Widget.</IntroInstall>
  <BuildDate>?</BuildDate>
  <BuildHost>?</BuildHost>

  <Filelist>
    <File Permission="644"
Location="Kernel/Config/Files/XML/TicketInformation.xml"></File>
    <File Permission="644" Location="Kernel/Modules/TicketInformation.pm"></File>
    <File Permission="644"
Location="Kernel/Language/pl_TicketInformationWidget.pm"></File>

    <!-- FRONTEND -->
    <File Permission="644" Location="var/httpd/htdocs/customer-
panel/modules/WIDGET_OpenTickets.js"></File>

    <!-- TRANSLATION STRINGS -->
    <File Permission="644" Location="var/intalio-customer-
panel/ticket_widget_translation_strings.json"/>
  </Filelist>

  <CodeInstall Type="post"><![CDATA[
    use JSON::PP;
    use Data::Dumper;
    my $MainObject = $Kernel::OM->Get('Kernel::System::Main');
    my $GlobalTranslationStringsStrRef = $MainObject->FileRead(
      Location => '/opt/otrs/var/intalio-customer-
panel/translation_strings.json',
    );
    my $TranslationStrings = @{$GlobalTranslationStringsStrRef};

    my $json = JSON::PP->new->pretty->allow_nonref;
    $json = $json->allow_blessed;
    $json = $json->allow_unknown;
    $json = $json->convert_blessed;

    my @RequiredStrings = @{$json->decode( $TranslationStrings )};

    my $MyTranslationStringsStrRef = $MainObject->FileRead(
      Location => '/opt/otrs/var/intalio-customer-
panel/ticket_widget_translation_strings.json',

```



```

);
my $MyTranslationStrings = ${$MyTranslationStringsStrRef};
my @TranslationStrings = @{$json->decode( $MyTranslationStrings )};

print STDERR "RequiredStrings: ".Dumper(\@RequiredStrings);
print STDERR "TranslationStrings: ".Dumper(\@TranslationStrings);
@RequiredStrings = (@RequiredStrings, @TranslationStrings);

my $TranslationStringsResult = $json->encode( \@RequiredStrings );
print STDERR "Result: $TranslationStringsResult\n";

my $FileLocation = $MainObject->FileWrite(
    Directory => '/opt/otrs/var/intalio-customer-panel/',
    Filename   => "translation_strings.json",
    Content    => \$TranslationStringsResult,
);
]]></CodeInstall>

```

```

<CodeReinstall Type="post"><![CDATA[
    use JSON::PP;
    use Data::Dumper;
    my $MainObject = $Kernel::OM->Get('Kernel::System::Main');
    my $GlobalTranslationStringsStrRef = $MainObject->FileRead(
        Location => '/opt/otrs/var/intalio-customer-
panel/translation_strings.json',
    );
    my $TranslationStrings = ${$GlobalTranslationStringsStrRef};

    my $json = JSON::PP->new->pretty->allow_nonref;
    $json = $json->allow_blessed;
    $json = $json->allow_unknown;
    $json = $json->convert_blessed;

    my @RequiredStrings = @{$json->decode( $TranslationStrings )};

    my $MyTranslationStringsStrRef = $MainObject->FileRead(
        Location => '/opt/otrs/var/intalio-customer-
panel/ticket_widget_translation_strings.json',
    );
    my $MyTranslationStrings = ${$MyTranslationStringsStrRef};

```

```

my @TranslationStrings = @{$json->decode( $MyTranslationStrings )};

print STDERR "RequiredStrings: ".Dumper(\@RequiredStrings);
print STDERR "TranslationStrings: ".Dumper(\@TranslationStrings);
@RequiredStrings = (@RequiredStrings, @TranslationStrings);

my $TranslationStringsResult = $json->encode( \@RequiredStrings );
print STDERR "Result: $TranslationStringsResult\n";

my $FileLocation = $MainObject->FileWrite(
    Directory => '/opt/otrs/var/intalio-customer-panel/',
    Filename  => "translation_strings.json",
    Content   => \$TranslationStringsResult,
);
]]></CodeReinstall>

```

```

<CodeUpgrade Type="post"><![CDATA[
    ###
    # Dodanie nowych tłumaczeń tego modułu do globalnej listy fraz modułu otrs-
frontend.
    ###
    use JSON: :PP;
    use Data: :Dumper;
    my $MainObject = $Kernel::OM->Get('Kernel::System::Main');
    my $GlobalTranslationStringsStrRef = $MainObject->FileRead(
        Location => '/opt/otrs/var/intalio-customer-
panel/translation_strings.json',
    );
    my $TranslationStrings = @{$GlobalTranslationStringsStrRef};

    my $json = JSON: :PP->new->pretty->allow_nonref;
    $json = $json->allow_blessed;
    $json = $json->allow_unknown;
    $json = $json->convert_blessed;

    my @RequiredStrings = @{$json->decode( $TranslationStrings )};

    my $MyTranslationStringsStrRef = $MainObject->FileRead(
        Location => '/opt/otrs/var/intalio-customer-
panel/ticket_widget_translation_strings.json',

```

```

);
my $MyTranslationStrings = ${$MyTranslationStringsStrRef};
my @TranslationStrings = @{$json->decode( $MyTranslationStrings )};

print STDERR "RequiredStrings: ". Dumper(\@RequiredStrings);
print STDERR "TranslationStrings: ". Dumper(\@TranslationStrings);
@RequiredStrings = (@RequiredStrings, @TranslationStrings);

my $TranslationStringsResult = $json->encode( \@RequiredStrings );
print STDERR "Result: $TranslationStringsResult\n";

my $FileLocation = $MainObject->FileWrite(
    Directory => '/opt/otrs/var/intalio-customer-panel/',
    Filename  => "translation_strings.json",
    Content   => \$TranslationStringsResult,
);
]]></CodeUpgrade>

```

</otrs_package>

OTRS Installation

Now we can build our package and deploy it to the system.

```
otrs> /opt/otrs/bin/otrs.Console.pl Dev::Package::Build --module-directory ./ install.sopm ./
```

The following command will generate complete installation package with **.opm** file extension

Now everything left to do is to install generated package with one of the provided methods:

The following instruction explains how to install the package using one of the provided methods.

1. Admin Interface

Log in to your system as user with admin privileges and go to menu Admin ⇒ Package Manager. Select module file (with .opm extension) in the "Actions" panel and click "Install Package" button.

2. Command line

```
otrs> /opt/otrs/bin/otrs.Console.pl Admin::Package::Install /path/to/package/.opm
```

Summary

As You might see writiring your own widget is pretty straightforward and easy. It enables You to extend your CustomerPanel with powerful tools that highly depends on your needs.

Revision #11

Created Mon, Aug 10, 2020 1:25 PM by [editor](#)

Updated Wed, Oct 28, 2020 2:30 PM by [editor](#)