Define a View

Views in PlatformUI

Each route defines a View to render when the route is matched. As explained in the PlatformUI technical introduction, this kind of view is called a **Main View**. Like any view, it can have an arbitrary number of sub-views.

> **Good practice:** keep the views small and do not hesitate to create sub-views. This eases the maintenance and allows you to reuse the small sub-views in different context.

A View is responsible for generating the User Interface and handling the user input (click, keyboard, drag and drop, etc.). In its lifecycle, a view first receives a set of parameters, then it is rendered and added to DOM.

> PlatformUI reuses the YUI View system. `Y.View` is designed to be a very simple component. PlatformUI extends this concept to have views with more features like templates or asynchronous behavior.

Creating and using a custom Main View

Creating a new View

As for the plugin in the previous step, the first thing to do is to declare a new module in the extension bundle `yui.yml` file:

New module in yui.yml

```
ezconf-listview:
    requires: ['ez-view']
    path: %extending_platformui.public_dir%/js/views/ezconf-listview.js
```

Our basic view will extend the PlatformUI basic view which is available in the `ez-view` module. As a result, `ez-view` has to be added in the module requirements.

Then we have to create the corresponding module file. In this file, we'll create a new View component by extending `Y.eZ.View` provided by `ez-view`.

%extending_platformui.public_dir%/js/views/ezconf-listview.js

```
YUI.add('ezconf-listview', function (Y) {
    Y.namespace('eZConf');

    Y.eZConf.ListView = Y.Base.create('ezconfListView', Y.eZ.View, [], {
        initializer: function () {
            console.log("Hey, I'm the list view");
        },

        render: function () {
            // this.get('container') is an auto generated <div>
            // here, it's not yet in the DOM of the page and it will be added
            // after the execution of render().
            this.get('container').setContent(
                "Hey, I'm the listView and I was rendered it seems"
            );
            this.get('container').setStyles({
                background: '#fff',
                fontSize: '200%',
            });
            return this;
        },
    });
});
```

This code creates the `Y.eZConf.ListView` by extending `Y.eZ.View`. The newly created view component has a custom `render` method. As its name suggests, this method is called when the view needs to be rendered. For now, we are directly manipulating the DOM. `this.get('container')` in a View allows you to retrieve the container DOM node, it's actually a Y.Node instance that is automatically created and added to the page when the View is needed in the application.

> A View is responsible for handling only what happens in its own container. While it's technically possible, it is a very bad practice for a View to handle anything that is outside its own container.

At this point, if you open/refresh PlatformUI in your browser, nothing will have really changed, because the newly added View is not used anywhere yet.

Using it as a Main View

Now that we have a View, it's time to change the route configuration added in the previous step so that our custom route uses it. To do that, we'll have to change the application plugin to register the new view as a main view in the application and then to define the custom route with that view. Since we want to use the new view in the plugin, the `ezconf-listview` module becomes a dependency of the plugin module. The plugin module declaration becomes:

```
ezconf-listapplugin:
    requires: ['ez-pluginregistry', 'plugin', 'base', 'ezconf-listview'] # we've added
'ezconf-listview'
    dependencyOf: ['ez-platformuiapp']
    path: %extending_platformui.public_dir%/js/apps/plugins/ezconf-listappplugin.js
```

Then in the plugin, `Y.eZConf.ListView` becomes available; we can register it as a potential main view and change the route to use it:

ezconf-listappplugin.js

```
YUI.add('ezconf-listapplugin', function (Y) {
    Y.namespace('eZConf.Plugin');

    Y.eZConf.Plugin.ListAppPlugin = Y.Base.create('ezconfListAppPlugin',
Y.Plugin.Base, [], {
        initializer: function () {
            var app = this.get('host'); // the plugged object is called host

            console.log("Hey, I'm a plugin for PlatformUI App!");
            console.log("And I'm plugged in ", app);

            console.log('Registering the ezconfListView in the app');
            app.views.ezconfListView = {
                type: Y.eZConf.ListView,
            };

            console.log("Let's add a route");
            app.route({
                name: "eZConfList",
                path: "/ezconf/list",
                view: "ezconfListView", // because we registered the view in
app.views.ezconfListView
                sideViews: {'navigationHub': true, 'discoveryBar': false},
                callbacks: ['open', 'checkUser', 'handleSideViews', 'handleMainView'],
            });
        },
    }, {
        NS: 'ezconfTypeApp' // don't forget that
    });

    Y.eZ.PluginRegistry.registerPlugin(
        Y.eZConf.Plugin.ListAppPlugin, ['platformuiApp']
    );
});
```

After doing that, `/ez#/ezconf/list` should no longer display the dashboard, but you should see the message generated by `Y.eZConf.ListView`.

Using a template

Manipulating the DOM in a view render method is fine for small changes but not very handy as soon as you want to generate a more complex UI. It's also a great way to separate pure UI/markup code from the actual view logic.

> In PlatformUI, most views are using a template to generate their own markup, those templates are interpreted by the Handlebars.js template engine embed into YUI.

The templates are made available in the application by defining them as modules in `yui.yml`. The template modules are a bit special though. To be recognized as a template and correctly handled by the application, a template module has a `type` property that should be set to `template`. Also, to get everything ready *automatically*, the module name should follow a naming convention. The module name should consist of the lowercase View internal name (the first `Y.Base.create` parameter) where the template is supposed to be used followed by the suffix `-ez-template`. In our case, the internal name of `Y.eZConf.ListView` is `ezconfListView`, so if we want to use a template in this view, the module for this template should be `ezconflistview-ez-template`. As a result, the module declaration for the template module will be:

```
ezconflistview-ez-template: # internal view name + '-ez-template' suffix
    type: 'template' # mandatory so that the template is available in JavaScript
    path: %extending_platformui.public_dir%/templates/ezconflistview.hbt
```

In `yui.yml`, we also have to change the `ezconf-listview` module to now require `ez-templatebasedview` instead of `ez-view` so that `Y.eZConf.ListView` can inherit from `Y.eZ.TemplateBasedView` instead of `Y.eZ.View.` Once that in place, the rendering logic can be changed to use the template:

```
YUI.add('ezconf-listview', function (Y) {
    Y.namespace('eZConf');

    Y.eZConf.ListView = Y.Base.create('ezconfListView', Y.eZ.TemplateBasedView, [], {
// Y.eZ.TemplateBasedView now!
        initializer: function () {
            console.log("Hey, I'm the list view");
        },

        render: function () {
            // when extending Y.eZ.TemplateBasedView
            // this.template is the result of the template
            // compilation, it's a function. You can pass an object
            // in parameters and each property will be available in the template
            // as a variable named after the property.
            this.get('container').setHTML(
                this.template({
                    "name": "listView"
                })
            );
            this.get('container').setStyles({
                background: '#fff',
                fontSize: '200%',
            });
            return this;
        },
    });
});
```

And the last part of the chain is the Handlebars template file itself:

ezconflistview.hbt

```
<h1 class="ezconf-list-title">List view</h1>

Hey, I'm the {{ name }} and I'm rendered with the template!
```

At this point, `/ez#/ezconf/list` should display the `Y.eZConf.ListView` rendered with the template.

Adding a CSS

We've just moved the markup logic from the view component to a template file. It is also time to do the same for the CSS styles that are still in the `render` method. For that, a CSS file can be created on the disk to replace the inline styles:

list.css

```
.ez-view-ezconflistview {
    background: #fff;
}

.ez-view-ezconflistview .ezconf-list-title {
    margin-top: 0;
}
```

By default, a view container element has an auto-generated class built from the internal view name.

Good practice: using that auto-generated class to write the CSS rule greatly limits the risk of side effects when styling a view.

Then this file has to be listed in the extension bundle `css.yml` configuration file:

css.yml

```
system:
    default:
        css:
            files:
                - %extending_platformui.css_dir%/views/list.css
```

After this is done, a custom view is now used when reaching `/ez#/ezconf/list` and the UI is now styled with a custom external stylesheet.

Results and next step:
The resulting code can be seen in the 4_view tag on GitHub, this step result can also be viewed as a diff between tags `3_routing` and `4_view`.

The next step is then to configure the navigation so that user can easily reach the new page.