**Argosy.Web.FrontEnd.Models**

Comment: This one is used for NavigationAction class. Adding new NavigationAction objects to ViewBag.Navigation list that is already defined in the GlobalViewFilter.Those are links which are located bellow main navigation.

**Argosy.BusinessLogic.FrontEnd.Managers.ThemeManager.ThemesViewModel**

By including a **@model** statement at the top of the view template file, you can specify the type of object that the view expects, in our case it is **ThemeViewModel** in **ThemeManager**.cs file. This **@model** directive allows you to access object type **ThemeViewModel,** and as a result returns all the data(in this case all Theme) from database to show this view (page).

*@{*

*ViewBag.Title = "~{ThemePicker}~";*

*Layout = "~/Views/Shared/\_Layout.cshtml";*

*ViewBag.Navigation.Add(new NavigationAction { Name = "~{Admin}~", InnerClass = "active" });*

*ViewBag.Navigation.Add(new NavigationAction { Name = "~{ThemePicker}~", InnerClass = "active" });*

*}*

Code that we start with **@ {}** means that we can write a C# code in html, brackets mean that we can write more lines of code. In this case, I have provided a title that will be dynamically displayed in the browser tab, links and layout that we use for this page.

*@{*

*var standardThemesGroup = Model.ThemeGroupings.FirstOrDefault(tg => tg.Name.Equals("Standard Themes"));*

*var createdThemesGroup = Model.ThemeGroupings.FirstOrDefault(tg => tg.Name.Equals("Created Themes"));*

*var showCreatedThemes = createdThemesGroup != null;*

*}*

In this C# code, from the model obtained in this case type **ThemesViewModel,** we are looking for a first or a default name, and with lambda query we ask if this is a name from the model equally fixed to the given name. In the background, the java script code will in that html div to show the names of the group's themes, in this case it will be Standard and Created.

<ul class="tc-theme-list" id="StandardThemes">

@foreach (var themeInfo in standardThemesGroup.ThemeInfoList) {

<li class="tc-item" id="@themeInfo.ThemeId">

<div class="input-style">

<input name="themegroupnvalue="@themeInfo.ThemeId"

@if (themeInfo.ThemeId == Model.CurrentThemeId)

{

@: checked }

class="bootstrap-toggle"

type="radio"

id="@themeInfo.ThemeId"

onclick="changeCSS('/Style/ChangeCss/@themeInfo.ThemeId')" />

<label class="tc-name" for="@themeInfo.ThemeId">

@themeInfo.Name

@if (themeInfo.IsDefault)

{

@:(Default)

}

</label>

</div>

<div class="label-style">

<span class="tc-color" name="primary"

style="background-color: @themeInfo.PrimaryColor"></span>

<span class="tc-color" name="secondary"

style="background-color: @themeInfo.SecondaryColor"></span>

<span class="tc-color" name="tertiary"

style="background-color: @themeInfo.TertiaryColor"></span>

<span class="marl10">

<a data-argosy-dataid="@themeInfo.ThemeId" data-argosy-action="editTheme" id="showBuildTheme">

<i title="Add new Theme with these styles" class="fa fa-plus" data-id="@themeInfo.ThemeId"></i>

</a>

</span>

</div>

</li>

}

In this part of the code idea was to charge **ul list** with data from the model that we received from the database. The web page view is realized via the AJAX call from the **Theme.js** file. That AJAX type is POST which sends the request **DataView** controller and call the function **GetTheme()** (DataView/GetTheme). As a result, JSON data is expected. And javascript then displays the HTML content with that obtained JSON data. For the **created themes** everything is similar to the **standard theme**, the only difference is that it is asked if there is at least one theme created that JS would know if there is something to display in html.

Radio buttons that are created with these themes in their **input** tag as an attribute has the specified event onclick= *"changeCSS('/Style/ChangeCss/@themeInfo.ThemeId')"* (**Theme.js**). This event tells us that when you click on the **radio button,** the css website styling changes. This is achieved by sending AJAX request to the type of **GET** and who calls **StyleControler** and its action **ChangeCss** (Style/ChangeCSS). This **ChangeCSS** action is passed to the id parameter, which is the type int. By this, we will change the css style of the web page so as an AJAX response we get the appropriate style for the selected theme. All of the css styles we are looking for are in the database.

When clicking on the link **Build your own theme** or on **+** next to theme, and by selecting some of the themes from the pop-up dialog, AJAX POST is sent to the DataView and the GetTheme function gets 1 record ie json data, for this theme we have chosen and gives clients the ability to modify it before clicking the button for making.

It also sends ajax request to the Style controller and calls the ChangeCss method, it is assigned the parameter id type int, in order to result in changing the css style when choosing a theme for creation. When clicking the button **save new theme**, data is collected and AJAX by request the type of POST is sent to the **Create** action in the **Theme** controller to execute. When the theme is entered the user is redirected to the same page ie a website is refreshed and then the topic of the **created themes** also shows the theme that the user has entered.