ThemeController

**ThemeController.cs**

**Action:**

**1.Index**

**1.1 Description of the functionality of the page**

The Admin/Theme/index/1 page displays two types of templates for the user, created templates created by the user and standard that exist in the web application. By choosing a template (clicking on a radio button), the user can change, or stylize, the appearance of a web application (website).

**1.2 Add code, comment, remove or edit code in action**

Within the action index, the following code line is commented *var viewModel = ThemeManager.GetViewModel();*    
It is commented because it is a connection with the database, and ThemeManager class and accesses method GetViewMethod() that as a result returns all the topics that exist in the database.The result of which is obtained in the controller forward the View and so we get the appearance of web pages. Since we do not currently work with the database, I have set up fixed data.

First, make a reference **viewModel** to an object of type *ThemeViewModel (new ThemeViewModel())* and by this reference we approach the properties of that object. Since one of the properties of type List, I made two lists, a **lista** that will contain standard themes and a **lista2** that will contain all the topics that the user made. Both lists are filled with fixed values by creating a new instance of the *ThemeInfo()*, and in that object, its properties that I indicate the fixed data that will eventually be displayed in the browser. Finally, since the reference **viewModel** is the ThemesViewModel type, we add the data it expects, which are the property of the **name** and the list (**lista**,**lista2**) we made and filled with fixed data. The last line of code *return View(viewModel)* in the Action index, I'm forwarding the data to view, and in this way we get a visual view of the page, in this case they are standard and created themes.

**1.2.1 Add dependedancies classes**

In the folder AdditionalContent I created folder Areas and in it I created the Admin folder and in the Admin I created the folder Theme. In the theme folder I added the following classes that I needed to process the index actions in the controller: **ThemeManager.cs, IObjectMap.cs, ESM\_THEME.cs, ESM\_THEME\_STYLE.cs, ESM\_THEME\_OPTION.cs, ESM\_THEME\_GROUP.cs, ESM\_THEME\_STRUCTURE.cs.**

**ThemeManager.cs** contains several methods that are some kind of services that are called for each event. In the class **ThemeManager.cs**, I commented everything except for the following classes: **Theme.cs, ThemeStructure.cs, ThemeGroup.cs, ThemeStyle.cs,ThemeOption.cs, ThemeGrouping.cs, ThemeInfo.cs, ThemeEnum.cs.**

**IObjectMap.cs** contained interface in which the generic type is stated. Each of the classes I stated that I did not comment are inherited from the **IObemtmap** in which the generic type is stated.

**2. Create**

**2.1 Description of the functionality of the page**

When a user wants to add a new topic, he should click on the link called *BuildOwnTheme.* When a user clicks on that link, it displays a dialogue with existing topics that belong to the group of the *standard themes*. When a dialog appears, select one of the links (*UseStyle*), the stylization of this theme is applied on the site and the user has the opportunity to review the theme (click on the link *Preview*), back to existing Themes and Create that new theme (click on the link *Save as new Theme*When a user creates a theme, AJAX request is a type POST, the data is sent to the database and the user is redirected to the page Admin/Theme/index/1 where in the part *CreatedTheme* see also this new theme.

**2.2 Add code, comment, remove or edit code in action**

In the class **Theme.cs** in ThemeManager.cs file, I made the constructor without **parameters public Theme(),** so that I can call this constructor to *Create* the action and send him fixed data that will be sent by Ajax. In action *Create,* I made a list called **listaStilova** and set fixed values in it. I made a reference **response** to a theme type object,and forwarded to the constructor all the values that he expected. And finally, we put this reference in return and the result is expected in the form of json.

**2.2.1 Add dependedancies classes**

For this action, I had to create a json file called **GetThemes.json** in folder Controller/JSON. On web page [*www.vanityurlportal.com/Admin/Theme*](http://www.vanityurlportal.com/Admin/Theme) , with the inspect element tools I captured the response request when creating a theme and this result of the response I copied into that file, so that I can simulate the creation action.

In View/Shared/Partials/Bundles/**Main**.**cshtml,** I commented the next line code:

**Var userSettings=Html.Raw(SerializerFactory.GetSerializer(SerializationType.JSON).Serialize(FrontEndSession.Instance.UserSettings));**

And this commented line I replace with **var userSettings=null;**

Then In file *SessionPrimer.cs*  I added **set** the following properties (HashedUserId, HashedCompanyId, HashedSiteId, HashedCompanyUserGroupId).

In file **FrontEndSession.cs**

**StyleController.cs** I added the next line codes:

*HashedUserId = "bS3dHFgEdww=",*

*HashedCompanyId = "9E3/sF5hr4I=",*

*HashedSiteId = "vvme335qhTA=",*

*HashedCompanyUserGroupId = "kGBQ3n7RIkQ="*

The point of this was to be able to load some js files

**3. Update and Delete**

**3.1 Description of the functionality**

The **update** action are invited 2 JS files, when you open the *edit theme* (view theme), **controls.min.js** are loaded, and when updating the theme **jqueryLibs.min.js and jquery-3.2.1.js** is loaded to send ajax request.

In the insect element tool I found that for this controller when deleting and updating themes when I need to enter the edit mode, to display the *delete theme* button or *save theme* button, the controller **Style** should be include and in his method the **index** expects parameter **id** so I could not simulate this page with fixed values, and do not show me the *save theme* button (update) and *delete theme* in my project for that.

When updating a company theme, By clicking the *set company theme* button the **UpdateCompanyTheme** function in the **Theme** Controller is called and the ID is forwarded as a parameter. Since we do not work with the database, it is impossible to simulate this action, however, it is necessary to load the **CompanyManager** class, because it uses its sinus functions.

In the body of this function we also have a call function **update** where we are forwarding the *id theme* and the *id company* and for this reason, we must also include the **BaseManager** class in order for this **update** to work. **UpdateCompanyTheme** is done by sending AJAX request which is the type of GET and in the **updateSessionTheme** function, this change is remembered as the **updateSessionTheme** parameter specifies the **id** theme that is selected.

When deleting a theme, you must enter the edit mode as well as for the update. To display a page in edit mode, the js file *jQueryLibs.min.js* and *controls.min.js* must be loaded in the project. Deletion is done via ajax request, which is a type of post, contacts in the **ThemeControler** function **delete**, forward the theme that we want to delete and finally in the successful action as a result received a message that the theme of the group *Created Themes* successfully deleted.

**StyleController.cs**

**Action:**

**1. ChangeCss**

**1.1 Description of the functionality**

This controller and action ChangeCss is important when a user clicks the radio button at the Admin/Theme/index/1 page, and chooses some of the themes to change the site's appearance (for example, his css).

**1.2 Add code, comment, remove or edit code in action**

I've commented on the following lines of code:

*- //var themeManager = new ArgosyManager<ESM\_THEME>();*

*- // IQueryable<ESM\_THEME> theme;*

*-//theme = themeManager.Find(t => t.THEME\_ID == id);*

*-//styleSheet = theme.Select(t => t.STYLESHEET).FirstOrDefault();*

I also commented on the *StyleController* all actions except this action *ChangeCss*.

I've added two lists, **listaStilova**(type **ESM\_THEME**)and this list is important because it will contain hardcoded data that will replace those that are derived from the database and the second list **stilTema** (type **ESM\_THEME\_STYLE**) which will be passed to the **ESM\_THEME\_STYLE** property from the first list.

*listaStilova.Select(t => t.STYLESHEET).FirstOrDefault();* By this line of code I executed the linq query over the list and as a result I added a reference styleSheet to the next line of code.

A variable **file** is created in which we are reading the whole file style.css, ie the css code that is in it.

A new css file called **style.css** was created in the Content folder/dodato. And in that created css file the css data we captured from the browser was copied when testing the action on web site [*www.vanityurlportal.com*](http://www.vanityurlportal.com)*.* This has to be done for the test because all the styles in the original project are obtained from the database (there are no physical files in the project).

**2. Index**

**2.2 Add code, comment, remove or edit code in action**

In the action **Index** I used the copied part of the code from the action **changecss**, while I left the rest of the code in the **index** code commented. I had to do this to solve the error that I encountered when entering the edit mode and which required some parameter to pass the **index** function in the **Style** controller.