# Workshop: Forum

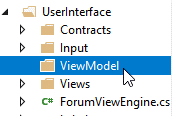
## Overview

In the previous two parts we’ve implemented the Data management and signup and login functionality. In this part comes the real fun. We will implement adding and displaying categories, posts and their replies.

## Creating View Models

In almost every architecture with user interface **view models** are used in order to make displaying the information from the models ([DTO](https://en.wikipedia.org/wiki/Data_transfer_object)) easier. In general view models differ from the entities in the fact that they only store information that is significant for **view** to display and they can store logic of **formatting** that information. For example, the PostViewModel you’re about to implement, doesn’t care about its author or category id. It will keep strings with their names instead. Enough talking, let’s write some code and it will come to you.

Now first create a ViewModels folder in UserInterface directory. There we will store our view models



### Post View Model

In the ViewModels directory you’ve just created add a new public class called PostViewModel. First we need to add a private constant of type int called LINE\_LENGTH. We are going to need it for wrapping our text to fit the view layout.

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_19-36-40.png

We continue with the properties of our view model. You’ll be surprised that they look a lot like the Post model from your **Models** project. First one is an integer and we are going to name it PostId:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_20-01-03.png

Next comes the title which is the same:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_20-01-52.png

After that instead of integer that keeps the author id we will keep his name as string:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_20-05-04.png

We do the same for the category name:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_20-05-36.png

Next, we will define a property for our content, but instead of keeping it in a single string, we will create an IList<string>:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_20-06-24.png

Now we have to make an IList of ReplyViewModels in order to keep our replies, create the ReplyViewModel class in the same folder (we’ll be there in a bit) and come back to implement it:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_20-07-22.png

The last thing for now is a text wrapper method we are going to call GetLines. It will be of type IList<string> and will take the content string as a parameter.

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_20-08-32.png

The implementation is simple. Fist we take our string and covert it to a nice comfy **array of characters**, like so:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_20-09-35.png

Next, we are going to make ourselves a List<string> which we will return in the end:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_20-10-14.png

Now the logic of the method is going to take place in a **for** loop statement, whose counter value will be assigned to zero, it’s condition will be that the value should be less than the content length (parameter) and the value will be incremented with LINE\_LENGTH’s value on each iteration. Then all you have to do is **calculate the characters** that has to be on the current row, concatenate them to a **single string** and add them to that list of yours you’ve created. I will appreciate if you try and do it yourself. In any case, there is an implementation coming right up.

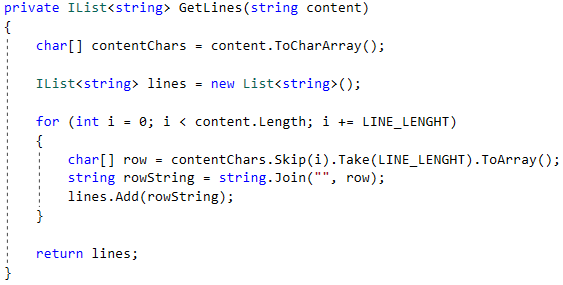
This is how the loop should look like:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_20-12-03.png

This way you can calculate the line that you are on:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_20-12-32.png

This is hot the whole method should look like:



That’s it for now, we’ll be back later when we finish our services. Now let’s proceed to the ReplyViewModel.

### Reply View Model

Again, we start with the LINE\_LENGTH constant and is going to be the same as in Post View Model because we want our text to look consistent in our app.



Next, we define a property which is a string and holds the Author’s username:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_20-14-17.png

And a property that holds the content which will be of type… guess what… IList<string>:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_20-06-24.png

Next, we will need a text wrapper that will take a string content and return IList<string> representing the lines. Sounds familiar? You can Copy and Paste it from the previous View model, and let’s hope you didn’t mess it all up the first time, right? ☺

That’s all we could do to our view models without the help of our services (remember?). We will be back to finish them in a bit.

## Proceed the Implementation of Service Layer

In this part we’re going to finish the functionality of our service layer.

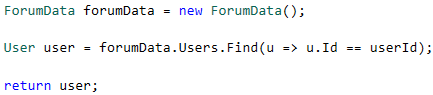
### User Service

We have created most of the stuff here in the previous part. The new thing that we need our **user service** to provide now is to fetch an existing user for us, and we are going to create two overloads of the same method. First one is going to search by **username** and the second by **id**.

First, we create a new public static method that returns a User and takes an int id as a parameter:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_20-18-13.png

In the method instantiate an object of type ForumData, find in its users the one with the same **id** and **return** it. It should look something like this:



Having this, implement an **overload** to this method that takes string username as a parameter.

#### Finish Reply View Model

Just go back to **Reply View Model** and add a constructor that takes a **Reply** as an argument. First set the author name calling the UserService‘s corresponding method:

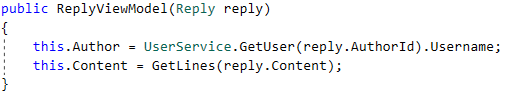
C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_20-21-35.png

Then set the content lines using the method you write in **Part I.** of this document:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_20-22-14.png

Finally add an empty constructor that takes no arguments.

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_20-23-02.png



### Post Service

Create a new public static class called PostService. For now, we are going to implement just two methods in order to finish the PostViewModel Implementation. Don’t worry, we’ll add more functionality with the post data later.

The first method we are going to implement will fetch us a category model found by **id**. It’s called GetCategory and takes an **id** as a parameter:



Since you saw how the GetUser method works, I am sure you can handle this.

And the second method is called GetPostReplies with parameter postId and return type IList<ReplyViewModel>. The task is simple you need to find the post by **id**, fetch it’s replies, instantiate a ReplyViewModel with each of them and return them as an IList. Easy.

First you create an object of good old ForumData.

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_20-27-56.png

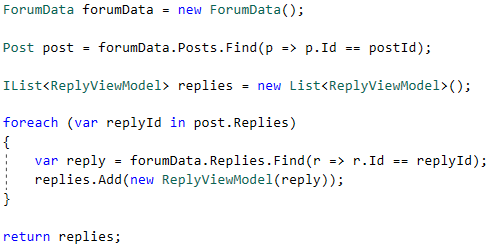
Get the post with the given **id.**

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_20-28-33.png

Create an IList<ReplyViewModel> that you are going to fill.

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_20-29-18.png

In a foreach loop through current post’s replies **id**, find the post (in ForumData) with the same **id**, make RepleyViewModelObject with it and add it to the list you’ve just created. Finally, you should return the list. All of it looks like this.



#### Finish Post View Model

The thing we have to finish here are the constructors. Add one that takes no arguments and sets the Content to new **List of string**. There won’t be a screenshot, I **believe** in you.

The second constructor is going to take a Post model as an argument and it will:

Set view model’s post id and title:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_20-34-06.png

Set the content lines with the result of get lines with post.Content as parameter:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_20-34-45.png

Fetching the author name by **id** from **user service:**

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_20-35-24.png

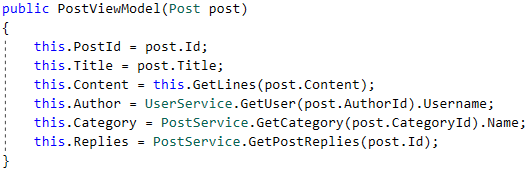
Calling the method for getting the category form post service

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_20-35-47.png

Getting the replies by **id** from post service:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_20-36-38.png

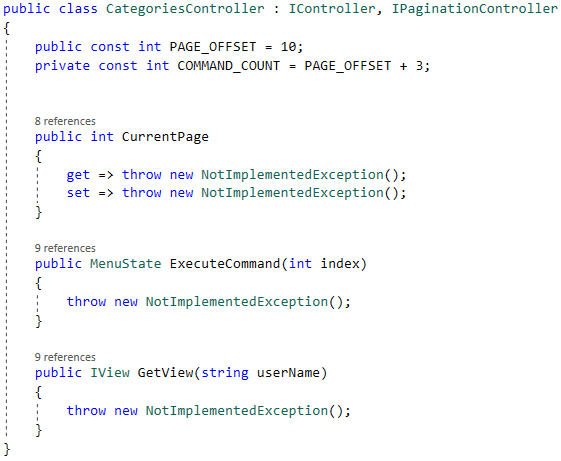
All of it:



Now that we’re ready with the view models and most of the Services we can dive into implementing controllers.

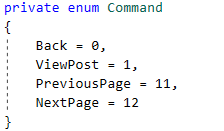
## Implementing Categories Controller

Open the CategoriesController class, it should look like this:



### Helpers

We are going to need an enum once again, but this time we’re going to set its integer values, because the count of the categories may vary:



### Properties

First remove CurrentPage’s NotImplementedException so it becomes like this:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_21-14-27.png

You will need two **array** properties, one to keep all the category names:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_22-50-18.png

And one for the ones on the current page:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_22-51-03.png

Then there’s a property that calculates what the index of the last page is:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_22-55-09.png

And two Boolean properties that determine whether your controller is on the **first page**:

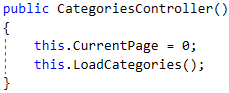
C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_21-29-18.png

Or the **last one**:

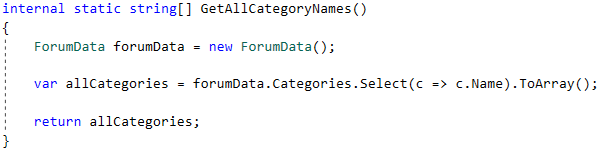
C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_21-29-57.png

### Methods

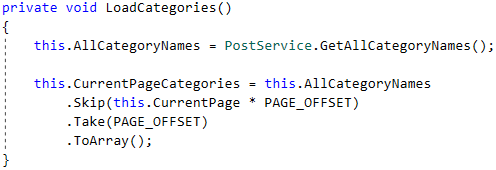
The next method is called ChangePage and changes the CurrentPage depending on a **boolean** that is given to it as a parameter. This is the implementation:



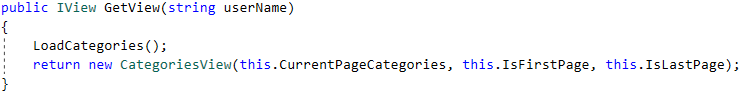
For the next method we first will have to implement a service method in PostService whose job will be to fetch all the categories from ForumData and **return** their names as an array of strings. It’s called GetAllGategoryNames and the implementation is something like this:



Now we can return to our controller and continue with a method called LoadCategories and its task will be to call the service method we’ve just wrote, set the result to AllCategoryNames and then the set the **current ones**. The implementation goes like this:

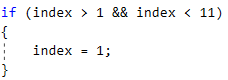


The GetView method’s job is to call LoadCategories and **return new view**

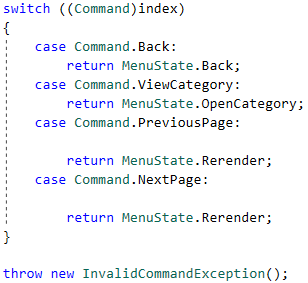


### Execute Command

Once again, we’re in the method that decides which command should be implemented depending on the menu state. There is a slight difference however. If the **index** that we get as a parameter is **between 1 and 10** that means the cursor is on a **post** and its value should be set to 1 (take a look at the enum you’ve implemented). This is done by a simple if statement:



For the next step I again will rely on magic abilities and expect that crafting such statement won’t cost you any trouble.



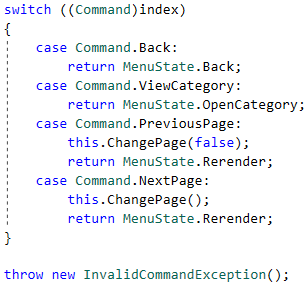
All of the commands you need to invoke are PreviousPage and NextPage… And is actually just one method, right? So, in case of NextPage call the ChangePage method with no parameters since its parameter has a default value:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_23-18-12.png

And PreviousPage calls it with false as a parameter:

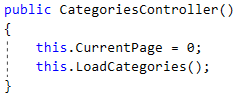
C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_23-17-50.png

The whole thing should look like this:



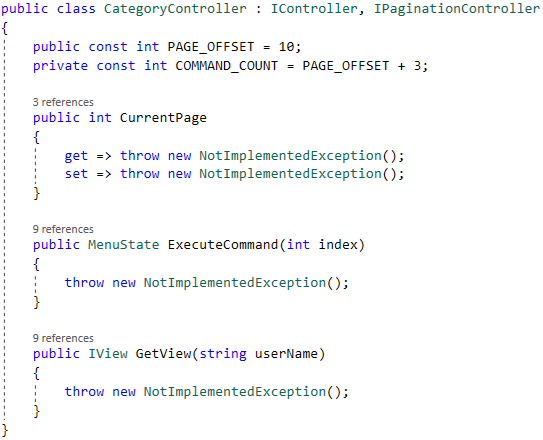
### Constructor

All the constructor does is setting CurrentPage to 0 and calling LoadCategories method:



## Implementing Category Controller

Now, go to category controller and you should find something like this:



### Helpers

The enum is exactly the same with the previous controller.

### Properties

First remove CurrentPage’s NotImplementedException so it becomes like this: C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_21-14-27.png

Now there are somethings we need to have in our controller. One of them is array of string holding every **post’s title**:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_21-19-44.png

For the next property you need to calculate the index of the last page of the posts so you can check if you are on the last page further in the implementation:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_21-27-56.png

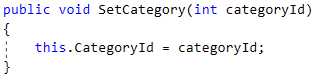
Next two are Boolean properties that check if you are on the first page or the last page and they are the same as in CategoriesController.

The last property will be public and holding the category id:

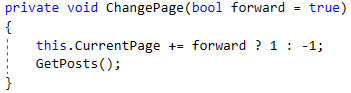
C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_21-30-59.png

### Methods

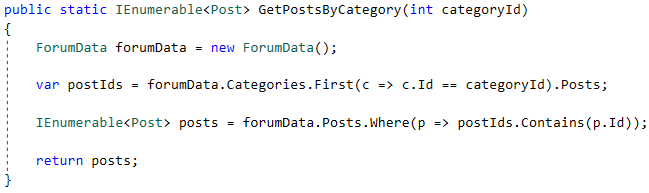
The first method we are going to realize in this controller is the public void SetCategory method taking int id as an argument. Its job is simple: just sets the corresponding property with the value of the parameter it is given:



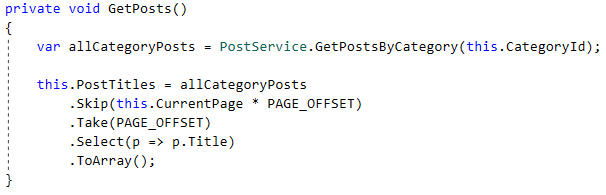
The next method is called ChangePage and changes the CurrentPage depending on a boolean that is given to it as a parameter. Sounds familiar?



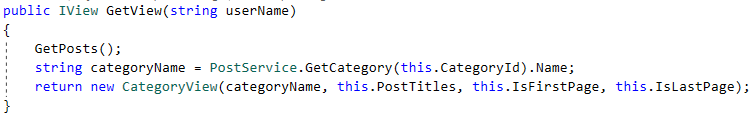
For our next method we will need a post service we need to implement first. So, go to the PostService and implement a method of type IEnumerable<Post>, called GetPostsByCategory and takes categoryId as a parameter. All it should do is get the Category’s post ids and then filter the posts models with the given ids and return the result. Easy, right? Here is a screenshot just for sure:



Now that you have that service you can implement GetPosts method that takes all the posts and assigns them to the PostTitles property depending on the page that the controller is currently on. It should look something like this:

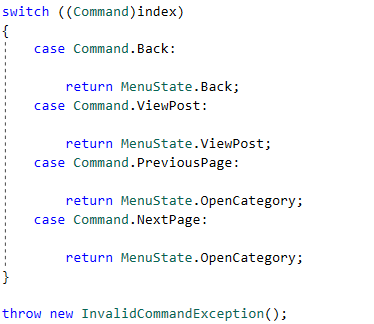


Next in the line is GetView method who calls GetPosts method to be sure that current PostsTitles are the ones that the controller needs. Gets the **category** **name** from the service we’ve implemented and returns new CategoryView with the data:



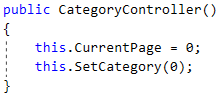
### Execute Command

Identically with ExecuteCommand in the previous view, just change pages and return states show below:



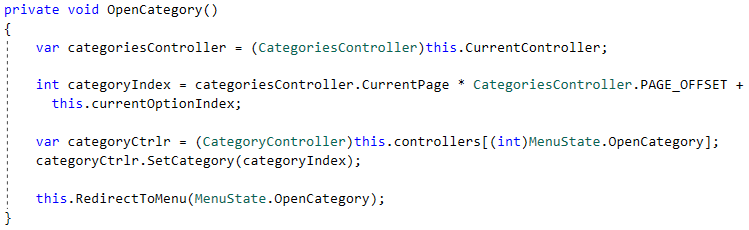
### Constructor

Slightly different from categories controller **constructor** it sets current page to 0 and call SetCategory with parameter 0, like this:



### Menu Controller’s Open Category

The one last thing we need to implement before testing your categories and category controllers is the OpenCategory method in menu controller. It does very important job. It determines the **category** you want to open by the **index of the current object** and gives the CategoryController the category’s index. This is the implementation:



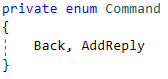
Now you can add manually some categories and test what you’ve done.

## Implementing Post Details Controller

This controller returns a PostDetailsView and tells it whether you’re logged or not. Let get straight to implementation:

### Helpers

This time the controller’s executable commands are only two:



### Properties

First of all, get rid of NotImplementedException that LoggedInUser throws and make it normal automatic property.

After that add public PostId property, like so:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-08_00-05-57.png

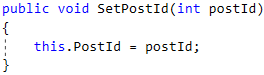
### Methods

UserLogIn method’s job is to set LoggedInUser property to true, like this:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-08_00-09-09.png

Guess what UserLogOut does…

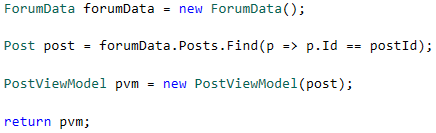
The next method you need to implement is the public void SetPostId method and you’ll never guess what it does:



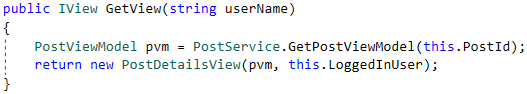
Now to implement the GetView method we need a PostViewModel to give the view. That’s why we are going to implement ourselves a **service method** that does that. Go to PostService and create a public static method of type PostViewModel that takes int id as an argument, like this:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-08_00-20-13.png

The logic is simple, find a post with the given **id**, create PostViewModel with it, and returns it:



Now we can implement the GetView method:



### Execute Command

Again, we have to implement the switch that returns different menu state depending on the index of the command. Since we’ve got only two commands in our **enum** we will have only two cases in our **switch**. The AddReply case should return AddReplyPost MenuState:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-08_11-45-34.png

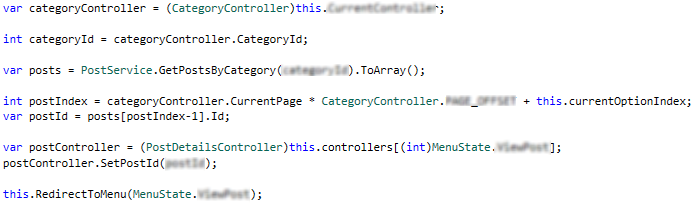
Next case is Back command and all it should do is reset ForumViewEngine’s Buffer and return the corresponding MenuState:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-08_11-45-09.png

That’s all you have to do here. Just don’t forget to throw Exception if the index doesn’t match any case.

### MenuController’s View Post method

The method you are about to implement looks a lot like Open Category method. Its job is to get the **category id** form the CurrentController. Fetch all the posts with the current **category id**. Calculates **the index of the post you want to view and gets his index**. Fetches the post controller at index Menu.ViewPost (you remember they are integers, right?), and call the RedirectToMenu with MenuState.ViewPost:



By now, you should be able to view a post that you have inserted **manually**. Try it!

## Implementing Add Post Controller

By now we have working functionality of logging in, signing up, and viewing our categories and posts. Now we shall step into implementing the most important functionality of our forum, adding posts.

### Fields

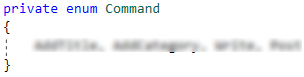
Take a look at the fields you are provided with by default, their names **speak** for themselves.

You have to add two more fields that will hold the coordinates of the center of the console. You can get them from the Position class that is located in UserInterface folder. They should look like this:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-08_14-06-39.png

### Helpers

Once again, we need to define the commands that our controller will have. They will be AddTitle, AddCategory, Write, and Post. Create the enum that is called Command:



### Properties

The properties that we need are a PostViewModel that we need to display the information from. Create it with private set, like this:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-08_14-14-48.png

Next, we need a text area as a property where we will get our text from. You could take a look at the implementation of the TextArea and add some functionality if you want. Anyways, your property should look like this:

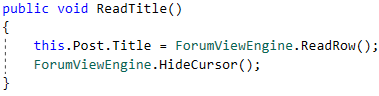
C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-08_14-17-46.png

The final thing that we going to need is boolean propety called Error**:**

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-08_14-19-16.png

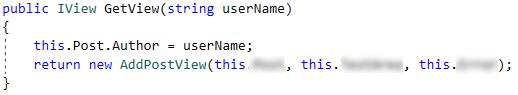
### Methods

Now, as I said earlier TextArea will be responsible for the content string, but the title and the category will be read like the **user data** in the previous document:

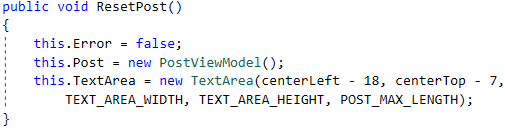


There, since you have the implementation of the ReadFile method use it to implement ReadCategory.

The next method is IController’s GetView and all it should do is set the **author** of the post and return new AddpostView giving it the Post, TextArea and Error as parameters:

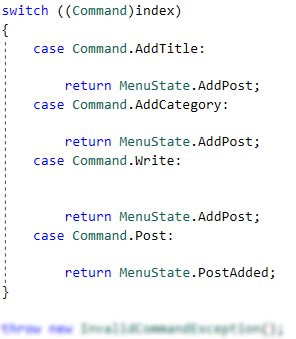


The last method we need is ResetPost and its name pretty much speaks for itself:



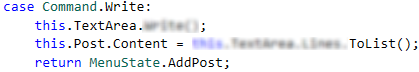
### Execute Command

I believe you’ve done enough times already. Create a switch on the casted parameter:



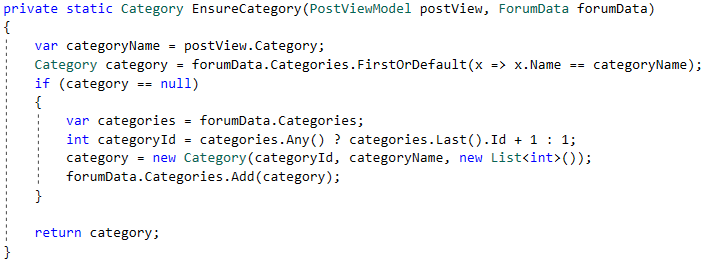
Now, AddTitle and AddCategory should call the corresponding methods you’ve just wrote.

The Write command should call TextArea’s Write method and the set the post content to TextArea’s lines something like:



The last command is Post and in order to implement that case we will have to make ourselves a service method called TrySavePost and takes PostViewModel as argument. Go to PostService class and add the following methods:

First we need to create a method that ensures that we have the category for the post we are trying to add:



Note that this method doesn’t use its **own** ForumData but takes one as an argument.

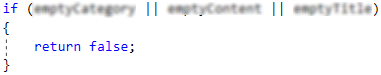
Now we proceed to adding posts. Define a method that looks like this.

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-08_15-01-33.png

In it crate **validations** for the **category**, **title** and the **content** represented as Booleans:

C:\Users\david\Desktop\devenv_2018-03-08_15-06-27.png

And if any of the variables is **true** your method should return false:



After we validate our ViewPostModel we should instantiate new ForumData and call the EnsureCategory method we’ve just created:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-08_15-18-49.png

After that we should generate a new **id** for the entity we’re about to create. You ca see how in the EnsureCategory method shown above.

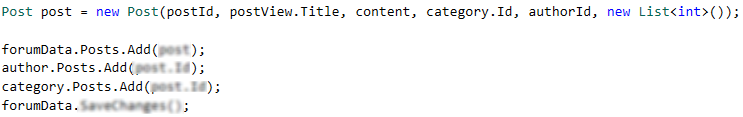
Next we fetch our post author calling the UserService GetUser method with PosViewMode.Author:

C:\Users\david\Desktop\devenv_2018-03-08_15-22-01.png

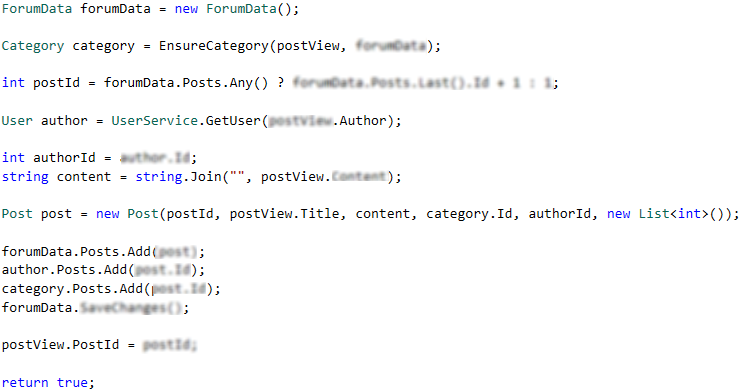
The next two steps are getting the author id and concatenating postView.Content into a single string:

C:\Users\david\Desktop\devenv_2018-03-08_15-24-36.png

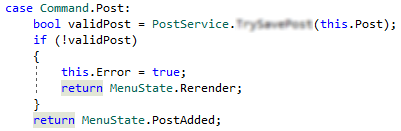
Finally create new Post model with the data you have and add it to forumData’s posts, the current author posts ids and current category posts ids and don’t forget to SaveChanges after you’re done:



All that is left to do is set postView.PostId to the curretn post id and return true in the end. All that should look like this:



Now we are ready to go back to AddPostController’s Execute command and add the following in Post command case:

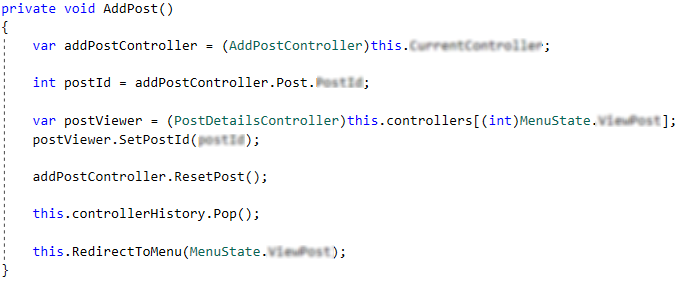


### Constructor

Create and empty constructor that calls this.ResetPost method.

### Menu Controller’s Add Post method

The last thing we need to implement in order for our forum to create posts is implement AddPost method in MenuController. All it should do is fetch current post **id** set, give it to PostDetails Controller, resets the AddPostController and Redirect method, like this:



By now your forum should can add posts. Having all this implement AddReplyController. The service methods you will need are called TryAddReply (similar to TryAddPost). Good luck!