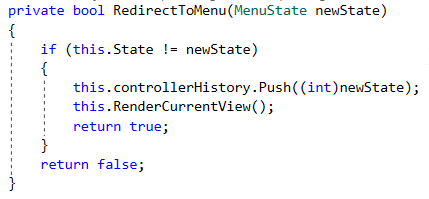
# Workshop: Forum

## Overview

In previous part we’ve implemented the “database”, now it’s time for **login** and **signup** functionality.

## Define Redirect to Menu

In Menu controller class, in Execute command’s switch, all of the states that require special logic are listed as cases if the current state is none of those cases, it ends up in the default case: RedirectToMenu method, taking MenuState as argument. The logic you need to implement there is simple. Check if the incoming state is different from the current one, if it is, just push the newState into controllerHistory, call RenderView and return true. Otherwise, take no action and return false. It should look pretty much like this.



With this method our menu changes its current state and then call the RenderCurrentView. If you take a close look you will find that RenderCurrentView does nothing but asking the current controller for a view, setting it to be CurrentView and calling it’s own forumViewer’s RenderView method with the CurrentView.

Now that we have a method that wants stuff form our controllers, we shold step to **implementing controllers**.

## Implementing LogInController

In the class LogInController all you have for now is the implementation of IController and IReadUserController interfaces and all of the methods throw NotImplementedException.

### Properties

One of the properties that needs to be defined already exists and all you have to change is give it a **public** **getter** and a **private** **setter**. Like this:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-06_20-59-11.png

Next, we should add a private string Password:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-06_21-01-57.png

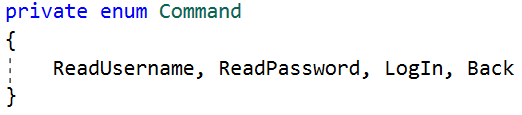
The last property that the controller needs for its functionality is the private bool Error:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-06_21-00-42.png

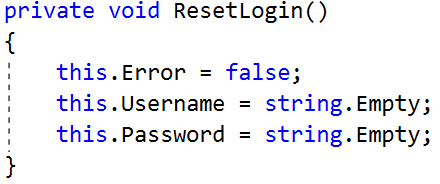
Note that Password and Error are not part of the interface, that’s because they are needed only for **inner usage**.

### Helpers

After we made the properties we need, we are going to implement some other stuff that we need. In this case this is an enum (your favorite) which is private, **called** Command and holds the indices of the controller’s **commands**. It should look a lot like this:

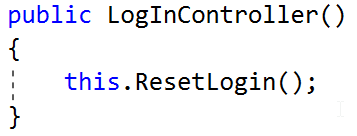


The next helper method is ResetLogin and it speaks for itself. It sets Error to false, Username and Password to string.Empty so they are not null… Simple as that.



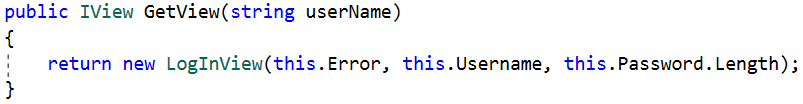
### Constructor

The constructor of this controller just calls the ResetLogin method we’ve just written, like so:



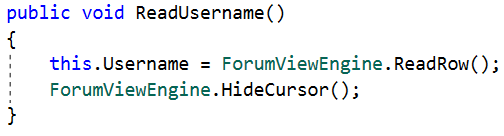
### Implement Interface methods

The first method we are going to implement is the GetView method that is called by MainController’s RenderCurrentView. As you can tell by its name the method **returns view** and the implementation is straightforward.



Note: Include the namespace in the using, so you can see the views.

Next in line are ReadUsername and ReadPassword and they are pretty much the same. They read a row for the corresponding property and call ForumViewEngine’s HideCursor method.

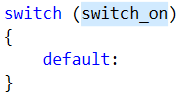
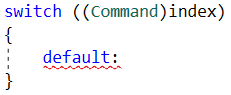


Having the implementation of ReadUsername, write the logic of ReadPassword. By this time, you should be able to go to LogIn and the controller should **return a view** for the ForumViewEngine to **render**. Try it out.

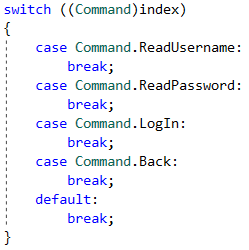
### ExecuteCommand

This is the method, every controller has. It determines, which of the commands should execute and calling the corresponding method for it. It should contain switch case with all of the commands in the Command **enum**. The easiest way of implementing it is:

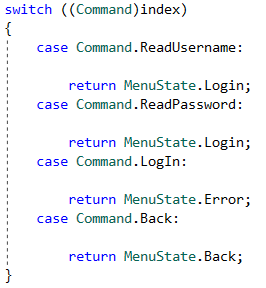
Define a switch statement: Cast the parameter to Command:

And then press enter twice. And… voilà!

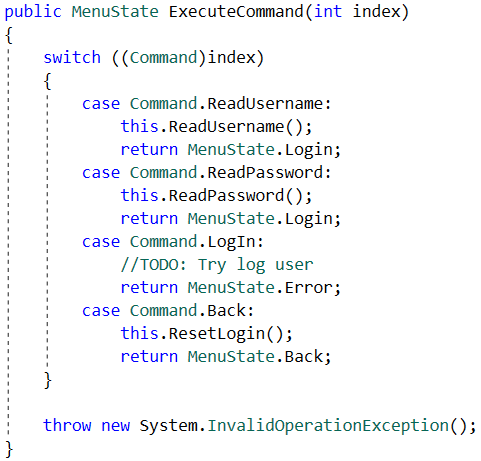


The next step is to **delete the default case** because we are not going to need it. Then, you should replace all the break statements with return "MenuState.SomeState", because the return type of the method is MenuState. First two cases (ReadUsername and ReadPassword) are going to return MenuState.Login in order for the user to **stay** on the same view. Login command case is going to return MenuState.Error, just for now. And finally Command.Back should return MenuState.Back. We are about to implement some logic in the cases in just a few moments, but before that check if your switch looks like this:



All that’s left to do is to **call the methods** we have already implemented that **execute the commands**. The ReadUsername command will call the ReadUsername method. I think you can guess what ReadPassword should call. Finally, the back command should **reset the login**. One last thing to do here for now is to throw an exception if the command doesn’t match any of the cases. (Ideally, you should implement your own custom exception, but the built-in InvalidOperationException works as well).

Finally, your method should look like this:



This is it for now, we will be back to finish that **TODO** later.

## Implementing SingnUpController

The SignUpCntroller is pretty similar to the LogInController so let’s get right to it.

### Fields

The fields we are going to are two private const strings holding the **error messages** of the view. The f one is called DETAILS\_ERROR and looks like this: C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_00-07-13.png

And the other one is called USERNAME\_TAKEN\_ERROR:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_00-07-26.png

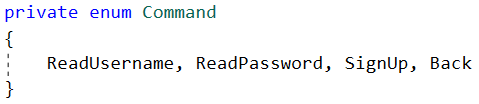
### Properties

The Username and Password properties are **exactly the same** as in the LoginController. The thing that is new is the private string ErrorMessage looking like this:

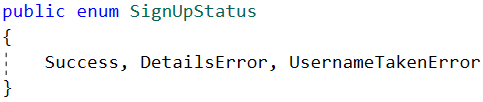
C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_00-12-52.png

### Helpers

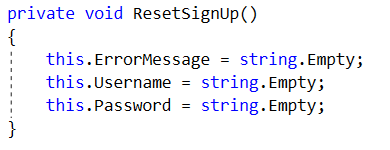
The first helper we will implement is the Command enum that every view has. This time, however, instead of a LogIn command it will have SignUp instead:



This time we are going to define a second enum holding SignUpStatus which will tell our controller if signing up our user is successful and if not, **why**.

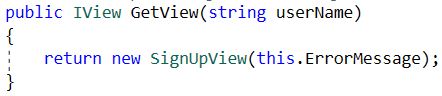


As you might have guess the ResetLogin method should look like this.



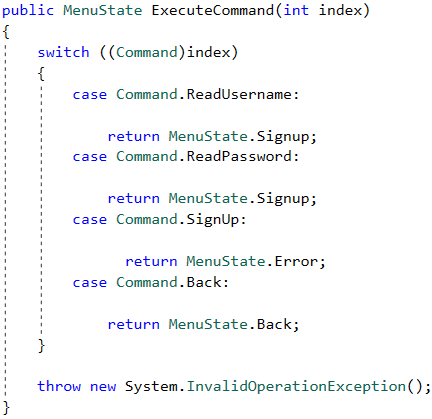
### Implement Interface Methods

ReadUsername and ReadPassword are the same as the ones in previous controller (surprise), GetView method looks like this:

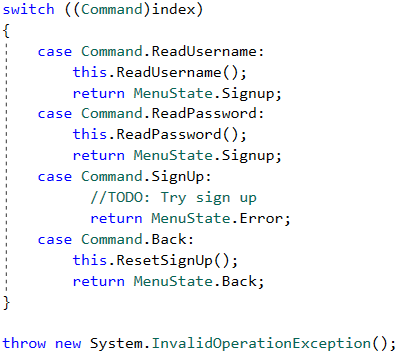


### ExecuteCommand

Since you are aware of the **witchcraft** from the previous part, I suppose you will be able to get to this state:



Again, everything is the same: Each command calls its corresponding method except the SignUp command. (We’ll get to that in seconds, really...) Final look before proceeding to the service:

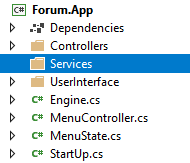


## Implementing UserService

In this part we will implement UserService class who will serve both controllers we’ve just implemented. According to Wikipedia service is something like *“…****software****functionality or a set of software functionalities (such as the retrieval of specified information or the execution of a set of operations) with a purpose that different****clients****can* ***reuse*** *for* ***different purposes****…*”. We’ll dive into that in the next course, but all you need to know for now is that the functionality of operating with the “database” **shouldn’t** take place in your **business logic**. That’s why you create services. ☺

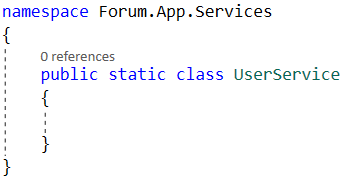
### Create Services folder

In the main directory of the project create new folder called **Services**.



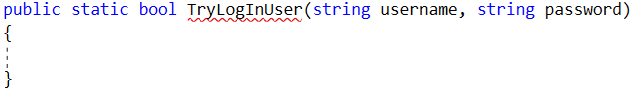
### Create UserService

In services directory create public static class called UserService

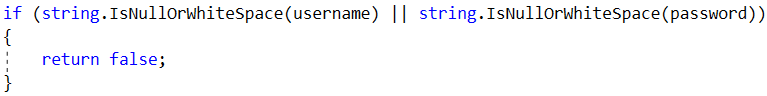


### Implement TryLogInUser

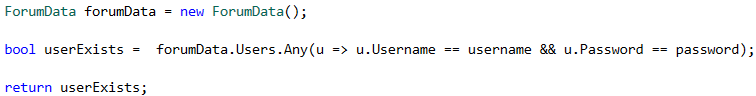
First of all, create a public static TryLogInUser method of type bool with username and password as parameters.



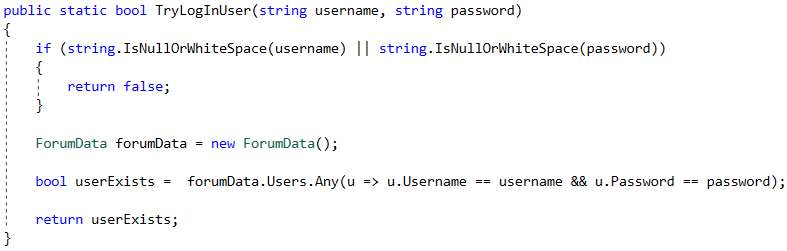
Then, check if the given username or password is null or empty and return false if any of them are.



After the check instantiate an object of type ForumData and check if there is any user with the same username and same password. Return the result.

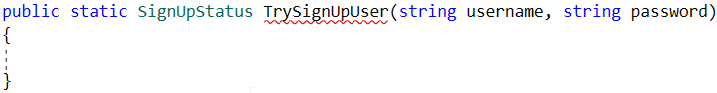


The whole method should look like this:



### Implement TrySignUpUser

Since the possible endings of signing up a user are more than 2, we are going to user the SignUpStatus enumeration you’ve created earlier. (You remember it, right?) We are going to define public static method of type SignUpStatus taking username and password as parameters.



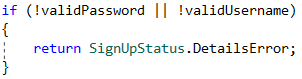
The validations you need to perform here are if the username and password are not null or empty and if they meet the requirements for **minimum length that is 3**. To do that define a bool variable that checks those two conditions for username:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_02-26-00.png

And then repeat it for password:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_02-26-18.png

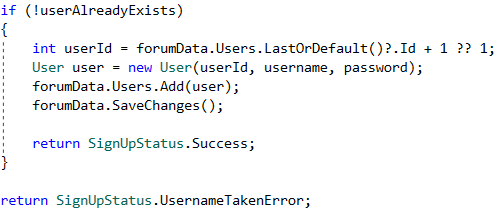
After that if one of the defined variables is false return “DetailsError” SingUpStatus:



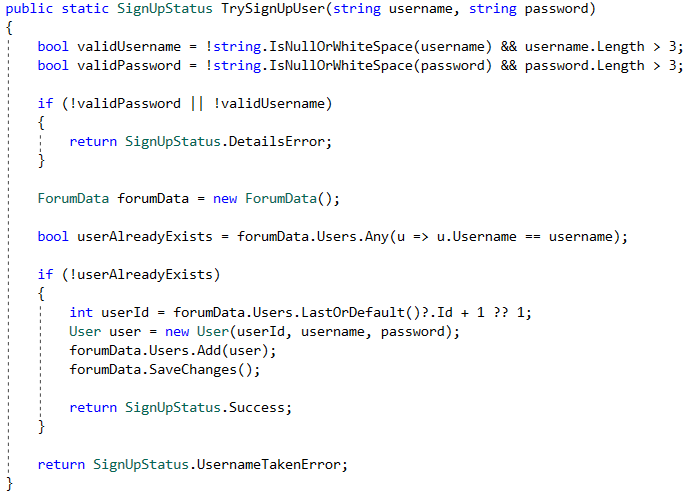
Next you need to instantiate object of type ForumData and in another variable determine if there is user with the username that is given as a parameter:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_02-31-37.png

If such user doesn’t exists in the “database” you need to **generate** an id for the new user by taking the **last user id and incrementing it with one** or **setting it to one if there are no existing users**. Then you create a User model with the data you have, **add the user to the base**, **save changes** and return Success status. If the user exists return UsernameTakenError status.



The entire method should look like this:



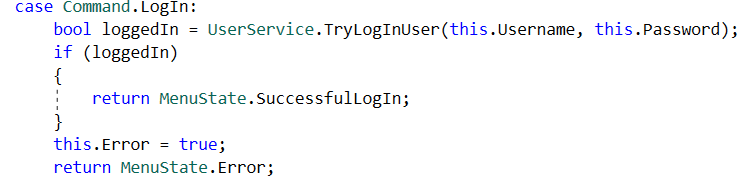
## Finish the Implementation of User Controllers

### LoginController

In ExecuteCommand‘s LogIn case define a Boolean called loggedIn (for example) and assign it with the result of UserServise’s TryLogInUser method with parameters Username and Password properties:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_02-47-35.png

If logged in is true return SuccessfulLogIn state otherwise set Error property to true and keep the return of Error state.



### SignUpController

SingUp case here is pretty much the same. We call the UserService TrySignUpUser with Username and Password and assign the result in a variable:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_02-55-40.png

Next we switch on signUp and return SuccesfulLogIn state in case of Success Status:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_02-58-28.png

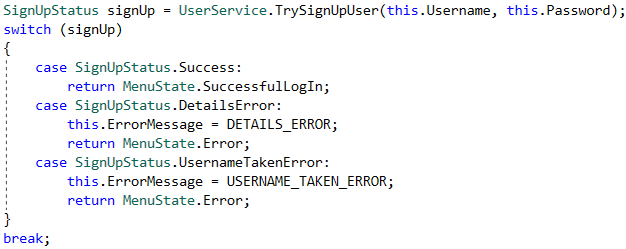
Setting ErrorMessage to DETAILS\_ERROR constant in case of DetailsError status:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_02-58-49.png

And USERNAME\_TAKEN\_ERROR in case of UsernameTakenError status:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_03-00-23.png

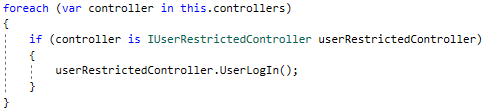
All of it looks like this:



All that is left for us to do is implementing LogInUser, LogOutUser, SuccessfulLogIn and LogOut in MenuController.

### LogInUser

Here you just go through all of the controllers and calls the UserLogIn method if the controller implements IUserRestrictedController interface, like this:



### LogOutUser

This method is the same as LogInUser but calling UserLogOut instead.

### SuccessfulLogIn

All you have to do here is cast the CurrentController so you can get the username from it and set it to MenuController’s Username:

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_11-26-49.png

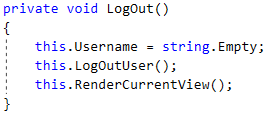
Next you call the LogIn method you’ve just implemented and the RedirectToMenu with the Main MenuState as a parameter.

C:\Users\david\Documents\ShareX\Screenshots\2018-03\devenv_2018-03-07_11-29-19.png

### LogOut

The last piece of code you need to write in this method does the following:

Setting Username to an empty string. Call the LogOutUser method and then renders the current view, like so:



That’s it! If you have done everything right you should be able to signup users and login with them. See you in the next part where we will implement posts and replies functionality. ☺