Table of Contents

[Section 15: User Management 2](#_Toc145815326)

[15.1 Display List of User 2](#_Toc145815327)

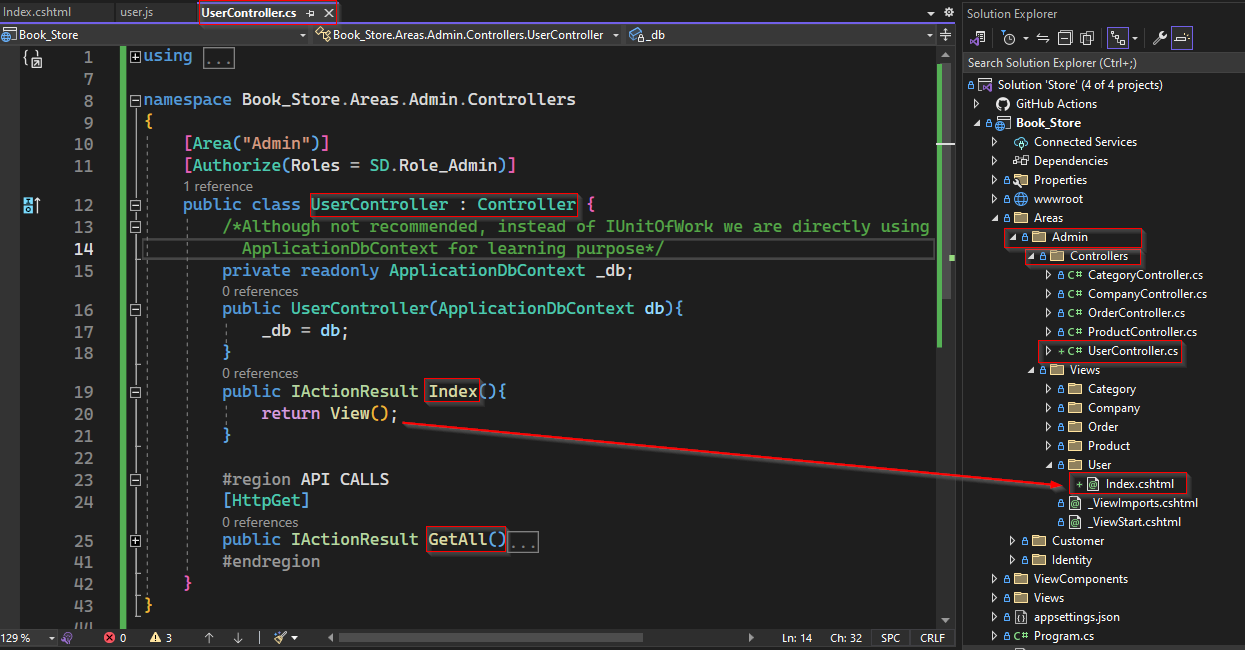
[15.2 Lock or Unlock User 7](#_Toc145815328)

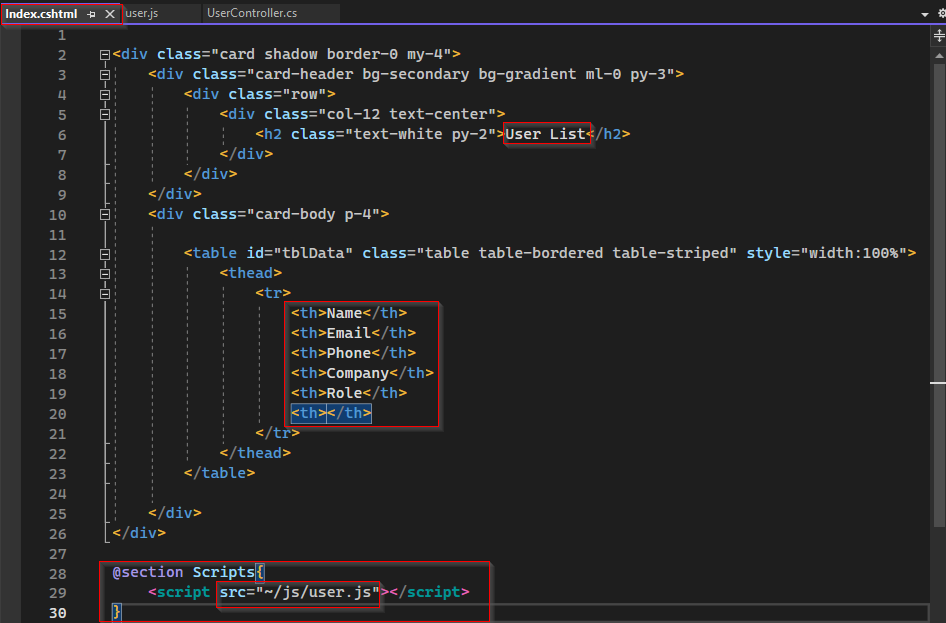
[15.3 Manage User Roles 9](#_Toc145815329)

# Section 15: User Management

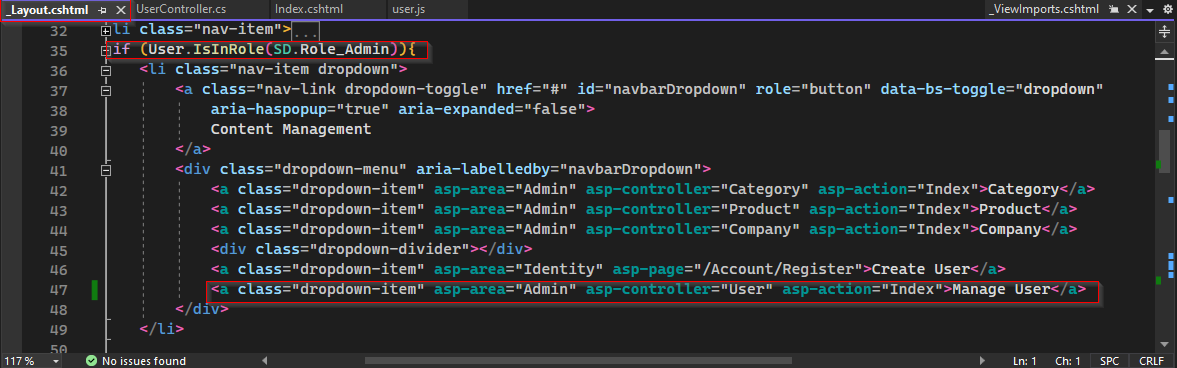
In this section we are going to add CRUD functionality for application users authorized to admin role to manage users.

## 15.1 Display List of User

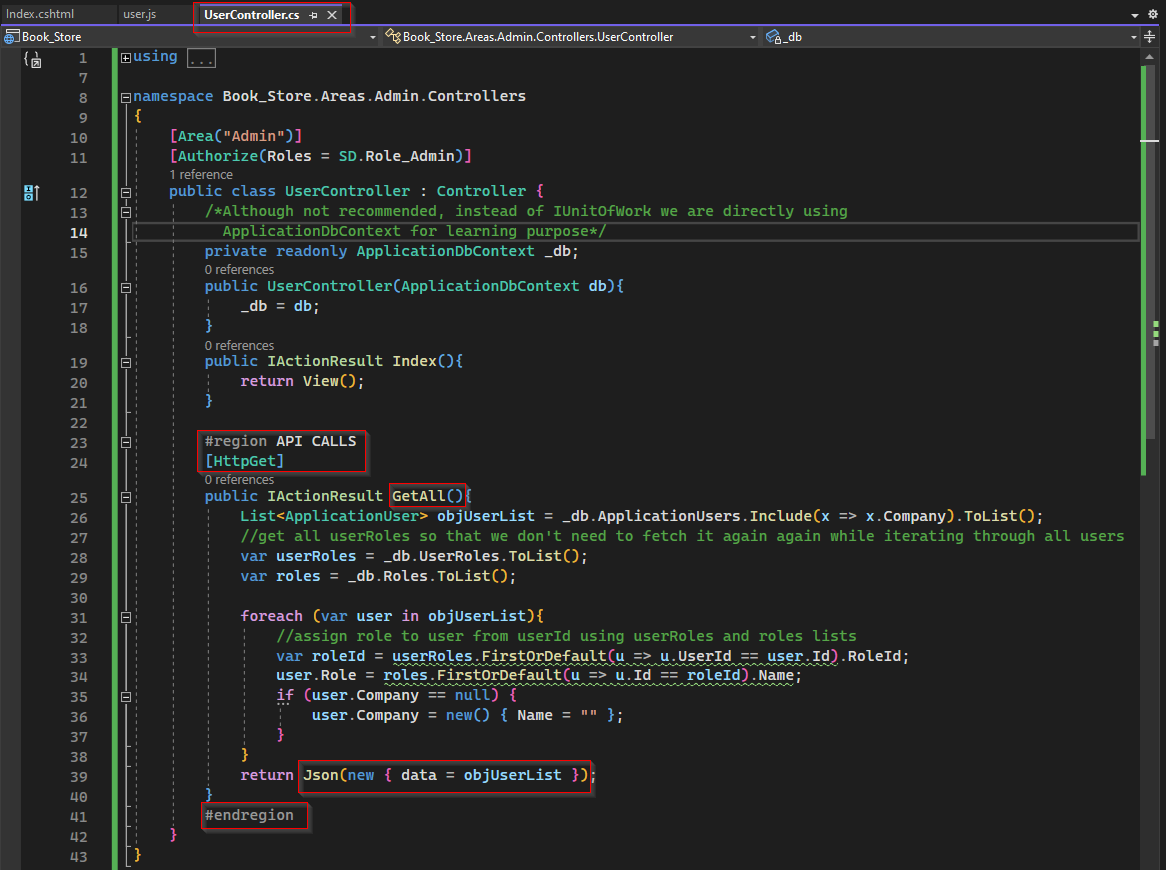
Since we already have model ApplicationUser now we need to create UserController inside "Area/Admin/Controller".Here, although it is not recommended for learning purpose lets injecting ApplicaitonDbCondext instead of IUnitOfWork.

Here we have an action method Index that simply returning a view with a table of user’s some details. But the data to be populated is to be provided by API call (GetAll method) using user.js.

Next we also need to navigation button on nav-bar under Content Management for Admin role as follows,

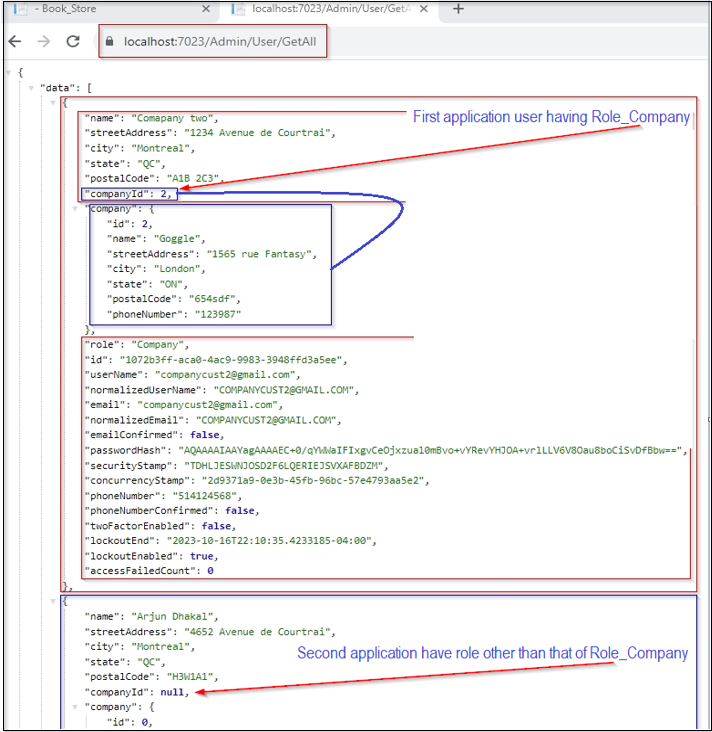


Now logic and code used for Action method GetAll is as follows:



In action method GetAll,

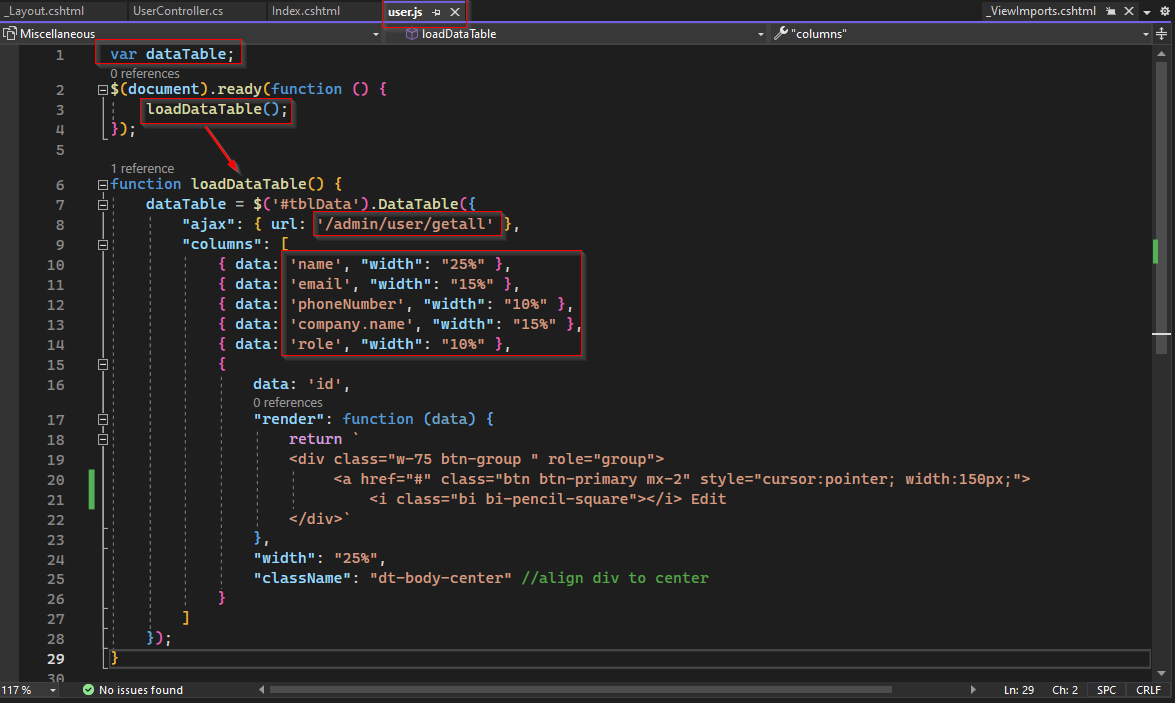
* we first get all the list of all application users (objUserList) from ApplicationUsers which also including Company detail is any application user has role of company.
* Next, we also get complete lists of user’s roles (userRoles) from UserRoles table and roles (roles) from Roles Table so that we don’t need to access these tables in database again and again for each application user.
* Next, we loop through each user in objUserList to get roleId and using roleId we get role name for each user.
* Finally, we all data need to be populated and we return a Json result as follows.



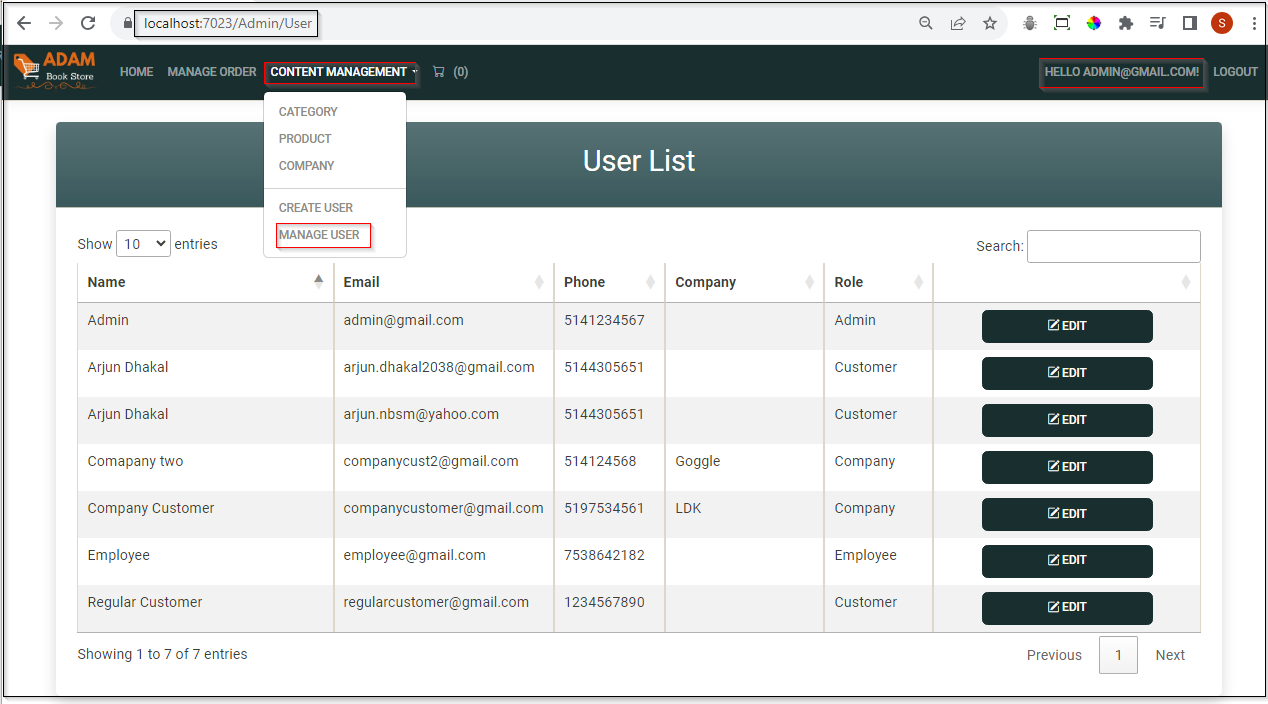
Base on the table in Index.cshtml, we need the value for following keys that need to populate to table body using user.js.

* "name"
* "email"
* "phoneNumber"
* "company"
* "company.name"
* "role"

Now we need to create a Javascript file user.js which basically use dataTable and uses the ajax to call the API GetAll from User Controller to populate the table body in view index.cshtml. The code used on user.js is as follows.



To view the list of users, first one need to sign with Admin credential, then click on Manage User on dropdown list of Content Management.



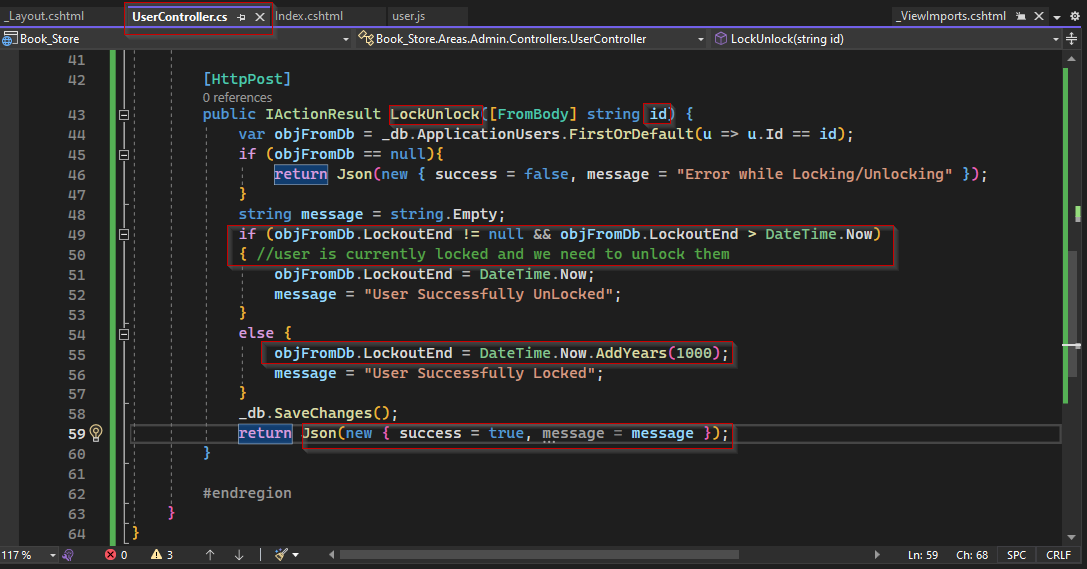
## 15.2 Lock or Unlock User

Next feature we want to have on user list, admin have some kind of access to lock or unlock to any user account to perform any business.

On implementing ASP.NET MVC Core Identity, AspNetUsers (simply Users) table by default has columns with name "LockoutEnd" and "LockoutEnabled". Column "LockoutEnabled" can accept Boolean and has default value true or enabled. While "LockoutEnd" can accept data of type DateTime.

ASP.NET Core Identity has inbuild feature that if the date time value on "LockoutEnd" is greater than current date time and "LockoutEnabled" is true for any user then that user will not be able to login or perform any transaction.

Hence to achieve lock or unlock feature, we simply need to create a button that the toggle the DateTime value for field "LockoutEnd" can between current time and some certain future time say current DateTime + 1000 years. Hence that account will be locked until that date.



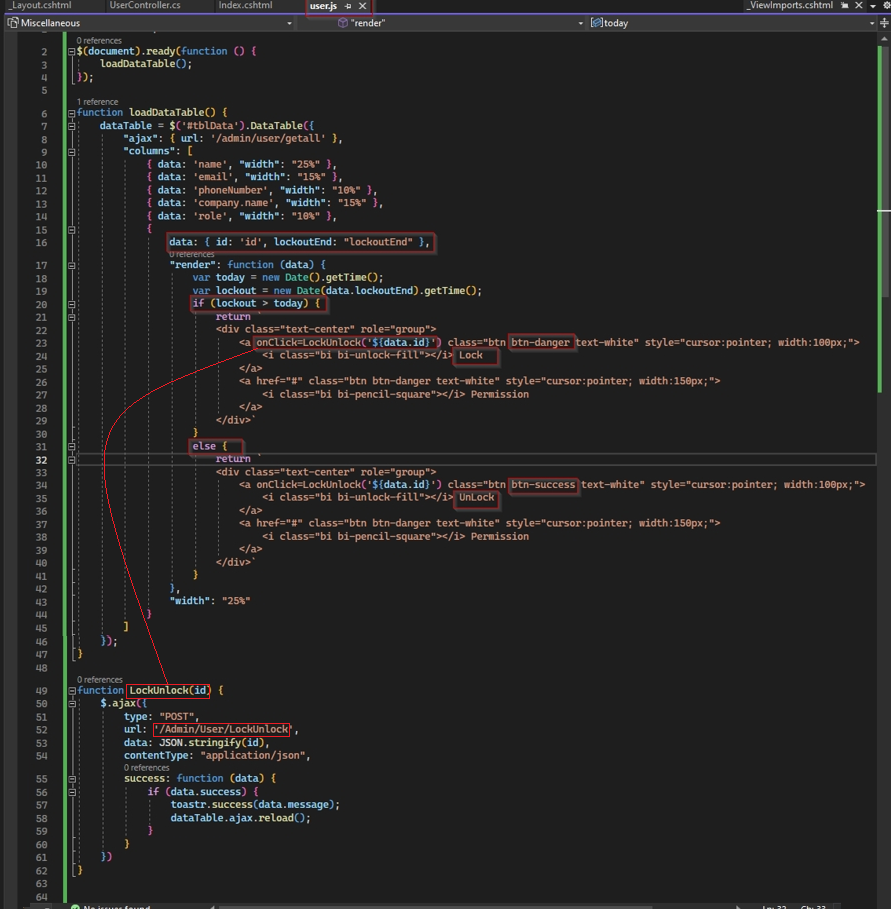
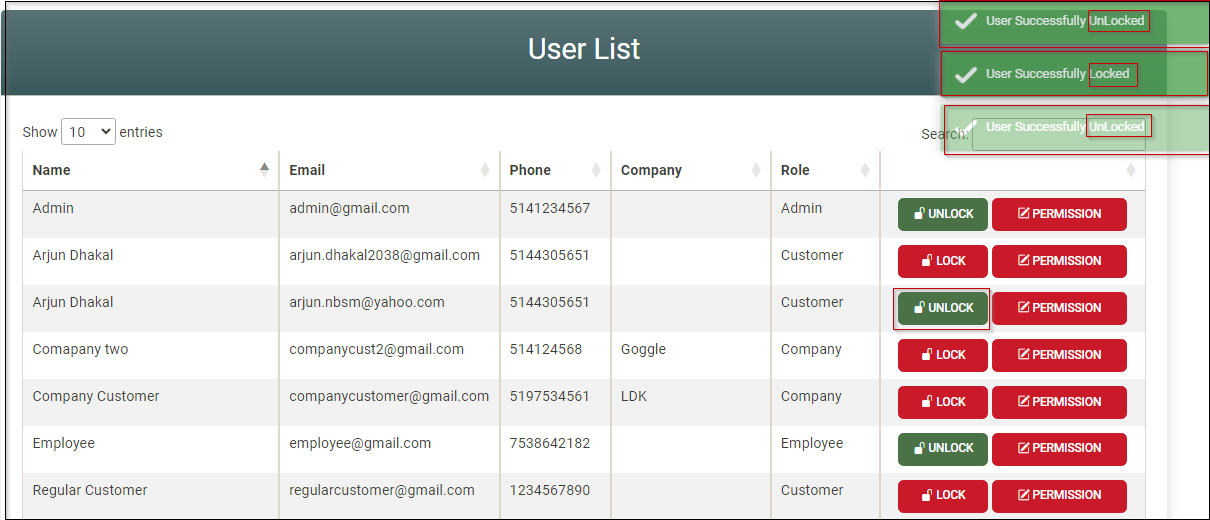
As describe above, on this snippet of code, the logic used is,

* First find application user using id
* Check value of LockoutEnd column
* Every time this method is called, it will toggle the value of LockoutEnd field between current date time and current date time plus 1000 years.

Note:

* Since by default, LockoutEnabled is true, we simply ignore in above logic.
* While supplying parameter to LockUnlock method we have used [FromBody] annotation. The [FromBody] attribute which inherits ParameterBindingAttribute class is used to populate a parameter and its properties from the body of an HTTP request. Similarly in some other case we can use [FromUri] which means the value should be read from the URI of the request.

To consume above action method, we need to make some modification on view index.cshtml as shown below.

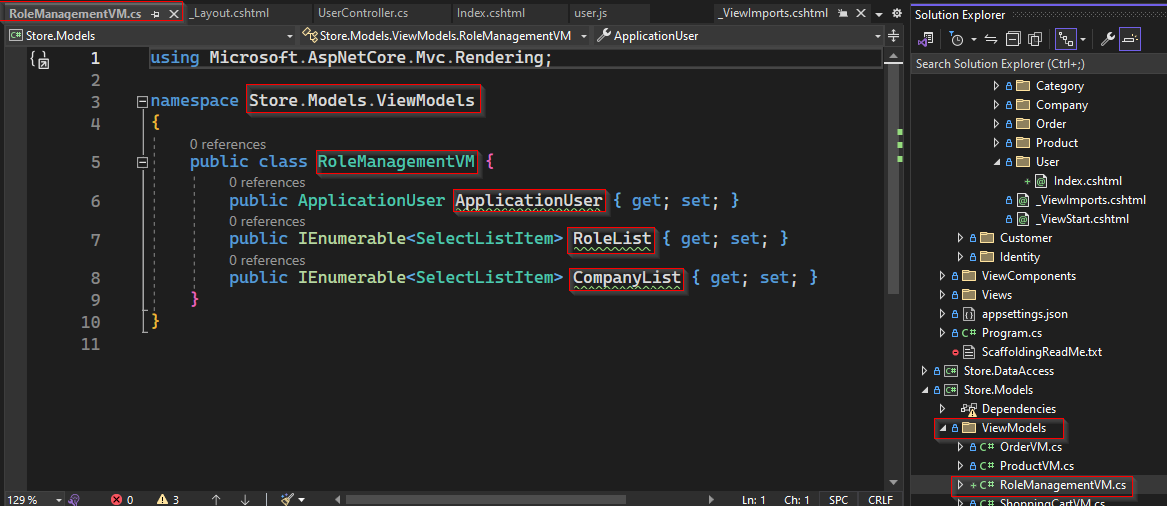
## 15.3 Manage User Roles

Here we want to give admin to authority that he can change role of any user. For this we are going to add action method with name of RoleManagement that can expect user Id. Base on user Id it will populate a form or view with three inputs for

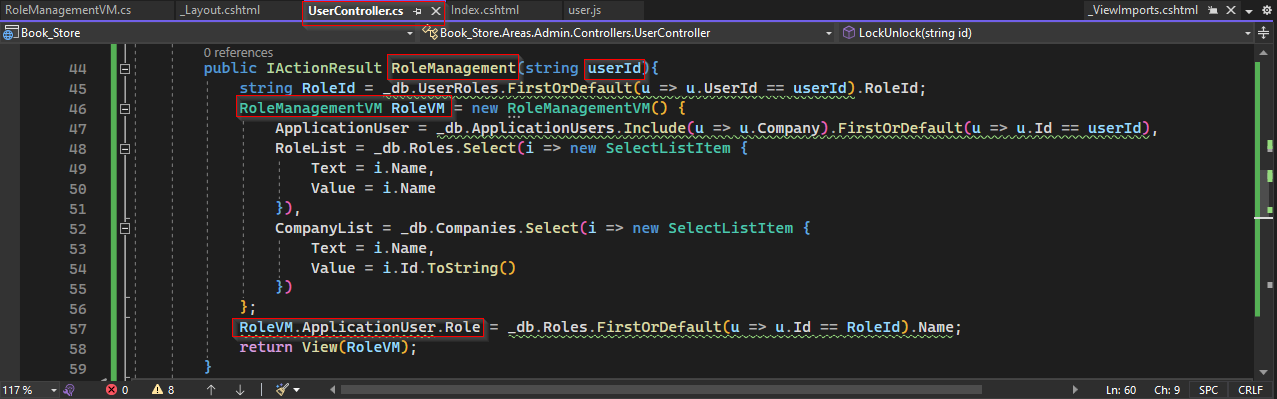
* Name,
* Current Role
* Company Name (if user’s current role is “Company”)

Beside this role and company field should be select dropdowns to enable Admin to select and update role and company when applicable. Company name field should appear is user has role of Company and vice versa.

Now to create a view with input for name, dropdown for roles and company, we first need to have view model with properties applicationUser, lists of roles and companies. Let give the name of view model is RoleManagementVM.



Get Method to populate view input and dropdown list.

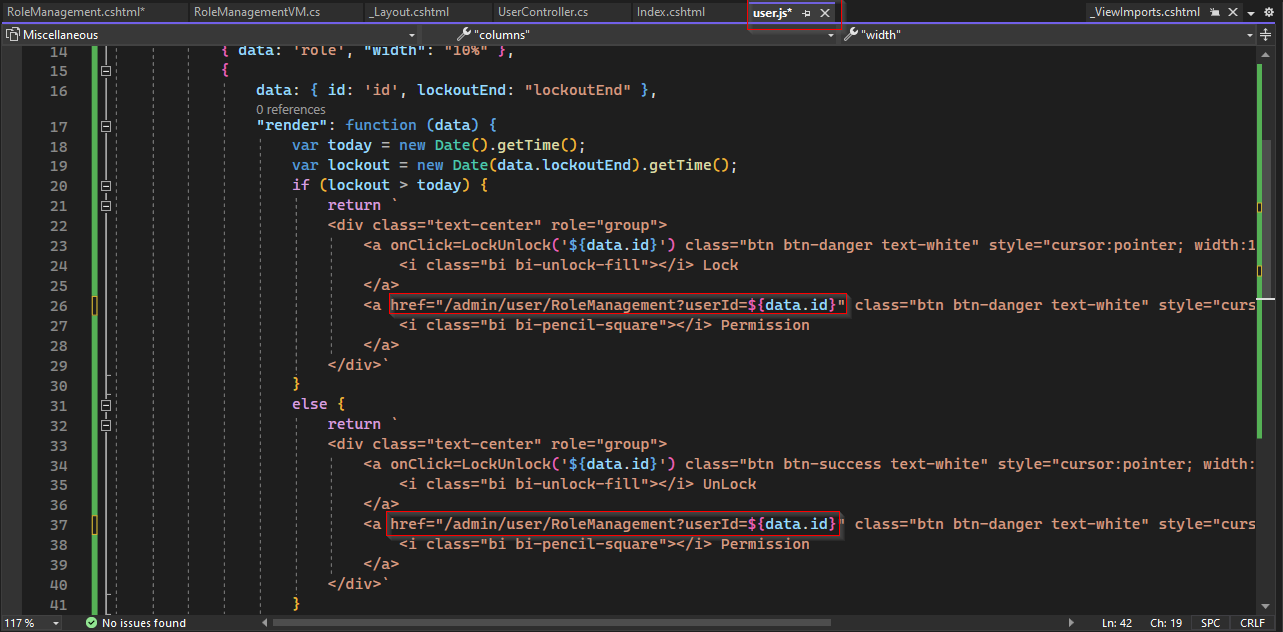


On above snippet of code, logic used are:

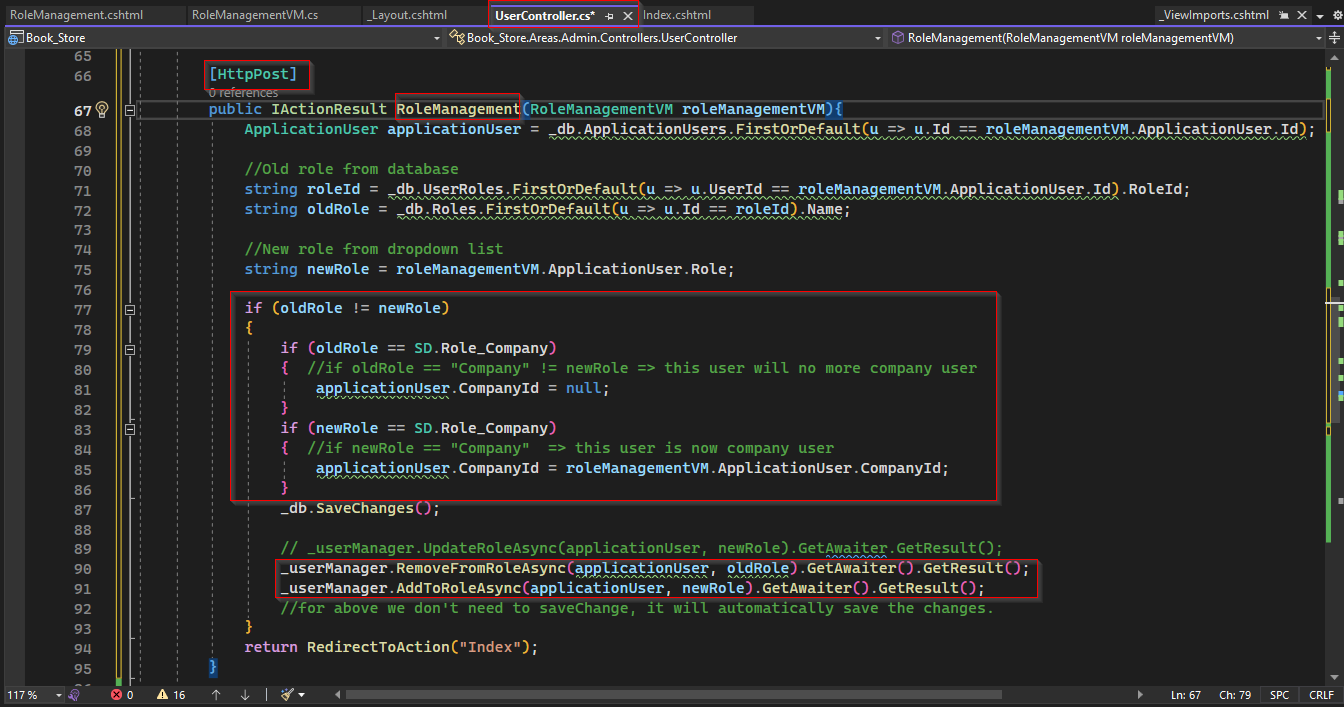
* First from user Id passed to method, we find RoleId from table UserRoles
* Create a RoleManagementVM with following properties and pass it to view as model.

1. Application user using user Id from ApplicationUsers Table
2. List of all roles (required to create dropdown list of roles
3. List of all companies (required to create drop down of companies

Next, we need to add action method to "Permission " button in table of user list as follows.



Post method to update role:



Now to create view for RoleManagement, we can copy content of existing view say create product for example and then modify as per your need. Snippit of code for RoleManagemetn is as follows.

