User Helper Classes and Methods for Role Management

For certain repetitive functions, particularly those that involve the creation of complex objects such as role and user managers, it is often beneficial to create Helper classes to simplify the implementation of such functions. We might consider, for example, creating a helper class to assist us in managing the assignment/de-assignment of user roles, determining whether a user is in a particular role, or getting a list of roles to which the user is assigned.

It should be noted that building a Helper class to perform these functions is certainly not mandatory. It is simply *one* way of completing these tasks. The code for such Helper classes could very easily be integrated into your application’s controller actions.

Building a User-Roles Helper Class

1) You can construct a Helper class anywhere in your project. It might be placed in either the Models or Controllers folder, or you might construct a separate Helpers folder, particularly if you expect to create multiple Helper classes for your project.

2) Once the class is defined, you must set up a UserManager object through which you must obtain all user-role information.

3) Within the UserManager object, methods exist to help you a) determine whether a user is in a particular role, b) add a user to a role, and c) remove a user from a role. While these methods already exist in the UserManager, some developers prefer to include wrappers for these methods in their own Helper classes.

4) It is also possible to construct custom methods to obtain information that is not directly accessible from the UserManager, such as a list of users not currently in a particular role, or a list of users assigned to a given role.

5) Build it.

```
using System.Linq;
using System.Data.Entity;
using System.Collections.Generic;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.EntityFramework;

namespace MyApp.Models
{
    public class UserRolesHelper
    {
        private UserManager<ApplicationUser> userManager;
        private ApplicationDbContext context;

        public UserRolesHelper(ApplicationDbContext db)
        {
            context = db;
            userManager = new UserManager<ApplicationUser>(new
        UserStore<ApplicationUser>(new ApplicationDbContext()));

        }

        public bool IsUserInRole(string userId, string roleName)
         {
             return userManager.IsInRole(userId, roleName);
         }
         public ICollection<string> ListUserRoles(string userId)
         {
             return userManager.GetRoles(userId);
         }
         public bool AddUserToRole(string userId, string roleName)
         {
             var result = userManager.AddToRole(userId, roleName);
             return result.Succeeded;
         }
         public bool RemoveUserFromRole(string userId, string roleName)
         {
             var result = userManager.RemoveFromRole(userId, roleName);
             return result.Succeeded;
         }

         public ICollection<ApplicationUser> UsersInRole(string roleName)
         {
             var resultList = new List<ApplicationUser>();
             var List = userManager.Users.ToList();
             foreach (var user in List)
             {
                 if (IsUserInRole(user.Id, roleName))
                     resultList.Add(user);
             }

             return resultList;
         }

         public ICollection<ApplicationUser> UsersNotInRole(string roleName)
         {
             var resultList = new List<ApplicationUser>();
             var List = userManager.Users.ToList();
             foreach (var user in List)
             {
                 if (!IsUserInRole(user.Id, roleName))
                     resultList.Add(user);
             }

             return resultList;
         }
    }
}
```

Building Other Helper Classes

The processes described in the previous section can be repeated to create any number of Helper classes to perform additional repetitive operations. For example, in your BugTracker application, you must create and manage a Projects table in your database. Just as you must assign and unassign users to and from Roles, you must also assign and unassign users to and from Projects. The structure of these relationships is virtually identical, and therefore the processes are likewise very similar. The only difference is that there is no mechanism in the UserManager class to deal with Projects in the BugTracker. You must build that yourself.

Consider your solution in step 5 above. Use the same thought processes you did here to devise a way to assign and unassign users to and from Projects in the BugTracker.