Project work 4

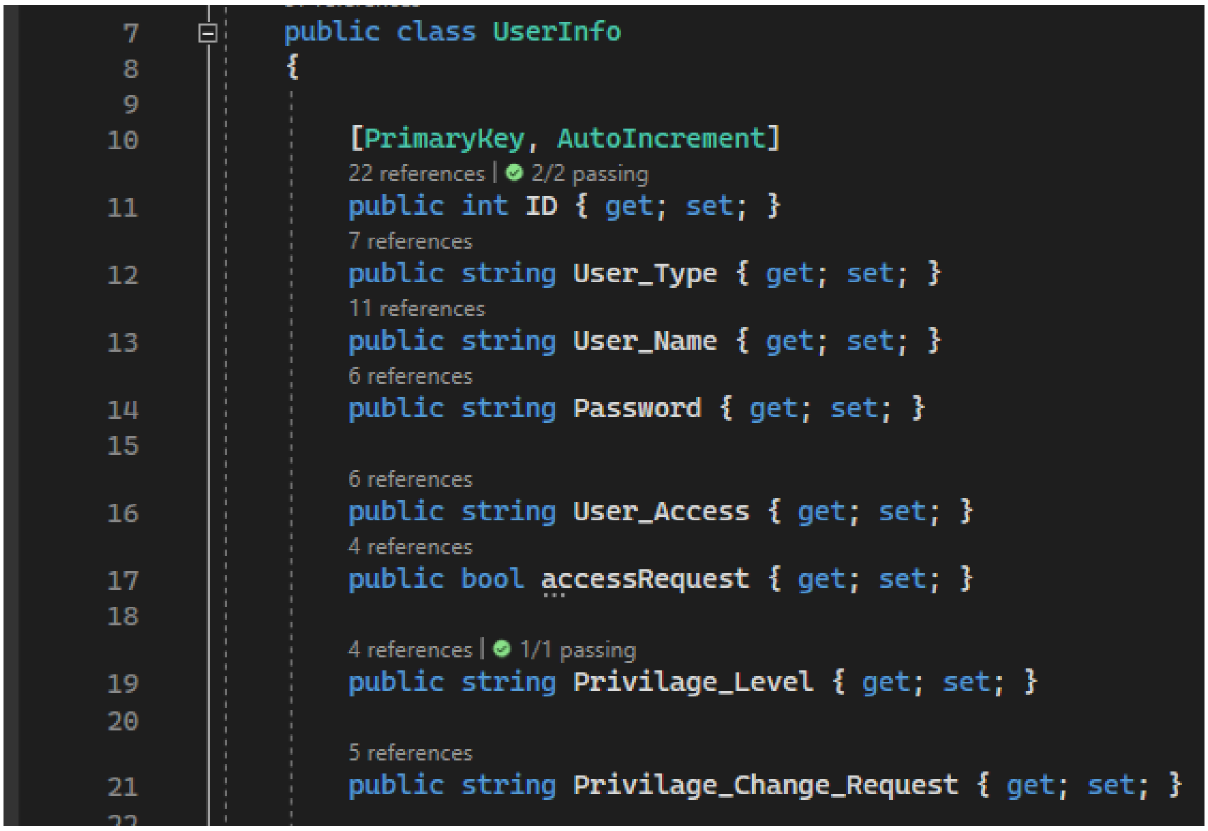
Breif

The task worked on this week is :

```
"As an UNDAC Team Support and Logistics Manager, I want to request privilege
changes for system users so that effective security is maintained

End user goal: To control access to mission systems

End business goal: To ensure the data security of the mission

Acceptance criteria:

Details of a team member's current system access privileges can be viewed
Requests for lower privilege levels are automatically approved
Requests for higher privileges must be approved by the Deputy Team Leader"
```

To achieve this I have modified the following classes:

UserInfo



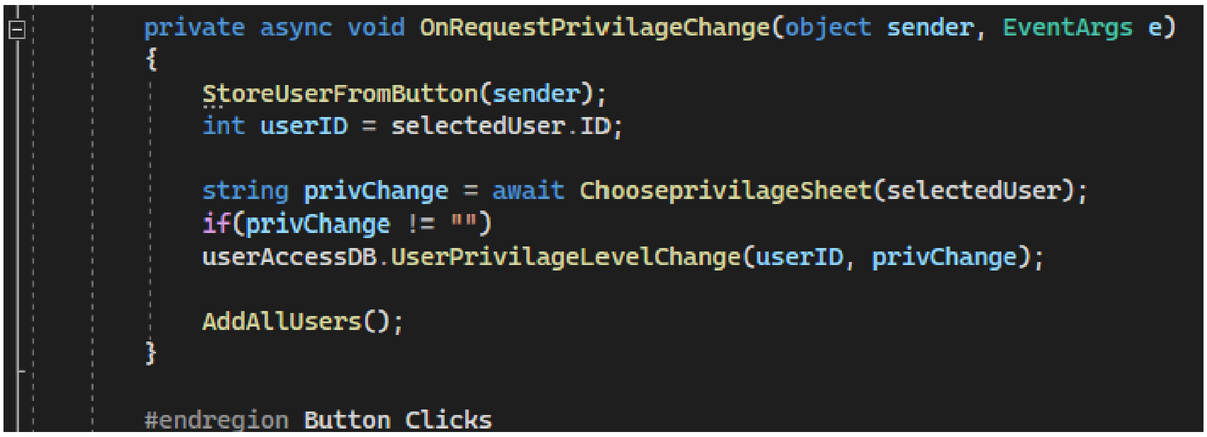
```
public class UserInfo
{

    [PrimaryKey, AutoIncrement]
    public int ID { get; set; }
    public string User_Type { get; set; }
    public string User_Name { get; set; }
    public string Password { get; set; }

    public string User_Access { get; set; }
    public bool accessRequest { get; set; }

    public string Privilage_Level { get; set; }

    public string Privilage_Change_Request { get; set; }
```

Fig.1 User Info Table

- Database table additions:

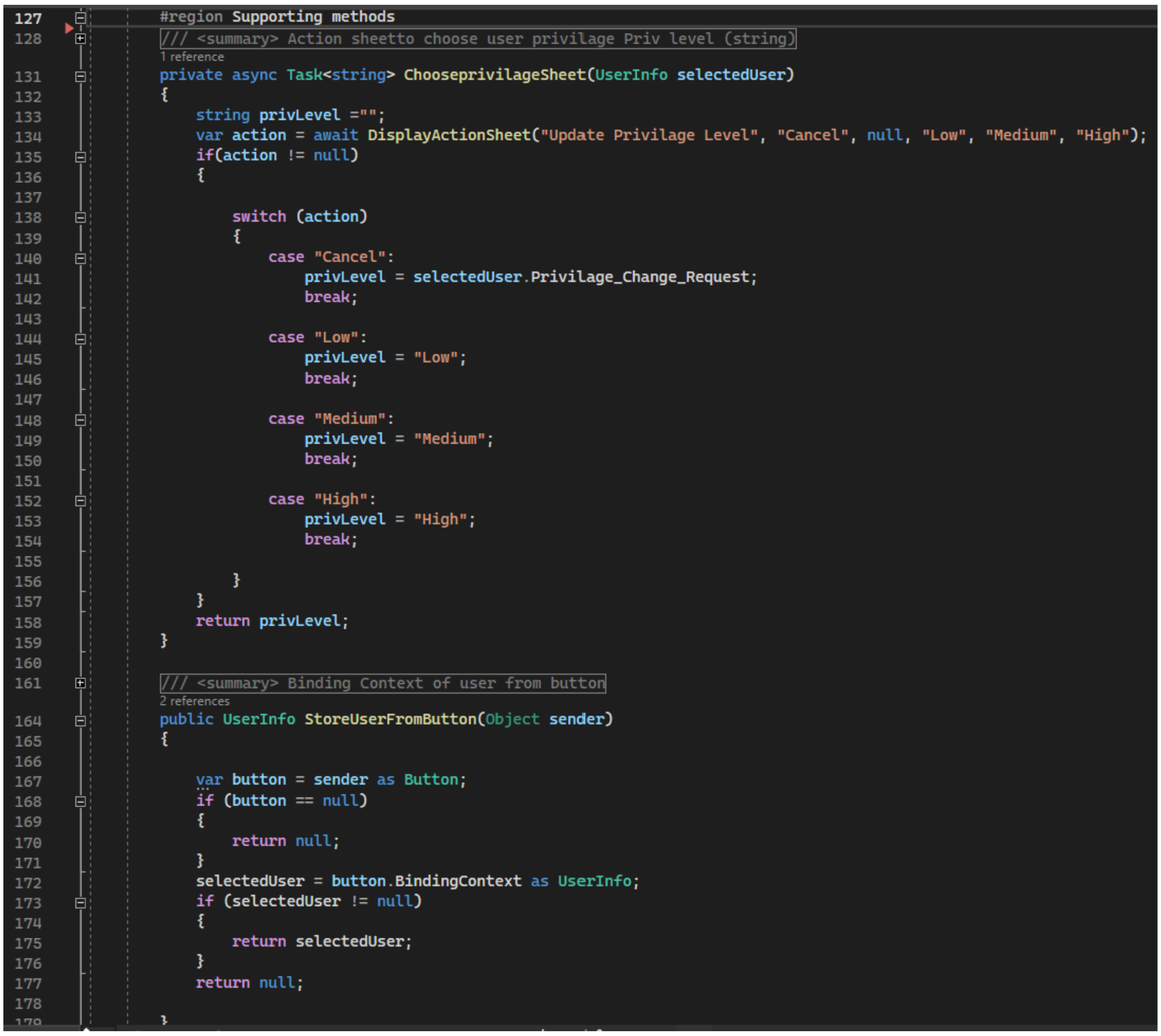
  - added field to store privilage level. (Line 19)
  - Added field store Privilage change requests.(Line 20)

UserAccessPage



```
private async void OnRequestPrivilageChange(object sender, EventArgs e)
{
    StoreUserFromButton(sender);
    int userID = selectedUser.ID;

    string privChange = await ChooseprivilageSheet(selectedUser);
    if(privChange != "")
    userAccessDB.UserPrivilageLevelChange(userID, privChange);

    AddAllUsers();
}

#endregion Button Clicks
```

Fig.2 UserAccessPage Request privilage changes



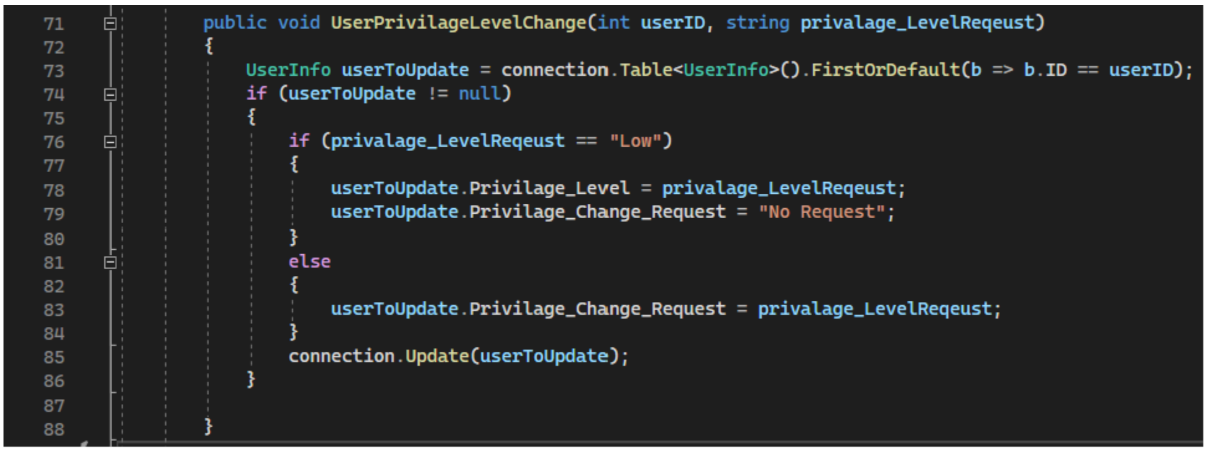
```
127     #region Supporting methods
128     /// <summary> Action sheetto choose user privilage Priv level (string)
        1 reference
131     private async Task<string> ChooseprivilageSheet(UserInfo selectedUser)
132     {
133         string privLevel ="";
134         var action = await DisplayActionSheet("Update Privilage Level", "Cancel", null, "Low", "Medium", "High");
135         if(action != null)
136         {
137 
138             switch (action)
139             {
140                 case "Cancel":
141                     privLevel = selectedUser.Privilage_Change_Request;
142                     break;
143 
144                 case "Low":
145                     privLevel = "Low";
146                     break;
147 
148                 case "Medium":
149                     privLevel = "Medium";
150                     break;
151 
152                 case "High":
153                     privLevel = "High";
154                     break;
155 
156             }
157         }
158         return privLevel;
159     }
160 
161     /// <summary> Binding Context of user from button
        2 references
164     public UserInfo StoreUserFromButton(object sender)
165     {
166 
167         var button = sender as Button;
168         if (button == null)
169         {
170             return null;
171         }
172         selectedUser = button.BindingContext as UserInfo;
173         if (selectedUser != null)
174         {
175             return selectedUser;
176         }
177         return null;
178 
179     }
```

Fig.3 UserAccessPage Supporting Methods

- Added supporting Methods (fig.3)
  - Lines 131-158: display visual prompt for user to select Privilage level to be requested and return the privilage level.

- Lines 164 - 177 : new method that returns the user object being selected from the button being pressed. (This methods is shared by multiple method to avoid repeat code)
- Added Method (Fig.2) that is called when button is pressed and utilizes both of the fig.3 supporting methods and also calls the *UserAccessDB* class method (this method is shown below in fig.4) passing in the users ID and privilage level chosen.

UserAccessDB

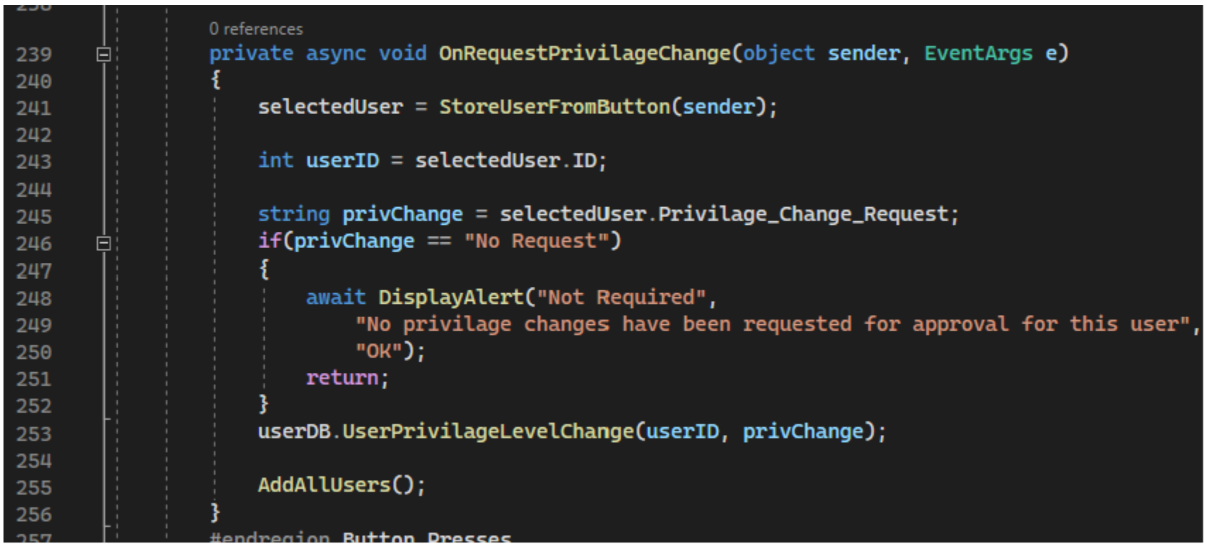


```
public void UserPrivilageLevelChange(int userID, string privalage_LevelReqeust)
{
    UserInfo userToUpdate = connection.Table<UserInfo>().FirstOrDefault(b => b.ID == userID);
    if (userToUpdate != null)
    {
        if (privalage_LevelReqeust == "Low")
        {
            userToUpdate.Privilage_Level = privalage_LevelReqeust;
            userToUpdate.Privilage_Change_Request = "No Request";
        }
        else
        {
            userToUpdate.Privilage_Change_Request = privalage_LevelReqeust;
        }
        connection.Update(userToUpdate);
    }

}
```

Fig.4 UserAccessDB Request privilage changes

- Added Method to handle CRUD operation for the newly added fields:
  - Updates user's privilage level field to "Low" if the request passed in is "Low" and sets privilage request field to "No Request".
  - Only updates the privilage change request field in database to match the request passed in if any other request is given (I.E "Medium" or "High").

UserPage

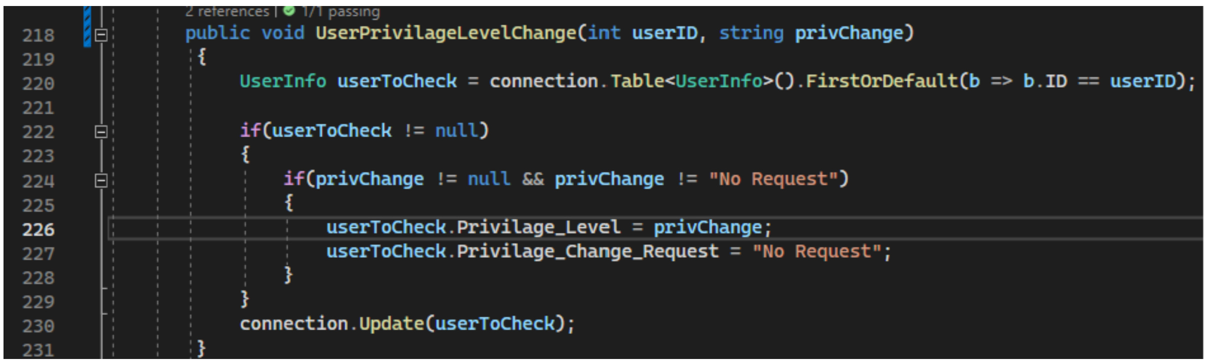


```
0 references
private async void OnRequestPrivilageChange(object sender, EventArgs e)
{
    selectedUser = StoreUserFromButton(sender);

    int userID = selectedUser.ID;

    string privChange = selectedUser.Privilage_Change_Request;
    if(privChange == "No Request")
    {
        await DisplayAlert("Not Required",
            "No privilage changes have been requested for approval for this user",
            "OK");
        return;
    }
    userDB.UserPrivilageLevelChange(userID, privChange);

    AddAllUsers();
}
#endregion Button Presses
```

Fig.5 UsersPage Approving Changes

- Added Method that:
- Grabs the current request status of a user.
- Calls method from *UserDB* class to handle if status is anything other than "No Request" (I.E "High"/"Low") (this method is shown in fig.6) passing in a Users ID and And the current Request status.

UserDB



```
2 references | 1/1 passing
public void UserPrivilageLevelChange(int userID, string privChange)
{
    UserInfo userToCheck = connection.Table<UserInfo>().FirstOrDefault(b => b.ID == userID);

    if(userToCheck != null)
    {
        if(privChange != null && privChange != "No Request")
        {
            userToCheck.Privilage_Level = privChange;
            userToCheck.Privilage_Change_Request = "No Request";
        }
    }
    connection.Update(userToCheck);
}
```

Fig. 6 UserDB approve/make privilage changes to Database

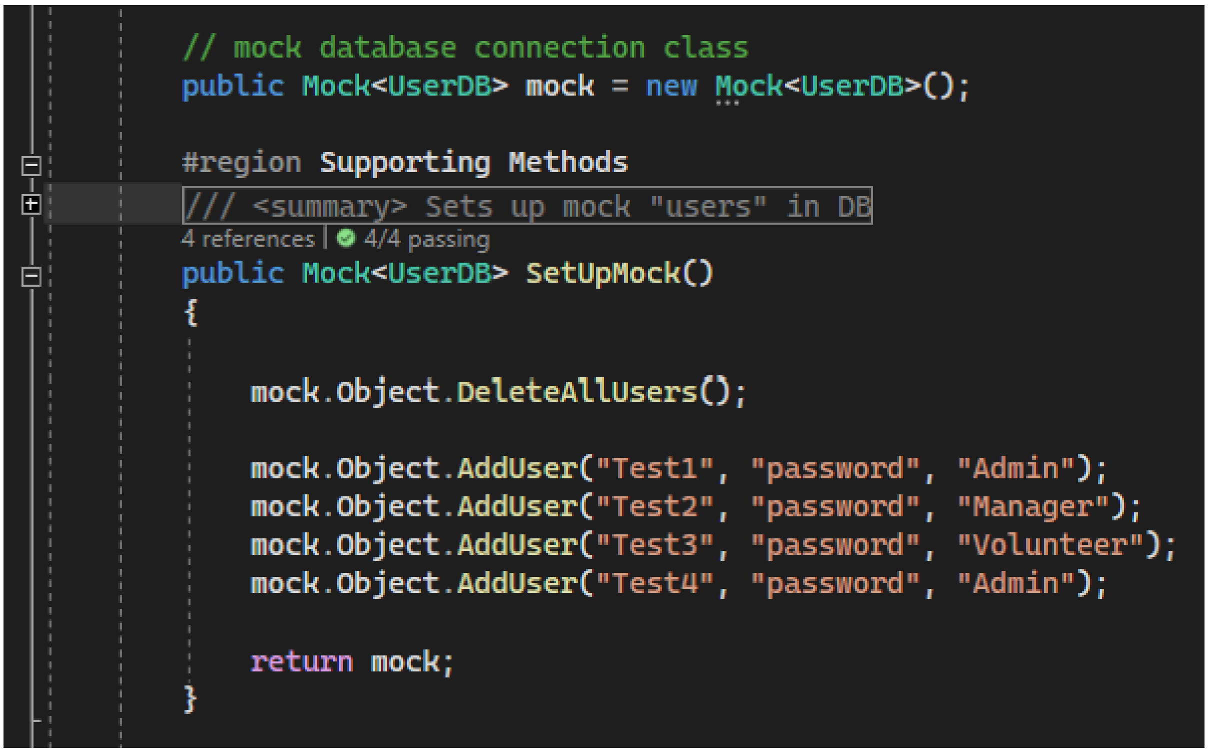
- Method Updates privilage level in database with the Privilage level request passed in.

Principles

Much of the principles utilsised here have been covered in prior submissions of the portfolio e.g single responisbility, YAGNI, KISS.

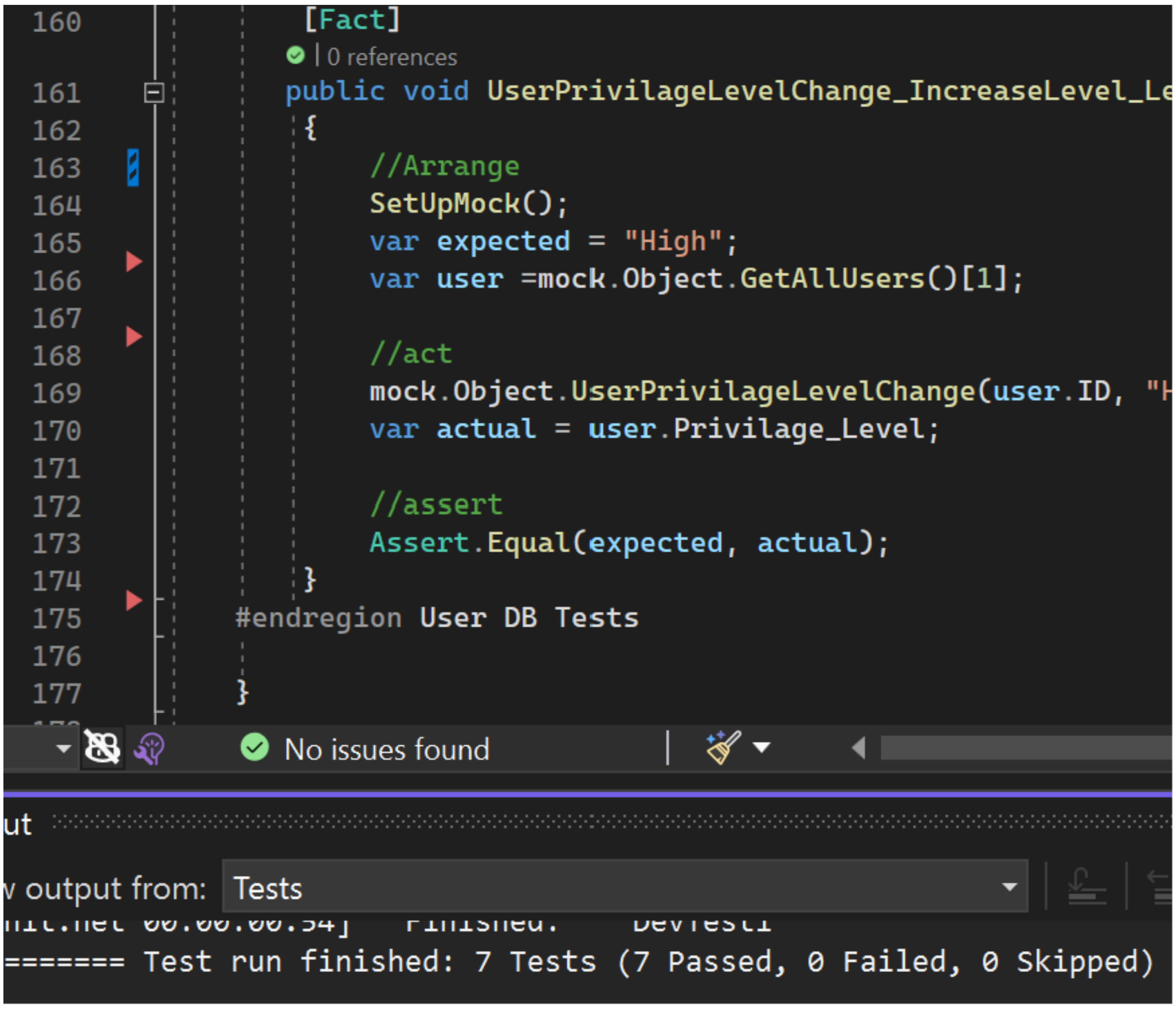
One that has stood out is my improvement of the principle DRY as I have began examining my code more carefully and been more mindfull in thinking of ways to create new supporting methods to eliminate repeat code, as discussed earlier regarding fig.3 many buttons where using the same code which I had previously overlooked.

Test Code



```
// mock database connection class
public Mock<UserDB> mock = new Mock<UserDB>();

#region Supporting Methods
/// <summary> Sets up mock "users" in DB
4 references | 4/4 passing
public Mock<UserDB> SetUpMock()
{

    mock.Object.DeleteAllUsers();

    mock.Object.AddUser("Test1", "password", "Admin");
    mock.Object.AddUser("Test2", "password", "Manager");
    mock.Object.AddUser("Test3", "password", "Volunteer");
    mock.Object.AddUser("Test4", "password", "Admin");

    return mock;
}
```

Fig. 7 Supporting Test Method



```
        [Fact]
        0 references
        public void UserPrivilageLevelChange_IncreaseLevel_Le
        {
            //Arrange
            SetUpMock();
            var expected = "High";
            var user =mock.Object.GetAllUsers()[1];

            //act
            mock.Object.UserPrivilageLevelChange(user.ID, "H
            var actual = user.Privilage_Level;

            //assert
            Assert.Equal(expected, actual);
        }
    #endregion User DB Tests

    }
```

```
======= Test run finished: 7 Tests (7 Passed, 0 Failed, 0 Skipped)
```

Fig. 8 Test Method Using Moq

For this weeks testing I have introduced Moq (Mocking Framework) to my testing environment which the team planned to use.

I am still getting used to this framework so I have created a method that populates the database with users by simply using the *AddUser()* method of the mocked *UserDB* class.(Fig. 7) I felt this to be usefull method for tests requiring to pull from the database and keep them performing a single responisibilty.

Fig.8 shows a test I have done on the *UserPrivilageLevelChange()* method. This Method utilizes the set up method shown in Fig.7, grabs the second user from the database list created by the set up method (which will have a privilage level of "Medium" by defualt) and then runs the method being tested to check if when "High" is passed in as a parameter the users Privilage level will be updated accordingly.

The bottom of Fig.8 shows all tests passing which includes the Test shown in Fig.8.

Code Review

Review By Team Member

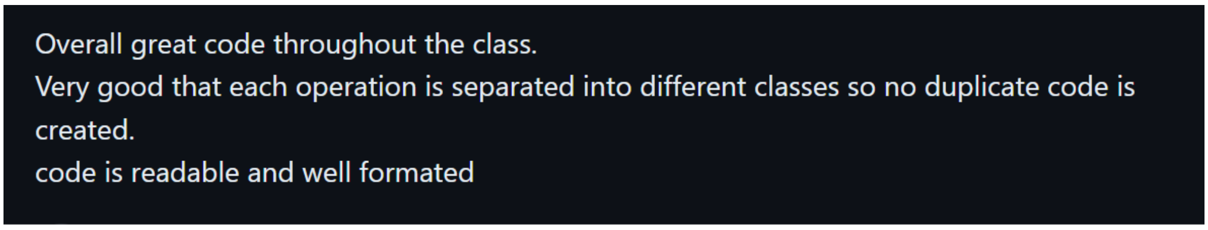
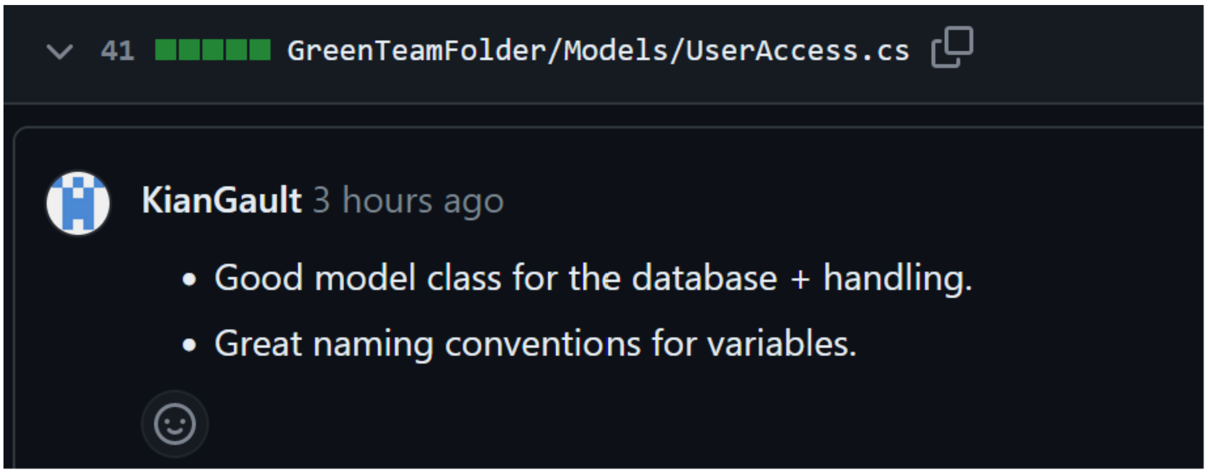

Overall great code throughout the class.
Very good that each operation is separated into different classes so no duplicate code is created.
code is readable and well formated

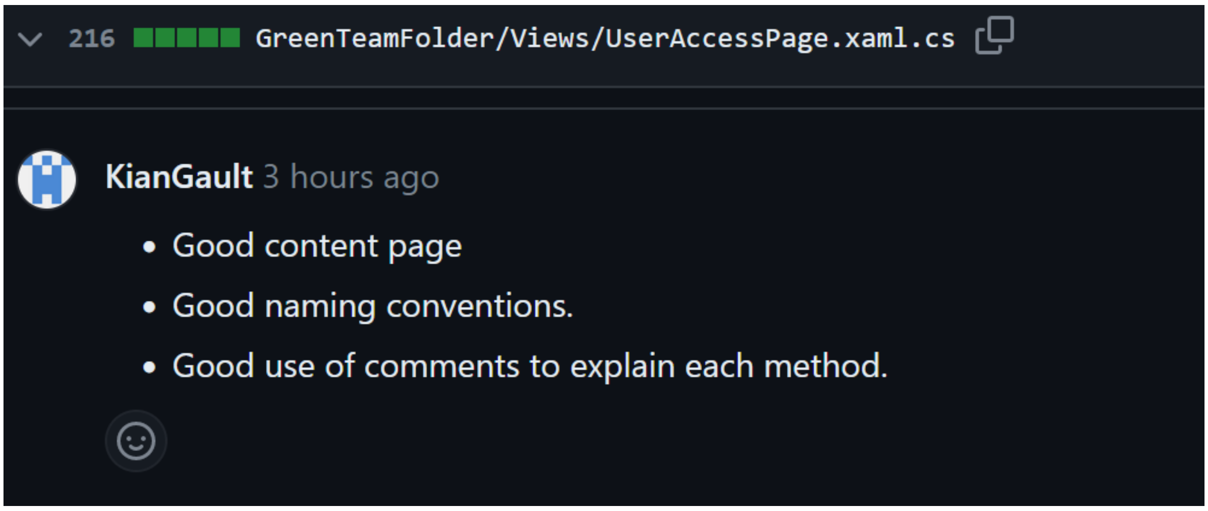
Fig. 9 Overall Review


41 GreenTeamFolder/Models/UserAccess.cs

KianGault 3 hours ago

- Good model class for the database + handling.
- Great naming conventions for variables.

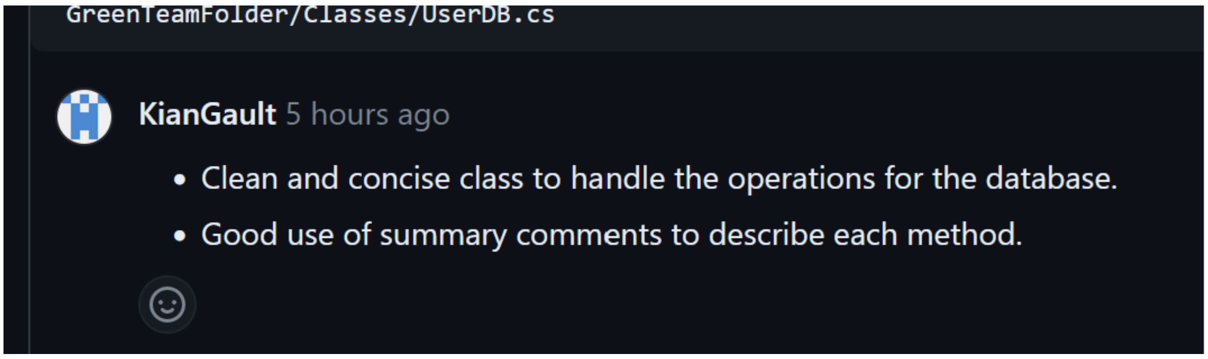
Fig. 10 UserAccess review


216 GreenTeamFolder/Views/UserAccessPage.xaml.cs

KianGault 3 hours ago

- Good content page
- Good naming conventions.
- Good use of comments to explain each method.

Fig. 11 UserAccessPage review


GreenTeamFolder/Classes/UserDB.cs

KianGault 5 hours ago

- Clean and concise class to handle the operations for the database.
- Good use of summary comments to describe each method.

Fig. 12 UserDB review

Overall The feedback was positive with no changes requested this week as shown in the above Figs.

Review On Team Member

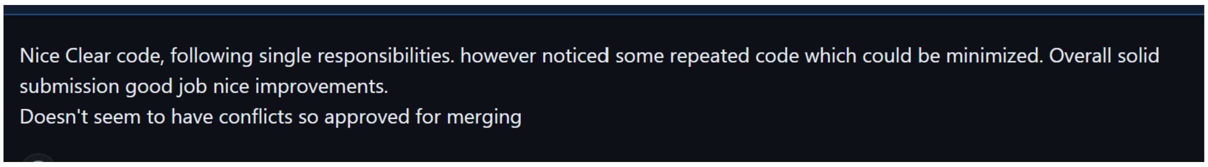
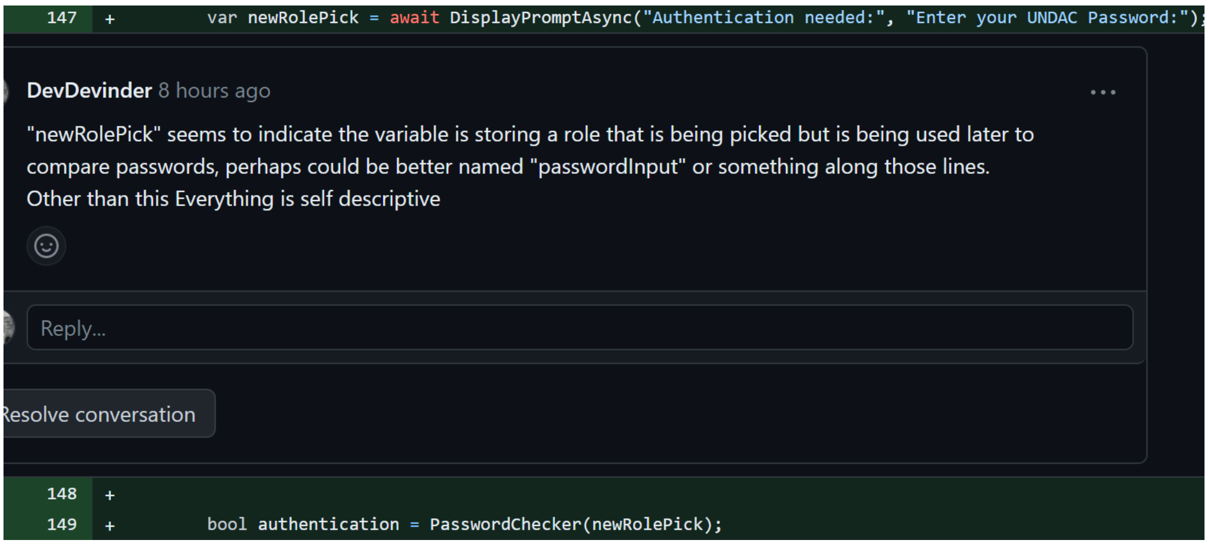

Nice Clear code, following single responsibilities. however noticed some repeated code which could be minimized. Overall solid submission good job nice improvements.
Doesn't seem to have conflicts so approved for merging

Fig. 13 Overall Review


```
147 +         var newRolePick = await DisplayPromptAsync("Authentication needed:", "Enter your UNDAC Password:");
```
DevDevinder 8 hours ago

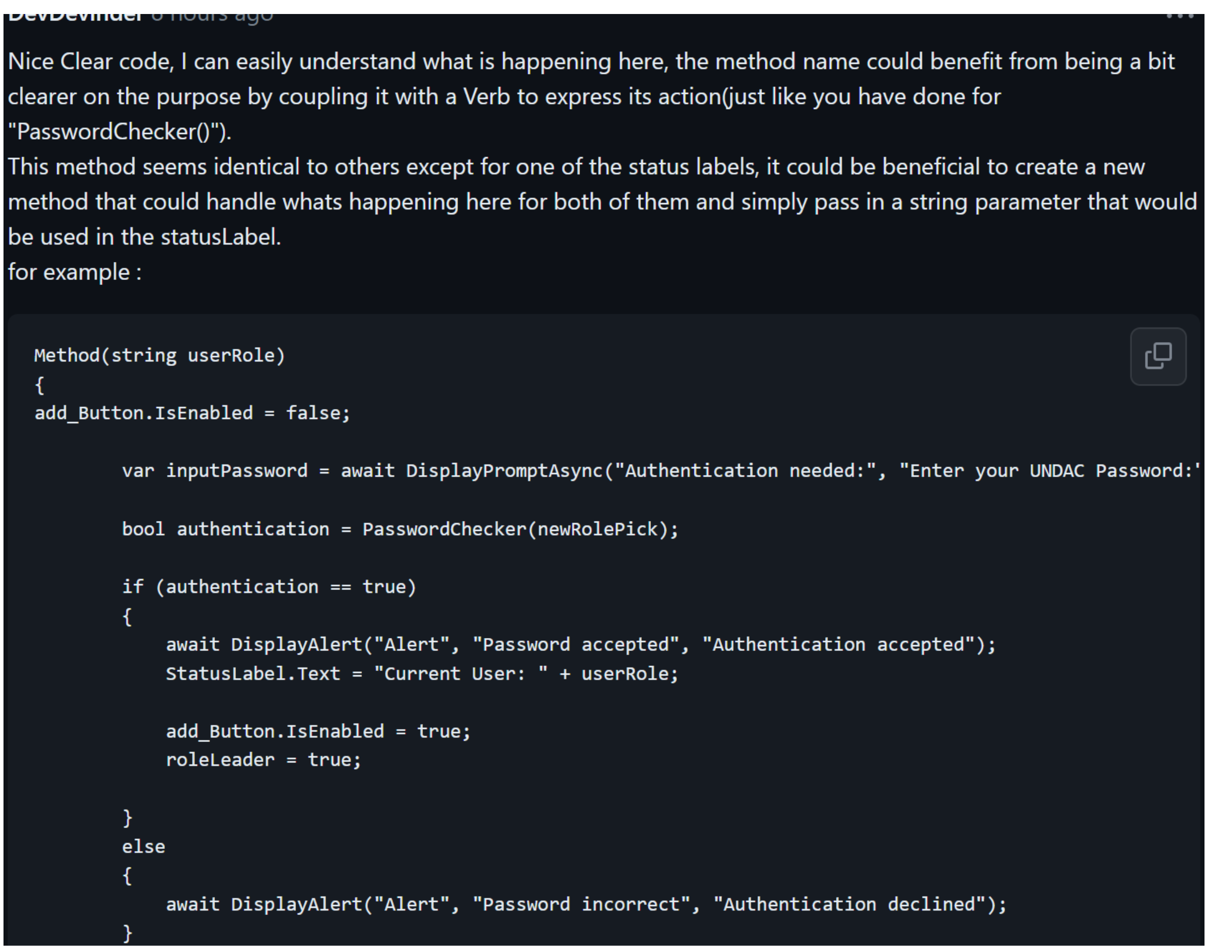
"newRolePick" seems to indicate the variable is storing a role that is being picked but is being used later to compare passwords, perhaps could be better named "passwordInput" or something along those lines.
Other than this Everything is self descriptive

Reply...

Resolve conversation

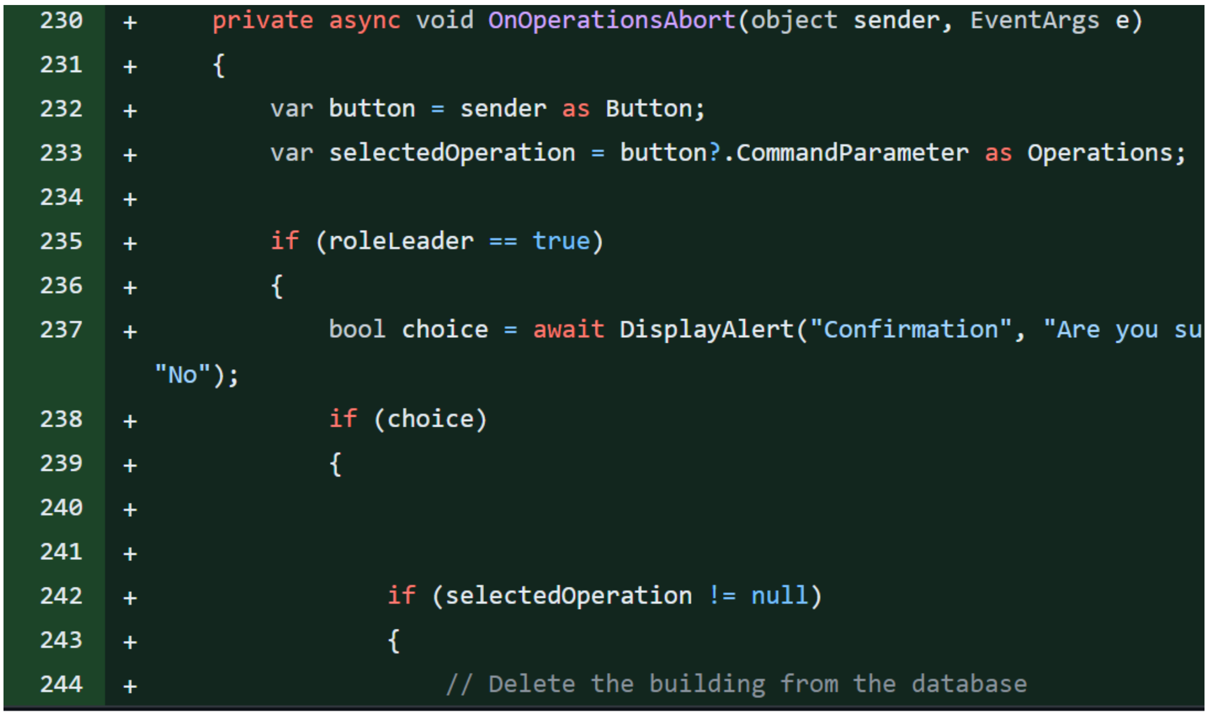
```
148 +
149 +         bool authentication = PasswordChecker(newRolePick);
```

Fig. 14 UserAccess review


Nice Clear code, I can easily understand what is happening here, the method name could benefit from being a bit clearer on the purpose by coupling it with a Verb to express its action(just like you have done for "PasswordChecker()").
This method seems identical to others except for one of the status labels, it could be beneficial to create a new method that could handle whats happening here for both of them and simply pass in a string parameter that would be used in the statusLabel.
for example :

```
Method(string userRole)
{
add_Button.IsEnabled = false;

        var inputPassword = await DisplayPromptAsync("Authentication needed:", "Enter your UNDAC Password:'

        bool authentication = PasswordChecker(newRolePick);

        if (authentication == true)
        {
            await DisplayAlert("Alert", "Password accepted", "Authentication accepted");
            StatusLabel.Text = "Current User: " + userRole;

            add_Button.IsEnabled = true;
            roleLeader = true;

        }
        else
        {
            await DisplayAlert("Alert", "Password incorrect", "Authentication declined");
        }
```

Fig. 15 UserAccessPage review



```
+     private async void OnOperationsAbort(object sender, EventArgs e)
+     {
+         var button = sender as Button;
+         var selectedOperation = button?.CommandParameter as Operations;
+
+         if (roleLeader == true)
+         {
+             bool choice = await DisplayAlert("Confirmation", "Are you su
  "No");
+             if (choice)
+             {
+
+
+                 if (selectedOperation != null)
+                 {
+                     // Delete the building from the database
```

Fig. 16 UserDB review

I had found that the code I reviewed was well written and very easy to understand both what is happening in each method as well as the responisbility of each variable. I had noticed that in some instances, naming could be improved. Such as a method name soley being a made up of nouns and a variable that's name did not match its intended use. I recomended introducing a verb to express the purpose of the class and to rename the variable to a more suitable one.

I had discovered some repeat code being used and advised to make a new method to take on the responsibility of the code being repeated which should drastically shorten the code and allow for easier maintenance.

I sense that some code has been copied and pasted from other methods which may be the reason for the variable named not matching its purpose.

I have reviewed this team members code before and notice improvements in naming conventions as a whole.

Reflections

This week I introduced myself to mocking with "*Moq*" and found that it can be really usefull for unit testing the CRUD side of the application.

I noticed that although I have gotten my testing environment functional and another team member had managed at one point check to it works on one of their machines successfully before merging however the team have decided not to allow the branch to merge as they are concernend this may cause set backs for them and would rather not try it.

This is an issue for me as due to this I am currently locked out of being able to merge and now my code environment is not matching the rest of the teams. Hopefully this is resolved by next weeks submission as it means I am working outside the agreed workflow with a new workflow more aimed at preserving the main branch at all cost and either give up on testing or work independantly from my branch.

I have also noticed that the team appears to be smaller and I assume members have joined another team to recieve additional support. One issue with this is as a team it was unnexpected and confusing as there was no notification of members leaving. fortunatly one of the benifits of this is there are now more tasks for myself to choose from that are similar to ones I have currently worked on.

This has shown me the importance of writing code following Software engineering principles and doing code reviews as when someone leaves unexpectedly without a trace it is important to be able to read and understand their code that has been left behind.

During his weeks code review I found my code to be a lot more modular, has less code smells such as "Duplicate code" and "Mysterious Name" and will reach a higher Internal quality criteria (also due to having test cases) My team member is Improving at a fast rate and was relying a bit more on comments to explain the code in previous weeks but has now removed uneccessary comments and instead now uses clear self descriptive code writing.So it is good to see previous advice on comments being implemented well by team members.

In Conclusion I feel I am still progressing and refining myself but at the same rate as previous weeks in terms of software principles, I do appreciate being more aware of the principles now and feel its becoming more habitual to ensure I am following them.