Implementation

Project work 4

Breif

This week I worked on improving code and to begin work on the following Task Issue:

```
"As a system administrator, I want to maintain reference values for position
statuses
```

End user goal: To be able to list, create, update and delete reference values for position statuses

End business goal: To have appropriate position statuses available to describe a person's position in an organisation (e.g. 'active', 'retired', 'seconded out')

Acceptance criteria:

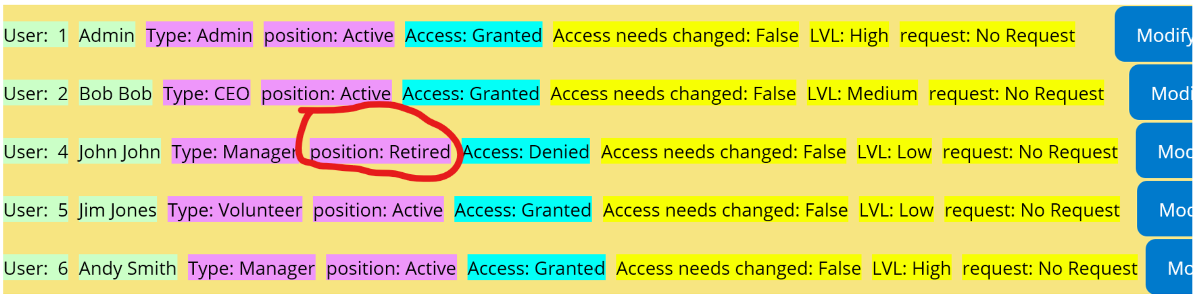
```
A position status item can be created, viewed, updated and deleted (CRUD
functionality)
```

Measurement of success:

```
Unit tests pass for all CRUD operations
```

Notes:

```
The database table will be called position_status
The table will have a single column, name"
```



User: 1	Admin	Type: Admin	position: Active	Access: Granted	Access needs changed: False	LVL: High	request: No Request	Modify
User: 2	Bob Bob	Type: CEO	position: Active	Access: Granted	Access needs changed: False	LVL: Medium	request: No Request	Modi
User: 4	John John	Type: Manager	position: Retired	Access: Denied	Access needs changed: False	LVL: Low	request: No Request	Moc
User: 5	Jim Jones	Type: Volunteer	position: Active	Access: Granted	Access needs changed: False	LVL: Low	request: No Request	Moc
User: 6	Andy Smith	Type: Manager	position: Active	Access: Granted	Access needs changed: False	LVL: High	request: No Request	Mc

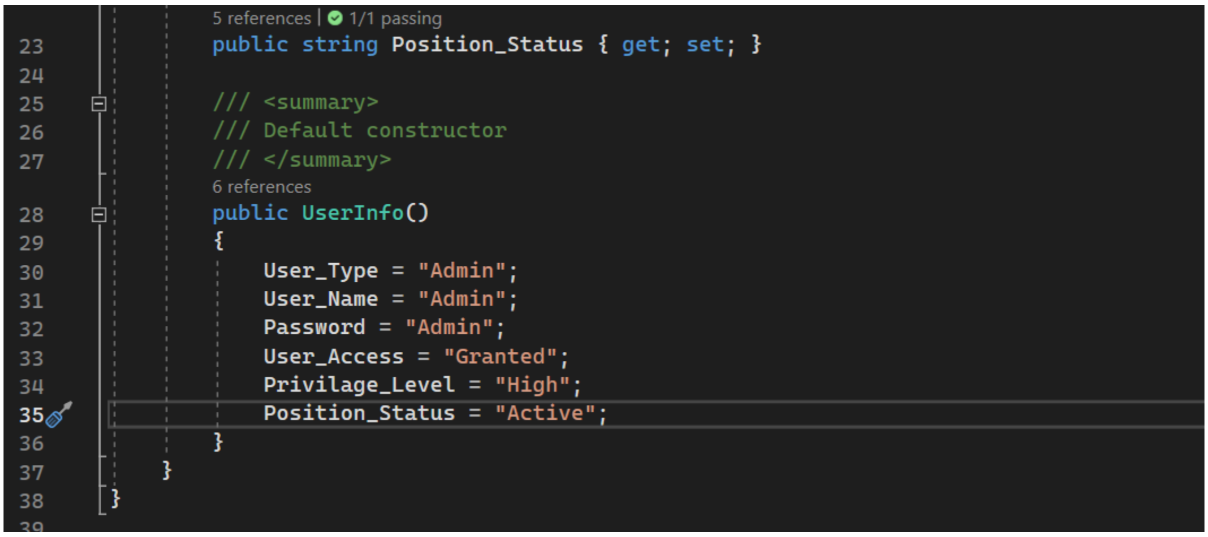
Fig.1 position Status Displayed

I took a decision to simply include a field "Position_Status" to the userInfo table and have the default status set to "Active" when ever a user is added. The user page which displays and provides the interface for managing users now displays position status(Shown in Fig.1) and communicates with the userDB class to handle updating this field.

In future updates I will implement a table for Position_Status that focuses only on this field with an ID column so that other tables can pair this based on ID, that way other classes can freely use the Position_Status table in a more modular fashion.

Changes Made

UserInfo

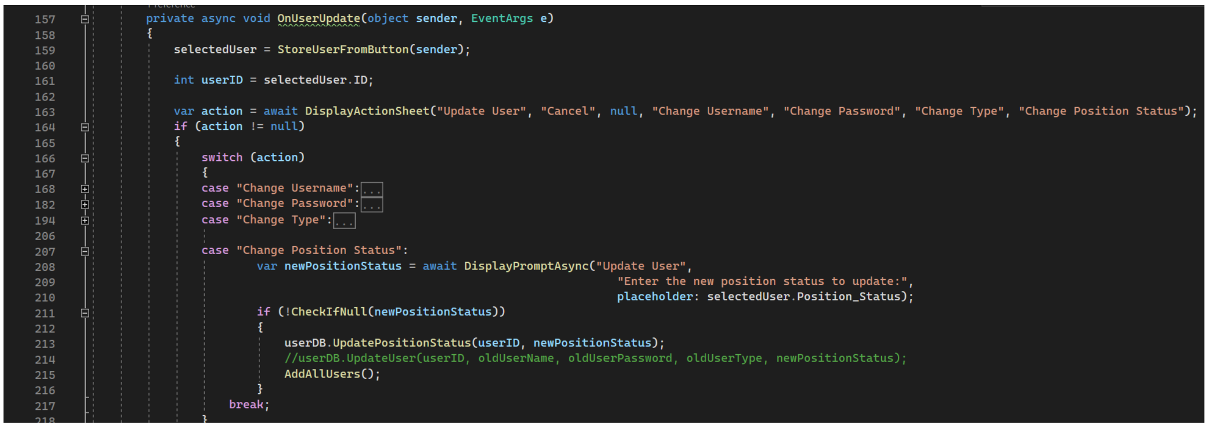


```
            5 references | 1/1 passing
23          public string Position_Status { get; set; }
24
25          /// <summary>
26          /// Default constructor
27          /// </summary>
            6 references
28          public UserInfo()
29          {
30              User_Type = "Admin";
31              User_Name = "Admin";
32              Password = "Admin";
33              User_Access = "Granted";
34              Privilage_Level = "High";
35              Position_Status = "Active";
36          }
37      }
38  }
```

Fig.2 User Info Table

- Database table additions:
  - added field to store Position Status. (Line 123)
  - Added Default Status "Active" (line 35)

UserPage

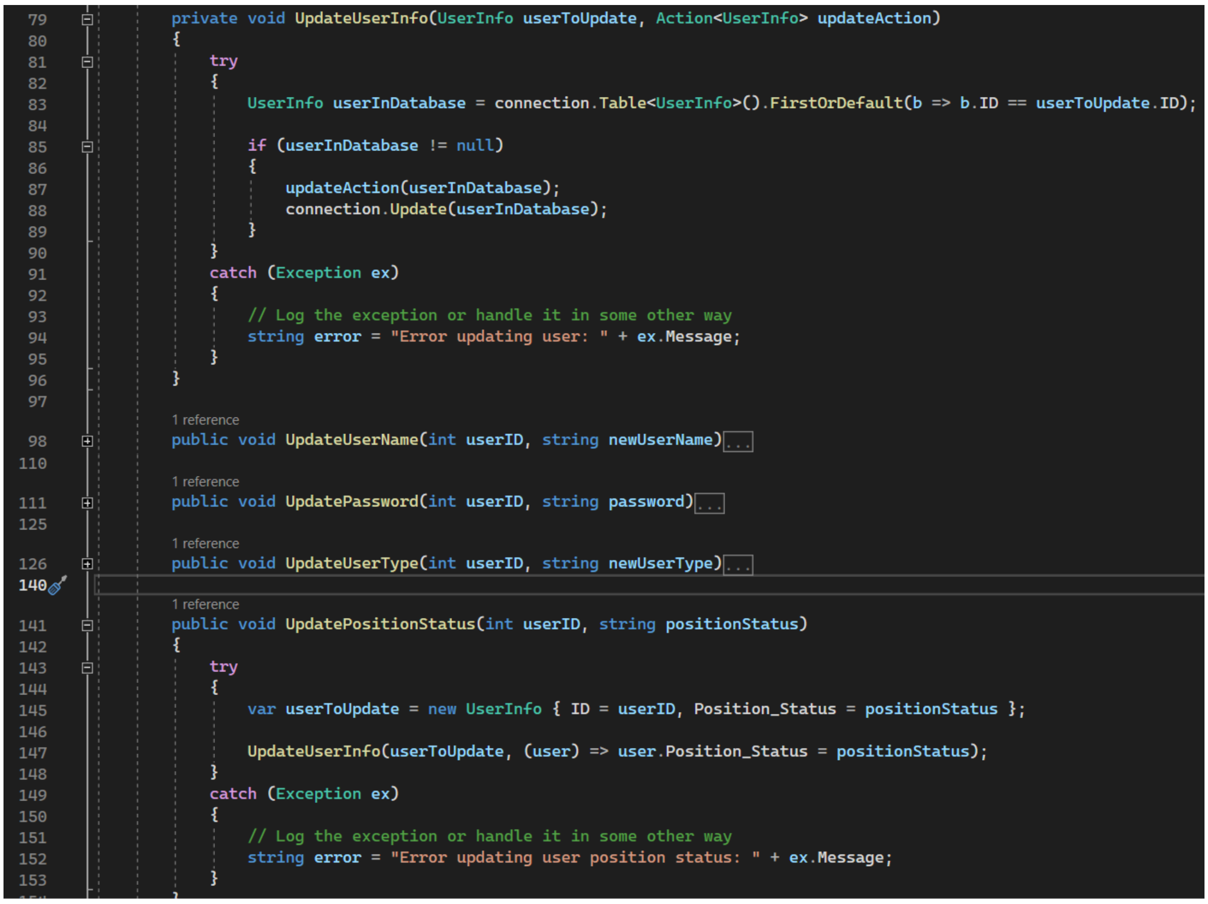


```
157     private async void OnUserUpdate(object sender, EventArgs e)
158     {
159         selectedUser = StoreUserFromButton(sender);
160
161         int userID = selectedUser.ID;
162
163         var action = await DisplayActionSheet("Update User", "Cancel", null, "Change Username", "Change Password", "Change Type", "Change Position Status");
164         if (action != null)
165         {
166             switch (action)
167             {
168             case "Change Username":...
182             case "Change Password":...
194             case "Change Type":...
206
207             case "Change Position Status":
208                     var newPositionStatus = await DisplayPromptAsync("Update User",
209                                                                      "Enter the new position status to update:",
210                                                                      placeholder: selectedUser.Position_Status);
211                     if (!CheckIfNull(newPositionStatus))
212                     {
213                         userDB.UpdatePositionStatus(userID, newPositionStatus);
214                         //userDB.UpdateUser(userID, oldUserName, oldUserPassword, oldUserType, newPositionStatus);
215                         AddAllUsers();
216                     }
217                 break;
```

Fig.3 Update Position status

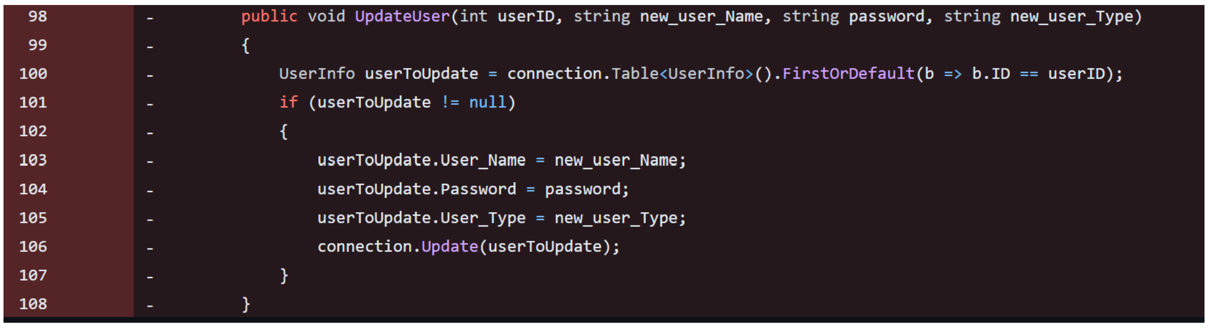
- Added option to action sheet for position status (end of line 163):
- Added a case in the switch case for position status which communicates with *UserDB* class to allow system users to update the status() based on string input (lines 207- 217).

UserDB



```
 79    private void UpdateUserInfo(UserInfo userToUpdate, Action<UserInfo> updateAction)
 80    {
 81        try
 82        {
 83            UserInfo userInDatabase = connection.Table<UserInfo>().FirstOrDefault(b => b.ID == userToUpdate.ID);
 84
 85            if (userInDatabase != null)
 86            {
 87                updateAction(userInDatabase);
 88                connection.Update(userInDatabase);
 89            }
 90        }
 91        catch (Exception ex)
 92        {
 93            // Log the exception or handle it in some other way
 94            string error = "Error updating user: " + ex.Message;
 95        }
 96    }
 97
       1 reference
 98    public void UpdateUserName(int userID, string newUserName)...
110
       1 reference
111    public void UpdatePassword(int userID, string password)...
125
       1 reference
126    public void UpdateUserType(int userID, string newUserType)...
140
       1 reference
141    public void UpdatePositionStatus(int userID, string positionStatus)
142    {
143        try
144        {
145            var userToUpdate = new UserInfo { ID = userID, Position_Status = positionStatus };
146
147            UpdateUserInfo(userToUpdate, (user) => user.Position_Status = positionStatus);
148        }
149        catch (Exception ex)
150        {
151            // Log the exception or handle it in some other way
152            string error = "Error updating user position status: " + ex.Message;
153        }
```

Fig. 4 changes to user Update methods



```
 98  -        public void UpdateUser(int userID, string new_user_Name, string password, string new_user_Type)
 99  -        {
100  -            UserInfo userToUpdate = connection.Table<UserInfo>().FirstOrDefault(b => b.ID == userID);
101  -            if (userToUpdate != null)
102  -            {
103  -                userToUpdate.User_Name = new_user_Name;
104  -                userToUpdate.Password = password;
105  -                userToUpdate.User_Type = new_user_Type;
106  -                connection.Update(userToUpdate);
107  -            }
108  -        }
```

Fig. 5 Previous Method System

- Made changes to user update methods which handle CRUD operations for the UserInfo table:
    - Added Helper method to reduce repeate code(Lines 79-96)
    - Created new methods (lines 98 - 141) to focus on each table field being updated (each method is the same with the exceptions for the parameter passed in and table value to be changed)
    - Began wrapping code with Try Catch blocks to log any errors caught when excecuting the code (Lines 143- 153)

Principles

This week I havent added as much code to the project as previous weeks and instead Focused on making some improvements to the code to try and improve Internal quality such as adding try catch blocks to help with debugging and catching errors and spliting the original method (Fig.5) of the *UserDB* class into a new set of methods, as there was already a large amount of parameters in the original method which was not something that would be viewed as clean code or good practice since this wasnt necessary and easily avoidable.

further refactoring would be benificial as since writing the report I realised that instead of splitting the methods in Fig.4 I could have used another parameter to get the column name to be changed and wrote a method like this:

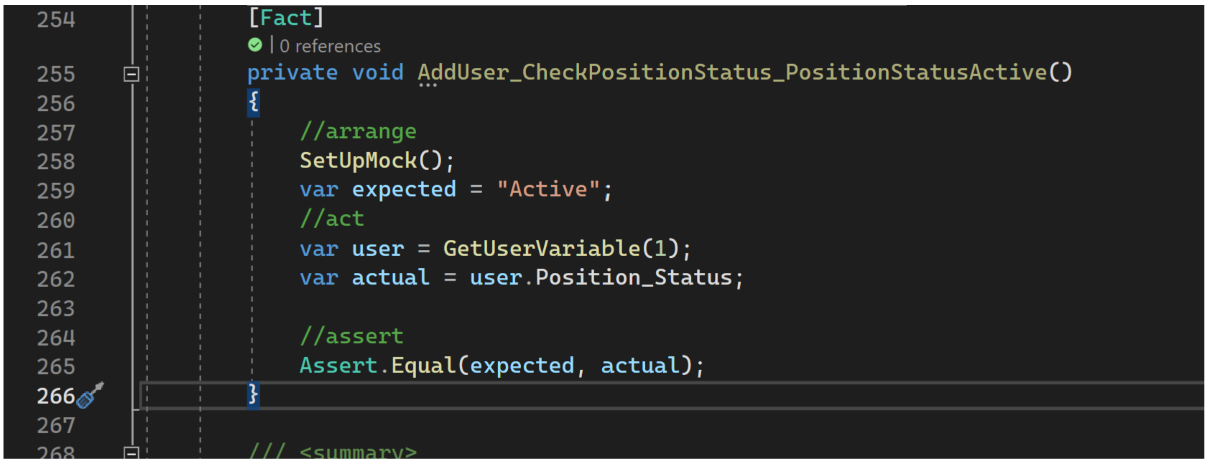
```
    public void UpdateUserExample(int userID, string propertyName, string newValue)
    {

            var userToUpdate = new UserInfo { ID = userID };
            UpdateUserInfo(userToUpdate, (user) =>
  user.GetType().GetProperty(propertyName).SetValue(user, newValue));
    }
```

This would reduce repeate code, and have a consice and readible way to update users whilst keeping the parameter count down to a suitable number.

Other changes I would make to my code is shortening method names as "UpdateUserName" for example is quite large and something like "SetUserName" would be a bit more consice.

Test Code



```
[Fact]
0 references
private void AddUser_CheckPositionStatus_PositionStatusActive()
{
    //arrange
    SetUpMock();
    var expected = "Active";
    //act
    var user = GetUserVariable(1);
    var actual = user.Position_Status;

    //assert
    Assert.Equal(expected, actual);
}

/// <summary>
```

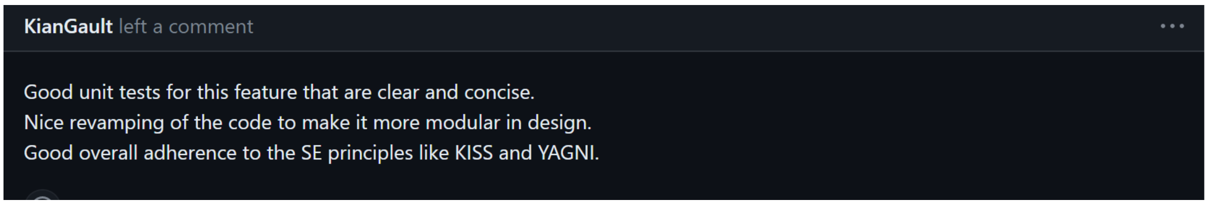
Fig.6 Testing users are assigned default Status

For this weeks testing in regards to the issue I have tested Position status's being asigned the default value when users are added which has passed.

As the CRUD operations are using a private method I still need time to learn using Reflections with moqs but I manually tested the code and the results are the system working as intended.

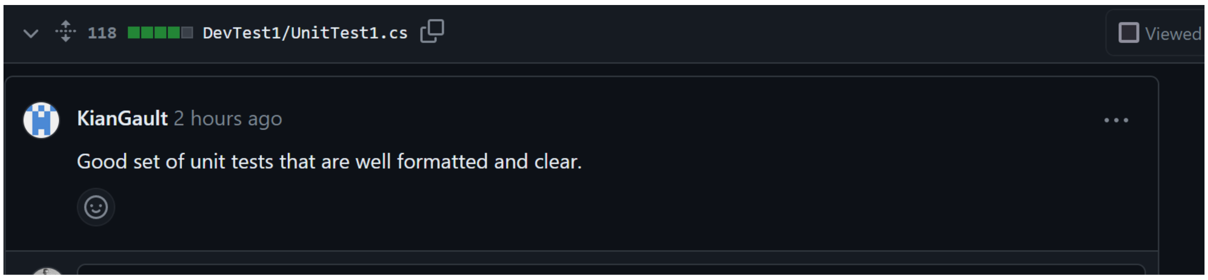
All other unit tests for my code have passed.

Reviews on myself


KianGault left a comment

Good unit tests for this feature that are clear and concise.
Nice revamping of the code to make it more modular in design.
Good overall adherence to the SE principles like KISS and YAGNI.

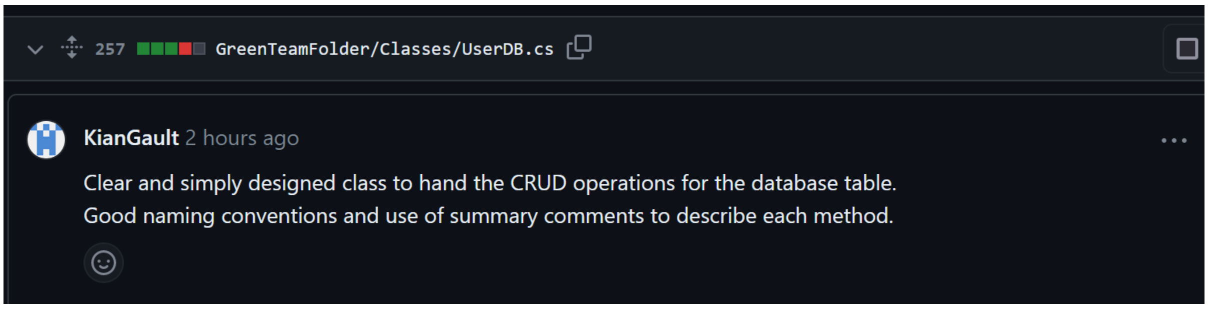
Fig.7 Review Overall


118 DevTest1/UnitTest1.cs

KianGault 2 hours ago

Good set of unit tests that are well formatted and clear.

Fig.8 Review Unit Tests


257 GreenTeamFolder/Classes/UserDB.cs

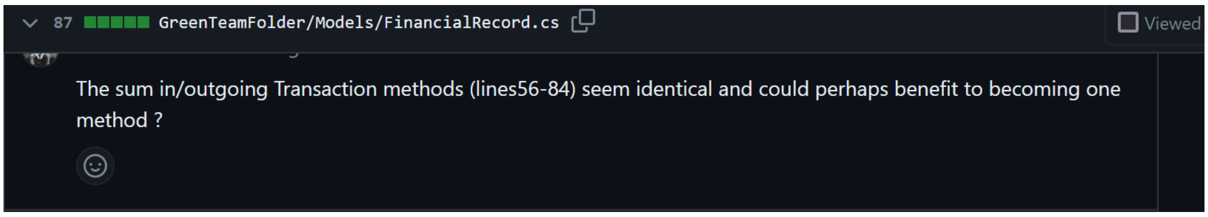
KianGault 2 hours ago

Clear and simply designed class to hand the CRUD operations for the database table.
Good naming conventions and use of summary comments to describe each method.

Fig.8 UserDB

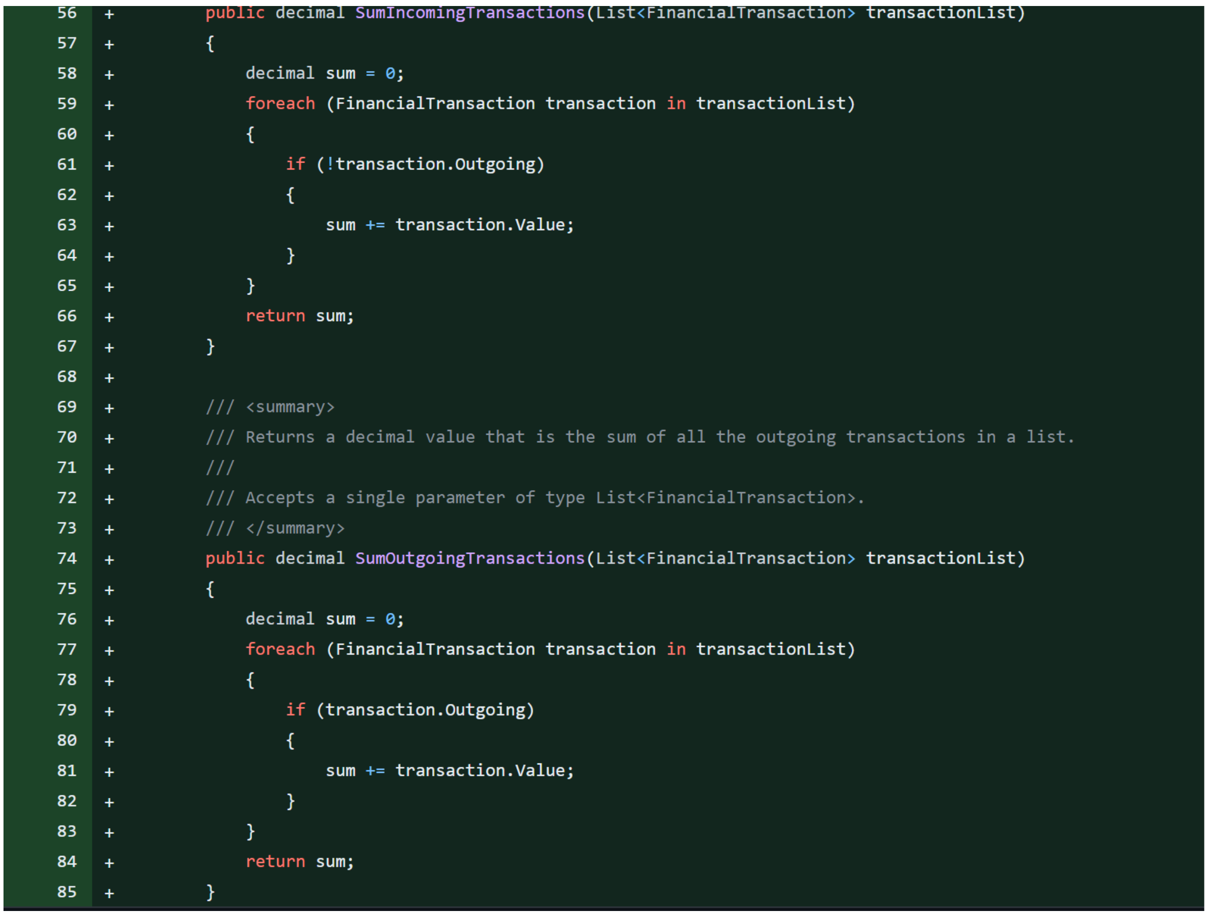
Overall The feedback was positive with no changes requested this week as shown in the above Figs.

Review On Team Member


87 GreenTeamFolder/Models/FinancialRecord.cs

The sum in/outgoing Transaction methods (lines56-84) seem identical and could perhaps benefit to becoming one method ?

Fig.9 My Review



```
+        public decimal SumIncomingTransactions(List<FinancialTransaction> transactionList)
+        {
+            decimal sum = 0;
+            foreach (FinancialTransaction transaction in transactionList)
+            {
+                if (!transaction.Outgoing)
+                {
+                    sum += transaction.Value;
+                }
+            }
+            return sum;
+        }
+
+        /// <summary>
+        /// Returns a decimal value that is the sum of all the outgoing transactions in a list.
+        ///
+        /// Accepts a single parameter of type List<FinancialTransaction>.
+        /// </summary>
+        public decimal SumOutgoingTransactions(List<FinancialTransaction> transactionList)
+        {
+            decimal sum = 0;
+            foreach (FinancialTransaction transaction in transactionList)
+            {
+                if (transaction.Outgoing)
+                {
+                    sum += transaction.Value;
+                }
+            }
+            return sum;
+        }
```

Fig.10 Code Reviewed

The code from this team member was very clear and consistent, the only issue I had found was that two methods are highly similar and could benifit from being merged into one method as it is venturing onto violating DRY.

Reflections

Workflow

Half way into the weak one member of the team merged the project branches into the main which has been a great help as now everyone is working with the same file patterns and environment again.

Recently everyone has been working on random tasks and it has been difficult to know which tasks are already being worked on as members who have left the team are not removed from the repo and their tasks are still assigned to them in the project board. without a clear indication who is still in the team and who is not I was advised by existing team members to pick random tasks and to just work on them.

We havent really been doing any planning and instead just goin straight into any task that we wish which is a system were all used to now as we are familiar with our capabilities but more seems like an individualistic survival workflow to just achieve enough content for a portfolio submission.

Moving forward

I would like to have achieved more this week in terms of refactoring my entire submissions so far and adding in a few more classes specifically for Position Status's but due to time constraints with work and other coursework and waiting for the merge of branches I have used my time to achieve what I have outlined in the *Brief* section above.

Moving forward I would like to review the class notes on code practices and improve my code accordingly before progressing onto any other task to take advantage of the available matierial.

Conclusion

This week I dont feel that I have learned much more in terms of principles from the project work itself, Code reviews on my submissions have been mainly positive during the project weeks so although it is a good sign about how im writing code it does limit my learning potential from others when there isn't much constructive feedback but during each portfolio submission I write I tend to self review my Code further even if unintentionally and I have been noticing ways to improve them for the following week.