Project Work Week 5 - MVVM 2

1. Introduction

Team Repo

This week was a continuation of the previous week where I started to implement the MVVM pattern. I continued with this task, and successfully implemented mocking and true separation using MVVM. My work is not complete in separation, but it is incredibly close, with results starting to show. Even though this is the final week in the portfolio, I will be continuing to work on this project, as I am very interested learning more and improving as a developer.

I will be doing the following in this portfolio:

- Provide a descriptive summary of the task I chose.
- Provide snippets of the code I created with a commentary to show good coding practices.
- A descriptive summary of the test code I wrote.
- A reflective summary of any changes requested during code review, along with fixes if required.
- A descriptive summary of the issues I found with the code I was asked to review.
- The final step will be a reflection with additional content pertaining to improvements and so on.

2. Descriptive Summary

This week, my task, as discussed, was to continue with developing the MVVM pattern which I assigned to myself last week on GitHub. I have been making strides this week especially. I have completed separation of a number of classes, implemented numerous interfaces which I am now able to mock effectively and will be shown in the testing section of this document. I also continued developing my skills with lambda expressions which I will be making use of in the testing section of this portfolio.

3. Code Snippets

The first code snippet this week is as follows:

```
/// <summary>
/// MemberViewModel using depedency injection for full seperation of concerns,
relay commands for async tasks
/// </summary>
/// <param name="teamModel"></param>
public MemberViewModel(TeamModel teamModel)
{
    _teamModel = teamModel;

    CollectionViewSC = new RelayCommand<MemberModel>
(CollectionView_SelectionChanged);
    getItems = new AsyncRelayCommand(LoadDataAsync);
}

/// <summary>
```

```
/// Loading data from the database
/// </summary>
/// <returns></returns>
public async Task LoadDataAsync()
{

    var items = await _teamModel.getListFromDBAync();

    Items.Clear();
    foreach (var item in items)
    {
        Items.Add(item);
    }

}
```

This is a much larger snippet than usual. The reason for this is the method wouldn't be as interesting without the constuctor which is showing dependency injection. This is part of the efforts I have been making to separate the code. Now, due to the clear separation between the view, view model and model, I am using dependency injection, thus I don't need to make any calls to the database in the view model. This is a huge step forward in separation of concerns.

The method itself is a simple method which contacts the model, which in turn contacts the database and returns a list of items. This list is then added to the observable collection, called Items. This is then bound to the view, which is then displayed to the user.

This code shows a clear understanding of MVVM and separation of converns. Not only that, it is a display of clean code, it follows all coding principles, such as KISS, DRY, YAGNI and so on. It also has doxygen suitable comments which explain what the code does in a simple and informative manner. It follows the single responsibility principle as it only does one thing.

However, I am not oblivious to the improvements which could be made in this code. For example, this code could instead have an injection for the database instead of the model.

Also, although it hasn't been discussed, there are two RelayCommands in this code, which are bound to buttons in the view. This again is a display of separation of concerns.

4. Test Code

As discussed, testing this week has been done successfully, whilst implementing a mock which wasn't previous possible due to the tightly coupled view and view model and a complete lack of a model. The testing code is as follows:

```
/// <summary>
/// Test using Mocking to check if the list is returned from the database and the
method is called once.
/// </summary>
/// <returns></returns>
[Fact]
public async Task TestGetListFromDBAsync()
```

```
{
    var mockTeamMemberDB = new Mock<ITeamMemberDB>();
    var teamModel = new TeamModel(mockTeamMemberDB.Object);
    var expectedList = new List<MemberModel>();

    mockTeamMemberDB.Setup(db => db.GetItemsAsync()).ReturnsAsync(expectedList);
    var result = await teamModel.getListFromDBAync();

    Assert.Equal(expectedList, result);
    mockTeamMemberDB.Verify(db => db.GetItemsAsync(), Times.Once);
}
```

As can be seen, this test is a lot more complex than previous entries. This was developed not only by me, but by copilot.

The test itself isn't really that complicated. All it does is creates a mock of the database, then creates a new instance of the model, passing the mock as a parameter. Then a variable is created which is a list of the member model. However, the next part of the code is where it gets slightly more interesting. It is the setup of the mock, where when the method GetItemsAsync is called, it returns the expected list. It is called internally when the teamModel.getListFromDBAync() method is called.

Finally, we assert that the expected list is equal to the list which is returned from the method, and then, the part copilot added, we verify that the method was called once.

This test is a display of a number of things. Firstly, it is a display of mocking, which is a display of separation of concerns. Secondly, it is a display of the use of interfaces, as well as lambda expressions.

It is also a display of clean code, as it follows all coding principles, such as KISS, DRY, YAGNI and so on. It also has doxygen suitable comments which explain what the code does in a simple and informative manner. It follows the single responsibility principle as it only does one thing.

5. Code Review

This week I reviewed the code of one of my team members.

The coding expertise of my team has improved massively, with each each member showing development in a number of areas.

However, I find that there isn't a lot of interesting code to look at, as some members have been doing the same thing week in week out for a number of weeks. Thus, they are now really good at replicating it for numerous tasks.

This makes finding issues to discuss difficult. And this week, I wasn't able to find anything obvious, excluding the lack of MVVM, separation of concerns and so on.

Also, in this week, I wasn't able to have my code reviewed. This is due to incomplete code. Although I have a number of areas of the code working, it isn't finished. Thus having a team member review it would likely lead to answers such as "This code doesn't work" and the like.

Instead, strides have been made to show the reader of this portfolio that the effort has been made and shown to work.

6. Reflection

This week has been an incredibly fulfilling week. I have as discussed made incredible strides in my coding journey. I have managed to implement a semi working MVVM, fully working mocks whilst maintaining a separation of concerns. This has been the culmination of many weeks of work, and I am incredibly proud of the results.

I have a lot to learn, I am still not finished yet, I need to work on the other pages of this project as there are simply too many to do in a short space of time.

I have a beginner understanding of lamba expressions which is something discussed in previous portfolios. I am proud to get this working and show my results here.

My code quality has drastically increased, I can spot so many issues before they present themselves which saves time and makes my code look much better and functional.

I have been taking more time to attempt to do things with my team as well, which as discussed in previous portfolios, can sometimes be a struggle with the lack of communication. But the teamworking has never been better, with most of the regular team members keeping in contact, reviewing code in a timely manner and being supportive of one another. This isn't something I could have predicted a number of weeks ago and I think it shows that this method of teaching works for those who are willing to commit to it.

Overall, I am very happy with the results of not only this week but the entire project. I am grateful that I was able to participate in it. I have developed a lot as a coder and a team member. I am proud of the work I have achieved, the skills I have developed and the team I had for this project and I am sad that I won't have anything to submit going forward. However, it is my intention to continue working on this project regardless as I can see the potential in it.