Project Work - Week 4

This portfolio will be slightly different from my previous entries. This week, after having a discussion with my lecturer, I made the decision to pursue a task that I felt was not only hindering my own learning but also caused problems each week for myself and my team.

The MVVM (Model View View Model) design is a pattern which involves a separation of the UI and the program logic. This improves literally everything within the software, examples being, maintainability, testability, reusability and so on.

For a number of weeks, due to lacking this separation, some tasks haven't been as easy to implement, for example, mocking. To mock a database, there needs to be a separation, however, in the project for the last few weeks, the database was being used within the view model class. I discovered this after attempting to mock for a number of portfolio entries but to fail each time.

So this week, I will follow the requested tasks in this portfolio just like the previous weeks, however, I will be adding more information related to my journey separating the logic and UI elements.

I will be doing the following:

- 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.

Descriptive Summary

The task I chose I have briefly discussed in the opening to this document. I chose and will continue to work on separating the logic and UI elements this week and next. The task was too large to achieve in one work week whilst juggling other commitments, however, I believe that I can achieve this by the end of my next portfolio entry.

I started by asking my team if they would be okay with me altering the core of this project. I met no disapproval and quickly made myself an issue on GitHub and assigned myself to it.

However, after starting, i realised the task isn't nearly as simple as it sounds. As stated prior, I have limited experience in C#, coupled with a lot of experience in a different design pattern, this has lead me down a lot of wrong paths.

My team, previously to this, had used a lot of strange or non explanatory naming conventions. I didn't pick up on this until recently, as I was unsure how it was supposed to be in C#. This contributed to my lack of understanding, however, I have been making strides in understanding in the last couple of weeks, hence why I believe I am up to the task of fixing these issue.

The largest issue was related to how the entire project was set up. It had no "Model", only view and view models. The view model was in direct contact with the database and contained no separation of logic at all. To

start fixing this, I removed the database from all of the view models and created a "Model" page. I started moving the logic into the model, mending issues such as bindings and missing directives along the way. After I completed this, I had a model which communicated with the database and with the view model. However, after this, I was told that this isn't true MVVM by a microsoft developer. In fact, the way I should do it is not the code behind way, but instead, the code behind would simply initialize the class and the view models would be separated further into a "ViewModel" folder. This is where issues started arising, issues with binding to classes which contained a dependency injection for example. The solution to this eludes me even now. However, given enough time, I will solve this problem, but as of right now I am unable to due to deadlines.

Now the issue is what to include in this portfolio. As most of my time this week was spent separating pages and bug fixing, I don't have an incredible amount of code which I am able to show which would demonstrate this. So instead, I will be showing the methods I created within the model which is working at this time.

Code Commentary

The first piece of code this week is as follows:

```
/// <summary>
/// Sends a request to the database to get a list of all items
/// </summary>
/// <returns></returns>
public async Task<List<MemberModel>> getListFromDBAync()
{
    var teamMemberDB = new TeamMemberDB();

    List<MemberModel> list = await teamMemberDB.GetItemsAsync();

    return list;
}
```

This code is not perfect. I can see issues with it myself which I don't like, however, it is functional. It has good comments, suitable for doxygen, it does one job, which follows the single responsibility principle, its simple and easy to understand which follows KISS and it contains no repeated code, which follows YAGNI.

Where the fault lies is that I want to be able to use dependency injection in the class for the database. It would be repeated code if every method needs to make a TeamMemberDB to be able to interact with the database, which will be a problem in the future if this is left unchecked.

The second piece of code this week is as follows:

```
/// <summary>
/// Saves a user in the database
/// </summary>
/// <param name="user"></param>
/// <returns></returns>
public async Task SaveUser(MemberModel user)
{
    var teamDB = new TeamMemberDB();
```

```
    try
    {
        Debug.WriteLine("Step 2: Saving user with Name: " + user.Name);
        await teamDB.SaveItemAsync(user);
        Debug.WriteLine("Step 2: User saved successfully.");
    }
    catch (Exception ex)
    {
        Debug.WriteLine("Step 2: Error while saving user: " + ex.Message);
    }
}
```

This piece of code is slightly more complex than the previous. It is a task which will take a MemberModel object and pass it to the database to be saved. It attempts to do this in a try catch block, first printing a message with the user name, performing the task and then displaying a follow up message after this is complete. If it fails, it will display an error message.

The try catch and debug code were written as so many changes were being made, I wanted ways to track exactly what information was being passed around the program.

The code again has good doxygen ready comments, with easy to understand variables, following good C# coding conventions. It isn't complicated, following KISS, it doesn't contain any code which it doesn't use, which follows YAGNI, it only performs one task following the single responsibility principle and finally it doesn't contain any repeated code, following DRY.

However, as discussed previous, this code could be in breach of DRY, as due to my inability to inject a dependency this week, I needed create my own teamDB inside the method. This is sub optimal.

Testing

This week, testing was more difficult as the program has been completely altered. I created some test code which worked until I started moving onto the next phase of separating the code behind to its own separate folder. I attmpted for numerous hours to get a working test for this task this week, using Moq and so on, however, it simply wouldn't work as I haven't finished with the separation of logic yet. For next portfolio, I intend to have 2 fully working tests, using a mock database, with full MVVM separation.

Code Review

This week, I looked at a fellow teammates code and found a few issues. I found that one of their method had two responsibilities. I found that they weren't using appropriate commenting and of course I found that there was no separation of logic between their view model and their model. Of course, the final issue is on me, and I didn't mention it in my review.

The teammate was receptive of my findings and corrected their code.

I wasn't able to get my code reviewed as it is entirely unfinished. It would simply contain far too many errors as it is a work in progress. I consulted with my team about this, and we agreed to leave it this week, and next week it would be reviewed fully when it was complete.

Reflection

This week has been full of ups and downs, with more ups than downs. I learned an incredible amount of information relating to C#, clean code, separation of logic, the MVVM design pattern and many other things.

I haven't been able to fully process all of this information as of yet, and it is still a learning experience. I went into this task thinking it would be difficult and time consuming, and so far I am correct. However, I believe that pushing myself in this direction can only result in positive things. I learned more this week than I normally do in numerous weeks.

Many, many issues arose over the course of my coding journey this week. Some took hours to solve. This isn't time I have at the minute due to other class commitments to fully envelope myself into coding sadly. I believe if I didn't need to put my time into these other classes, I could fully resolve all my issues this week. But this makes me want to try even harder in the limited time I will get next week.

I believe that my team overall is functioning incredibly well. I am very happy with the work they do and sad that it will end soon. Week in week out this team improves in great strides.

I agreed with the lecturer prior that I could work fully on this task and focus a little less on the other things required, thus, I haven't looked into everything yet this week. I will however ensure that I am fully caught up by next week, concerning persona's and security.

Finally, I hope that I can get a little bit of leniency this week with my portfolio, or, perhaps, graded on this one and the next one combined. As the task I chose this week was of such a scale, this portfolio has suffered. Testing of course couldn't be included as there aren't any testable methods yet. I am not able to fully comment on everything mentioned in the brief this week as again, I have been fully focused on realising this task.