Code Review

The purpose of this document is to summarize the comments of our code review spreadsheet into one concise format. Specific review comments for each file of our project can be found below. All pages were reviewed based on the following checklist taken from New York University suggested by Tim Maciag that can be found here:

Code review spreadsheet: https://bit.ly/3b0m0UB
Checklist: https://nyu-cds.github.io/effective-code-reviews/03-checklist/

General

I. Does the code work?

Yes, the entire project compiles and is logically sound based on our requirements. Unfortunately not all the forms have error checking fully implemented so an error on some pages will not inform the user if they did not properly fill a form. The navigation bar icons are sourced using a direct URL to AWS S3 and their behaviour is inconsistent.

II. Is all the code easily understood?

Yes, generally the code base is easy to follow and understand. The only obvious readability issues with our program are the naming conventions of the files. The file structure is well organized to follow the three user flows with a fourth set of files for shared files between all users.

III. Does it conform to your agreed coding conventions?

Yes, all front end code generally follows a similar layout with consistent braces, line formats, and comments. There is needed improvement in terms of using the CSS file. There are several redundant classes and most elements do not follow a proper hierarchical system of classes, IDs, elements, etc. Any future development should include a cleaned version of the existing file.

A convention we followed was the MVVM design pattern and most of the code present follows that pattern. There are a couple instances where this pattern had to be violated since there was no alternative way to accomplish the task. An example of this is the ListView background colors. Xamarin currently does not have an easy way to change the color of the ListView when a user selects an item. Therefore, as a workaround some code had to be written in the XAML.cs files which violates the rules of MVVM.

IV. Is there any redundant or duplicate code?

We tried to follow the DRY principle as best as possible. As a result if there was code that was being repeated more than twice we tried to make that code into a function or a class that can be reused. For example, for the Admin view the

ViewNewOrder and ViewActivePastOrder classes share the same fields that need to be displayed on the page. Instead of defining bindings in both the classes, we moved the bindings to the base class which both these files inherit and therefore they can share the bindings. With that being said, there are a couple of instances with redundant code especially in BaseCustomerOrderHistoryViewModel, BaseAdminOrderViewModel and BaseCarePartnerOrderViewModel. The functions defined in these classes are slightly different but share most of the same code so improvements can be made to avoid duplicate code.

There were several constant variables that were used throughout the application. To avoid any duplicate code a Settings and Constants file was created which hosts all the constant variables. These files are referenced whenever a constant variable needs to be used. This rule is violated when it comes to using images in the application since the image link is directly used in the file. In a future improvement, the image links will also be moved to the Constants file.

V. Is the code as modular as possible?

Yes, since we followed the MVVM design pattern we ended up creating small self contained files that have clearly defined roles. All the ViewModel files relate to a View and only do tasks that are relevant to that view.

VI. Can any global variables be replaced?

There are no global variables used in the application. There are instances where a base class defines variables to be used in the inherited classes but those variables are defined as ‘protected’ and are therefore only accessible by classes that inherit the base class.

VII. Is there any commented out code?

There are a couple of instances of commented out code which is code that is used for testing purposes. Since further work on this application is still to be done, we prefer the commented out code to be present for the time being.

VIII. Do loops have a set length and correct termination conditions?

Yes, all loops terminate without extra iterations or additional unneeded content. Each loop runs optimally to decrease the app response time. Most of the loops in the application iterate over an object so they all have set termination conditions. Special care was taken not to include any nested loops which might adversely affect the application.

IX. Do the names used in the program convey intent?

Generally yes, all functions, variables, and files are properly named and convey the intent that they need. As mentioned above, some files have inconsistent

names due to reworks the user flow part way through the project. Future development will include ensuring all files are consistently named for their intent.

Performance

I. **Are there any obvious optimizations that will improve performance?**

There are no obvious optimizations that are evident at the moment. Some of the functions where data is retrieved from the database do end up running very long so a future improvement might be to shorten those functions and divide up their responsibilities.

II. **Can any of the code be replaced with library or built-in functions?**

Libraries and built-in functions were used wherever possible. Examples include the AWS Cognito library to handle user sign up and login and the FluentValidation library to handle data validation.

III. **Can any logging or debugging code be removed?**

There is some logging code present in the application. Since further work is to be done on the app, it is preferred to keep the logging code present until the app is closer to completion.

Security

I. **Are all data inputs checked and encoded?**

No, due to time constraints, data input checking was not implemented on all the pages. There are pages such as the Register page where the input is checked but on pages like OrderDetails there is no data checking at the moment. Robust data checking is a future improvement.

II. **Where third-party utilities are used, are returning errors being caught?**

Yes, try catch statements are used when third party libraries are used. Example is when we sign up a user using the AWS Cognito library. The whole code is wrapped in a try catch statement which outputs any errors if they occur.

III. **Are output values checked and encoded?**

There was not a need to check output values since all data displayed is retrieved from the database. If there was an error in the data in the first place then it either would have been caught and showed the error to the user or a corresponding database entry for that data would not have been created.

IV. **Are invalid parameter values handled?**

As mentioned above, robust data checking was not implemented due to time constraints. This is a future improvement.

Documentation

I. **Do comments exist and describe the intent of the code?**

Yes, as part of the code review process, each page was given a header comment as well as general function comments. Most files fully describe their purpose and their process. Some files are lacking specific inline comments to more specifically describe their intent

II. **Are all functions commented?**

Yes, every function in the code base has a header comment block to describe the content and process of the function. There are also comments throughout the pages indicating the features of each function and their associated variable.

III. **Is any unusual behavior or edge-case handling described?**

There are comments to explain when there is a complicated task happening or if there is a hack way of accomplishing a task

IV. **Is the use and function of third-party libraries documented?**

Functions that use third party libraries have comments explaining why the third party library is being used.

V. **Is there any incomplete code? If so, should it be removed or flagged with a suitable marker like 'TODO'?**

There are no areas that contain incomplete code. However, some pages are still included in the document that are not being used by our project. Some of these pages are left blank and others contain partially written code that will be continued upon at a later date. Likewise, some pages were made for systems not implemented in our MVP but will be used in the future therefore they have remained part of the project.

Testing

I. **Is the code testable?**

Yes, the code is testable. Because of the MVVM development of our project, each model can be tested separately and because of the modularity, multiple models can be tested together smoothly.

II. **Do tests exist, and are they comprehensive?**

Inside the code, there are no tests written. Most testing was done by a user maneuvering throughout the app using a build rendered on a mobile device or through Visual Studio's device emulator function. All code and the functions were tested at one point.

III. Do unit tests actually test that the code is performing the intended functionality?

Our code does not contain any unit tests. All unit testing has been done manually.

IV. Could any test code be replaced with the use of an existing API?

As mentioned previously, no test code was written for this project. All tests were manually done by the team