# Code Quality Review Report

SSE Capstone - University of Regina
Safety Toolbox
Winter 2024

Mackenzie Kot
Mikayla Peterson

Table of Contents

# Code Quality Review Report

1. Overall Comments

Since we used .NET MAUI, there is some code that is automatically generated by the framework. Since this code is automatically generated, we will not be reviewing the quality of this code since we have no real control over it. Since .NET MAUI uses C# for all backend code, we can apply principles of object-oriented development as well as SOLID. Every single page and sub-page in the application has a single purpose, our overall code structure adheres to the Single Responsibility Principle. The structure of the dashboard and how new tabs can be added to pre-existing pages, our overall application structure adheres to the Open-Closed Principle. Comments were added to lines of code that required an explanation of why they were written the way they were. Other than that, the function names and variable names are self-explanatory enough to not require explanation comments.

2. Login, Signup, and Dashboard

### 1.1 Understandability

The Dashboard files are some of the simplest code in the application. Variables are appropriately named, and you can tell what the code is doing by looking at it, and there are a couple of clarifying comments as well. The Signup files are also quite understandable just by looking at it. The heavy lifting of password salting and hashing is provided by the Password class. MainPage is the login page, and it too is fairly straightforward.

### 1.2 Coding Standards

Variable and function naming standards are mostly intact, with camelCase notation for private variables and functions, and CamelCase for public ones. There are a few outliers to this rule that weren't caught. It also seems that the functions that were auto generated for events (such as clicking a specific button for example) are incorrectly using CamelCase when they are private functions, and this wasn't noticed until now. There are appropriate try catch statements in the case of a lost connection with the SQL database for example.

### 1.3 Duplication

There is a fair amount of duplication of code between these files, namely when it comes to the SQL queries. A possible improvement for this would have been to create a class that handles SQL queries, with separate functions for handling insert statements, select statements, and update statements. The main reason we didn't do this is that we would have to figure out how to handle reading the different results from the queries and the different data types involved.

### 1.4 Debug-ability

These files are easy to debug, it is as simple as making an account and ensuring the user can log in with that account. Different account types were also tested, and it is easy to tell if something goes wrong.

1.5 Function/Class Size

The sizes of the functions and classes are suitable, and for the most part there isn't a section of code that is doing more than it should. One function that could stand to be broken up is the saveSignupInformation() function in the Signup class, which has 3 SQL queries in a row. This function could be broken into a few smaller ones.

3. Settings

2.1 Understandability

The code for Settings is quite simple and easy to follow just by looking at it. Variables are all named in a way that makes sense.

2.2 Coding Standards

The previous issue of auto generated functions not following the proper camelCase vs CamelCase is present here again.

There is some code preserved in comments belonging to a feature planned for future work. We plan to come back to it after this semester. If this wasn't the case we would have removed the dead code.

2.3 Duplication

There is no code duplication anywhere within Settings or between Settings and somewhere else.

2.4 Debug-ability

This code, being easy to follow, is very easily debuggable. The tester simply has to check if the values on the page are being saved appropriately.

2.5 Function/Class Size

Everything to do with the Settings page is all from a single class. This class isn't doing too much work on its own, and functions are an appropriate length.

4. Toolbox Talk

3.1 Understandability

There are a few main classes running the pages under Toolbox Talk, namely: Notes, TopicIdeas, Attendance, and Toolbox Talk. All of them are straightforward and contain appropriately named variables.

3.2 Coding Standards

As is the trend with the rest of the application, auto generated names for private functions are following CamelCase formatting instead of camelCase.

3.3 Duplication

The previous point of SQL queries being duplicated code applies to this section of the app as well. To reiterate, this could have been improved by creating a SQL class with the ability to handle different insert, update, or select statements with or without parameters. Other than this, there is no repeated code.

3.4 Debug-ability

The code is easy to follow and user cases are easy to outline, making this section of the app easy to debug as well.

3.5 Function/Class Size

Most functions are a decent size, the only exception to this may be the RadioChanged() event in the Attendance class. Having a SQL statement in a function is a sure way to make it longer, but this function has two, an Insert statement followed by an Update statement in the case that an entry for the primary key (in this case it is the employee and the date) already exists. This makes that function fairly longer than the others, so it could have been split into two functions: one for the initial handling of the radio button value and a second for the SQL queries.

## 5. Certifications

### 5.1. Understandability

As with the other pages in the application, each page has it's own associated front end and back end code files. This makes it easier to understand and mentally separate the application's appearance from it's behaviours. Functions and variables are named appropriately and are descriptive enough to understand what a function is doing.

### 5.2. Coding Standards

Similarly to the rest of the application, auto generated names for private functions are following CamelCase formatting instead of camelCase.

### 5.3. Duplication

As discussed previously, the only duplicated code in the backend is the SQL query code. There is some duplication within the front end of the main Certifications page. This is due to the way we used data templates to format the rows for the table for expiring, expiring soon, and unexpired certifications. We could've used style triggers for the colours instead, but refactoring the data templates was lower on our priority list.

### 5.4. Debug-ability

Since there are no god functions and the functions all have a single responsibility, it is easy to find the root cause of a problem that occurs. The code is also easy to follow and it is easy to define the use cases for this portion of the application.

### 5.5. Function/Class Size

In terms of class size for the both the back end and front end code, all the files are 200 lines or less. Many of the helper classes, such as the file viewer are 100 or even 50 lines or less. None of the files do not have responsibilities outside what they're supposed to be doing; there are no god functions. The longest functions are the ones where there are SQL statements.

6. Library

6.1. Understandability

Similarily to the other pages in the application, each subpage has it’s own associated front end and back end code files. This separation makes it easier to understand and separate the application’s appearance from it’s behaviours. Functions and variables are named appropriately and are descriptive enough to understand what a function is doing.

6.2. Coding Standards

As is the trend with the rest of the application code, auto generated names for private functions are following CamelCase formatting instead of camelCase.

6.3. Duplication

There are similarities in the code for both the Work Procedures and the Training Documents code because they are functionally the exact same, but the documents they store have different purposes. Some code is repeated within these two pages but with minor differences. This code is repeated since each page gets its own front end and back end code file.

6.4. Debug-ability

Since there are no god functions and the functions all have a single responsibility, it is easy to find the root cause of a problem that occurs. The code is also easy to follow and it is easy to define the use cases for this portion of the application.

6.5. Function/Class Size

Similarly to the front end and back end of the certification pages, all the files are less than 200 lines. There are some shared helper classes between some of the subpages, and they are all 100 lines or less. None of the classes are doing things outside of their scope of responsibilities; there are no god functions or classes. As with the other SQL related code in the application, these functions tend to be the longest ones.