

COMP3404

Design documentation



Ethan Kirton

21007400

Contents

[Introduction 2](#_Toc153998272)

[Problem analysis 2](#_Toc153998273)

[Problem analysis use case 3](#_Toc153998274)

[Other useful information about the problem 3](#_Toc153998275)

[OO Software Design 4](#_Toc153998276)

[Test Strategy 4](#_Toc153998277)

[Quality evaluation 5](#_Toc153998278)

# Introduction

In this report there is a solution which is made for a company called ‘PetSimsRUs’ in which they have asked for an asset view tool for one of their aquarium simulation products. Through this report there will be use cases and other forms of diagrams to support the structure of the program with an explanation as to why they are used to create the solution.

Attached to this documentation will be the solution to the problem.

# Problem analysis

A diagram of a diagram of a diagram

Description automatically generated with medium confidenceThe client is requiring the solution to be an asset view tool for their ‘PetSimsRUs’ product. To make a success of this project CRC cards are used to show how the program will flow.

The project will open when the program class has started up, ‘Program’ talks to the ‘Fishyform’ class and opens the form. From here the program can flow move between the other classes as required for the image viewing. To view the images a user will select a button on the first form that has an event handler that asks the servers to locate an image (the IFileServer will carry out this operation) and to prepare to draw the image (the ResourceFileServer will carry out this operation). Servers will be used to store and manage images for the GUI, which scales and prepares the image to be viewed by the user. The second form ‘Fishes’ will be the GUI module where the images are printed for the user to see. There will be functional paths between the two forms to benefit user experience. The servers will be unavailable for the user to see and be running in the background. This is an important feature because it means the user will be able to move laterally through the whole program and view every image while only having to start the program once.

## Problem analysis use case

|  |  |
| --- | --- |
| **Use case 1** | Image select. |
| **Actor** | User |
| **Use Case Overview** | User will select a button related to viewing a specific image that is predefined by the button. |
| **Preconditions** | * Program has started. * User is real. * User wants to see an image. |
| **Triggers** | The event handlers on the buttons of ‘Fishyform’. |
| **Main success Scenario** | The button leads the user to see their image displayed on a new form. |
| **Alternative path** | The button is unresponsive, and an exception is thrown. |

|  |  |
| --- | --- |
| **Use case 2** | Delete Button |
| **Actor** | User |
| **Use Case Overview** | The X button at the top right of the forms either to close the form or to quit the program |
| **Preconditions** | * User has opened the form. * User would like to exit the current form. |
| **Triggers** | Event handlers listening for the user to click of the close buttons. |
| **Main success Scenario** | The current form or whole program closes/quits. |
| **Alternative path** | The form doesn’t quit, and the user is stuck on their screen. |

## Other useful information about the problem

To make a successful solution to the given problem, the system requires that the program consists of a server and GUI model which has been predefined using the CRC cards above, the CRC model also explains how the programs’ structure will allow the classes to interact with each other to fulfil their role. Class names have been considered as definitions of their roles in the program, this will make modifying code in future simpler and will mean that collaborators will be able to understand what will have already been programmed.

# OO Software Design

A diagram of a software

Description automatically generated

# Test Strategy

To test the program works there are two-unit tests in a Cs file called ‘ServerUnitTest.cs’, in the file there are test for ‘NoImageFound()’ and ‘ImageFound()’. In the ‘NoImageFound()’ test, it is told to look for the image that the user has selected in the Fishy form. It does this by testing the pathway ‘MediaServer.ResourceFileServer()’ and reporting what it found to the method ‘LoadImage()’. A computer screen with text

Description automatically generated If the Image isn’t located, then it returns “This Fish Doesn’t Exist”. This is a crucial test for the program since the images are the most important attribute of this system. In order for the soluti0on to meet the requirements the images must be found in the ‘MediaServer.ResourceFileServer()’ address due to the brief specifying the use of servers to store the images. So this test works as two tests for the solution.

The other unit test is a test to see whether the compilation was successful and that the solution is correctly addressed since it tests that the correct image was loaded into the GUI. A computer code on a black background

Description automatically generated

It does this by again following the route ‘MediaServer.ResourceFileServer()’ to locate the image with the name specified in ‘LoadImage()’ which in this case is ‘OrangeFish’, If the tag matches an image in the ‘ResourceFileServer()’ Then the image is drawn to the form using ‘System.Drawing.Bitmap’.

# Quality evaluation

In the program there is evidence of the Single Responsibility protocol when looking at the classes structure, each class is made with for a singular main responsibility. By using names that are related to what the class does this makes it easy to locate which part of code carries out which function. The classes work together as building blocks to create the whole program. The form ‘Fishes’ has multiple methods such as SetImage(), ResizeImage() and DeleteBtn\_click(). These classes are all related as in they all work together to produce the functionality behind the ‘Fishes’ form. Each of the methods has a single responsibility in its function because its good practice to give methods a single responsibility and a name that relates to that responsibility. The classes do not rely on each other to produce a result. Which shows loose coupling in the software.

The Server files ‘ResourceFileServer()’ and ‘IFileServer()’ are an example of where there is high cohesion between classes since ResourceFileServer hold a method called LoadImage() wherein it checks the resources in the server to find the tag that the user has selected in Fishyform. This method is a public method so when the IFileServer calls the LoadImage() method the ResourceFileServer carries out its function to produce the image on the form for the user. Since the IFileServer is also an interface it clearly represents the Interface segregation Principle as it is a small interface whose only job is to call the method LoadImage() from the ResourceFileServer().