Computer Science 3307A - Object-Oriented Design and Analysis, Fall 2021
University of Western Ontario, London, Ontario, N6A 3K7

Magic Mirror: Postmortem Report

Group 20:

Nathan Dinatale
Darwin Shiyi Liao
Peter Nicolaas Meijer
Nolan Morris
Yifei Zhang

Instructor: Mike Katchabaw

Date: December, 8th, 2021

1. Project Summary

The Magic Mirror consists of both a live video feed, to simulate a mirror surface, and an information overlay that contains various widgets, all of which provide some kind of information to the user. The main goal we had with this project was to provide an interface that the user could interact with to provide more detailed information should they require it, while simultaneously providing much of the useful information they would need directly on the screen by default. We decided that, in order to not clutter the screen, it would be sufficient to include the following widgets: A simple news scroller that could display the current headlines and would allow the user to click on them to show the full story in their default web browser; A weather widget that could show the current temperature, the current weather conditions and the weather conditions for the next 5 days; A simple calendar where events can be added and displayed on the screen; A clock displaying both the current date and time; A simple user system that would allow the user to login to a gmail service to scroll through their emails for the current day, providing they give a valid gmail username and password.

It was decided that each of the above mentioned widgets would be extensions of the provided QWidget class from the Qt 5 library as that would allow for reduced coupling between the objects and would allow for cohesiveness when designing and making classes, as each widget would be developed in it's own folder and then imported into the bootstrapper. The bootstrapper would simply instantiate the developed widgets, add them to a layout, using the provided classes from Qt 5, and then draw them to the screen, allowing for very minimal coupling between widgets.

2. Key Accomplishments

2.1 The Design

The modular design worked wonders for the project and was even better than predicted. Each of the modules were able to be worked on by a single individual with simple specifications of what methods would need to be exposed in order to that widget working with the interface in the desired manner. Once each of the modules was developed and pushed to the remote repository, it could be dropped right into the project and, most times, only a few minor changes had to be made to things like file paths to libraries or external resources that need to be used by the system. The design was very successful in decoupling the modules from one another, allowing for most of the time to be spent fixing bugs in those modules instead of fixing bugs in the bootstrapper.

The decision to make each of the modules extensions of the QWidget class was a huge success as it allowed us to use the provided layouts offered from the Qt 5 library to store and position each of the widgets easily and effectively. It also allowed for easier development of the overlay that is placed over the live camera feed. Since each widget could be added into a layout, it was easy to add that layout to the graphic scene, allowing for it to be made transparent while also having it be drawn over the live camera feed, providing the exact functionality desired from the system without much hassle or debugging.

2.2 The Camera Feed

The live video feed from a connected webcam also went relatively smoothly. Qt 5 has a built-in framework for getting a camera feed from a connected usb camera which was particularly useful. It was simple to encapsulate the camera operations in a single widget that stores the camera feed in a graphic view item that can then be added to the scene to project the video feed to the main window of the application. Initially, this process was thought to be the toughest part of the assignment, however, the provided classes from Qt 5 made the process much more straightforward than could have ever been expected.

Despite a few hiccups along the way with sizing the video feed, there were hardly any errors in implementing the system. In addition to this, if there is no camera feed that is connected to the computer, the camera feed will simply not display, leaving a white background with all of the widgets drawn on top of it. This means that, even if the user does not want to use a webcam or if the user does not have a webcam, they can still utilize all of the widgets the application has to offer.

2.3 News Widget

The News Widget was based off of a connection to a small third-party API. Originally, we had thought that we would need to use some sort of C++ library outside of Qt to help us create a network request, but luckily, Qt included many classes that greatly helped in setting up the connection to the API, specifically, QNetworkManager, QNetworkRequest, and QNetworkReply. All of these were extremely well documented and once we had learned how to properly implement them, it was relatively easy to send a get request to the API and any other URL that we needed to access.

Once called, the API that we were using returned a JSON file, which again, was processed by things included in Qt. The QJson classes allowed for easy reformatting and filtering of some unneeded information that was retrieved from the API. Since this data was to be updated constantly over the course of the day/runtime of the program, We decided that there wasn't a need to permanently store the information in some JSON file, but we could simply create an ADT that would contain the information and contained methods that returned whichever value that was requested.

2.4 User System

The user system created was a key accomplishment because, even as a standalone module, it is very feature rich and complex to implement. The user system required many features that needed additional implementation such as (1) persistence, (2) permissions.

A key feature of the user system was its ability to be persistent over time. We created a file management system that stored user account information in the /useraccounts/ folder of the project, which then acted as a database for the next time the program was run. We decided to store this user data as JSON files to simulate real world web applications which would typically send data via JSON to a true database storage such as SQL or mongoDB, etc. We used an external library (jsoncpp) to process the files which took a long to configure and integrate with C++ so this was a key accomplishment. The library enabled us to work with JSON objects in C++ and read and write to our filesystem storage.

Another feature involved was the ability of account creation - we wanted this ability to be limited to admin users. As such, each user was classified as an admin or not upon creation, and this information was stored and utilized within the program to open up certain features, and shield account creation features from standard users.

2.5 GMail Integration

2.6 Calendar and Time

For the time module, there were already many useful classes and functions in Qt that helped implement the functionalities of the clock. These included: getting current time, changing the timezone, getting the time to strings in any desired format, and many more. Given all these libraries, the time function and clock widget was a great success.

As for the calendar part, Qt also has everything needed to accomplish the task from QCalendarWidget for calendar UI to QDateTime for date and time as well as various data structures to store information. Without Qt libraries, many of these partial functions would have taken much longer to implement from scratch.

2.7 Weather Widget

Qt provided many tools to make the development of the Weather Widget a smooth experience. The initial hurdles were learning how to use the relevant Qt classes to fetch the response for the weather API and parse it. The weather API we used, openweatherAPI, provided all the necessary information but sometimes not in the most readily available fashion, so some work had to be done in order to extract it and format it for the GUI. Luckily the tools Qt provided for this were relatively easy to work with. So in the end we were able to create a Weather module that fully encapsulated all the fetching, parsing and analysis required, providing the values and images that needed to be displayed to the main window of the program.

3. Key Problem Areas

3.1 The Camera Feed

Despite the success in designing and implementing the camera widget, there were some serious problems that came up. The most significant being the frame rate of the video feed on the background. Since everything was developed on a LINUX virtual machine, we were unable to test to determine if this is a hardware issue due to the virtual machine, or if this is due to the way we implemented the camera. After extensive research, we determined the most likely cause is the virtual machine. On one system, where the problem was the worst, they had their virtual machine stored on an HDD due to storage constraints on their system. It is believed that this is the most likely cause for the poor frame rate as others who have used the camera functionality from Qt 5 have not reported such issues and videos demonstrating projects that use it have more than acceptable frame rates.

Given more time and more resources, it would be good to try to port this to the native OS used in the systems of the developers to see if the issue persists. If it is a hardware issue,

as hypothesised, it would be good to research ways to mitigate the impact it has on the system to allow for users with more lower-end hardware to run the project without issue.

3.2 News Widget

The news widget encountered problems in nearly every step of the project after the initial concept idea and user story creation. Originally, we had planned on using Google's API for every portion of this program, ie. Google Mail, Google Calendar, Google News, etc. However, Once we had finished the user stories, and began starting on our individual portions, it was quickly realized that the Google News API had been depreciated over 10 years ago. This led to the need to switch APIs to a new option. There were many different available replacements for Google's API, like Microsoft Azure's Bing News, but many of the options required either a paid account, or did not seem well documented. In the end, we had settled on gnews.io's free API. Additionally, this switch made it so that some of the options initially requested in the user stories (such as search by tags) could not be implemented.

On top of this switch, it had turned out that using a gnews.io was not the most reliable API, but by the time we had learned this, the project was too far in to make another switch. While it did do its job when it worked, the API just simply did not wish to return a proper request at times, which led to many days of bug testing and double guessing to try and fix this issue. In the end, and after nearly exhausting all of our options, we simply decided to not show the news if the request had failed, and retry again in the next update timeframe.

Additionally, when testing out the API, sometimes certain news websites would have a HTTPS protocol instead of an HTTP protocol. This had originally been an issue since when the code was first being written, there was a streak where no website was using HTTPS, so the error did not occur, and we had not known that this type of error existed. Midway through the development of the News Widget, it turned out that our initial installation of Qt did not include OpenSSL, so it could not process HTTPS requests.

3.3 User System

A key problem of the user system was smoothly integrating data saving requirements with the other widgets of the mirror. The most notable integration requirements was with the GMail system. Since we wanted a user's inbox to be configured to their account and saved such that when they log out and log in their inbox would still work, the GMail system had to be integrated with the user system. This required some redoing of the user system code midway through development to accommodate this. While we accept that this is a reality of agile development, at the time it was a problem area that interrupted our work plan.

3.4 GMail Integration

Some of the main problem areas arose in the GMail integration system. Google offers APIs, however these are tailored to be accessed by specific web IP addresses (they are meant for design with web applications). Additionally, they require a developer console app registration, OAUTH authentication setup, and various other configuration headaches for working with C++. We had to pivot this GMail system to work instead of through the API to function through IMAP - essentially we had to recreate an IMAP server in C++ that would allow us to access inboxes. We spent a lot of time exploring many different libraries, using

SMTP to test reading inboxes, analyzing POP3 vs other protocols, eventually settling on mailio (a mail server C++ library). Changing this implementation did allow for additional features, such as the ability to connect to any inbox (not only GMail, using different IMAP servers we can connect to outlook, etc.). The current implementation was hard specified to user GMail IMAP servers to be in line with our user stories, but it is easily modifiable to work with other mail service providers.

3.5 Calendar and Time

There isn't anything special that we encountered when implementing time and clock widgets, everything went relatively smooth. However, when implementing the calendar functions, there were a few things that required us to make decisions in terms of the data structure that holds the relevant dates and events information. Each data structure has its own pros and cons, and thus, choosing one was difficult. Also it was our first time creating a large project with C++, and it is quite different from other programming languages we've used throughout university to this point, especially the use of pointers and memory management. That was also one of the major obstacles that we encountered when writing the code for this class, many errors and bugs were due to inappropriate use of pointers. There were also a few things that didn't go perfectly when using LINUX and GNU, it was hard to debug in linux since the terminal will would sometimes be filled with irrelevant information when testing the code, and we had to often pick out the information that will help fix the bugs, after this project I feel a lot more confident locating problems in my codes. It was also my first time using bitbucket to manage my workflow, I had to search a lot online for useful commands to push and pull the works from remote directories.

4. Lessons Learned

The most important lesson learned throughout the course of this project was how to effectively work as a group and how essential good coding practices and design are to a successful project. The design of the project definitely contributed much to its success and if we hadn't designed the project with low coupling in mind, the project would have been much tougher to implement. Good communication between the relevant group members was also key in getting the entire thing working well. Since each person became essentially an expert on their own module, it made it super easy to integrate everything when we worked together.

In addition to this, all of us had never worked with C++, Qt, or git before this as none of the projects we've done in school nor our own have required us to do so. Learning these frameworks definitely provides a lot of utility and is valuable experience to have going forward. Moreover, getting hands-on experience with tools such as git and Jira have given us, not only valuable experience, but insight into why they are so widely used in the industry.

Upon reflection, if we were to start over again with the experience we had now, we would try to find a more reliable and stable API to use for news as well as a better, more flexible one, for the email system. This would ensure that the News API works much more often then it does and it would allow for more flexibility in what kind of email could be used to connect to, and display, the mailbox. Moreover, I think we would all have liked to flesh out the user system more as it could have allowed for things like a persistent calendar that is

accustomed to each user and perhaps could have allowed for user settings for the mirror to be saved as well. This would allow the configuration of the mirror to be set differently for each user and would allow for more customization on the users end. All that being said, we all would keep the design that we chose at the beginning due to how flexible and easy it was to work with throughout the project. As mentioned above, the design contributed heavily to the success of the project and it's something that none of us would change if we were to start over from scratch.

Finally, one of the best practices we all learned throughout this project is the importance of communication between the group members. There were certain instances where requirements for integrating modules with the main GUI were not clearly specified to the person working on the module, resulting in extra work that needed to be done that could have been avoided had things been communicated more effectively from the start. That being said, the regular meetings that we had during the development process were very effective at communicating general issues with particular modules, bugs that could be found in the code, or just ideas that could make the implementation smoother or more efficient. These meetings helped to keep the project on track and focused throughout the process.