COMPSCI 3307A - Hue Light Group Project

Final Deliverable: Project Retrospective

Prepared By:	***Prepared For:***	***Date Submitted:***
Anthony Tran (*atran94*), Omar Abdel-Qader (*oabdelqa*), Usant Kajendirarajah (*ukajendi*), Zhengyang Pan (*zpan45*), Jacob Fryer (*jfryer6*)	Professor Mike Katchabaw COMPSCI 3307A: *Object-Oriented Design and Analysis* Western University	Friday, December 08, 2017

Project Summary

The focus of the project was creating an interface for connecting to and interacting with a Philips Hue light system through an Internet browser. To accomplish this, we developed an application using C++ and the Wt framework (version 3.3.8).

Our application allows a user to register an account and log in with their appropriate credentials to access their Hue bridges. We developed a simple database system to store, manage, and access these account credentials as well as the information of any Hue bridges associated with that user. This provided us a way of developing persistence of user data after the conclusion of a session.

Once logged in, the user is redirected to the landing page of our application, which provides a list of all Hue bridges associated with that user. While the user is on this screen, they can add and connect to a Hue bridge, or select a previously-connected bridge to edit its properties. Once the user has connected to a bridge, they can manage all the lights and groups associated with that particular bridge. All this functionality involved creating classes that modelled each component for the purposes of our application, as well as corresponding front end "manager" classes using Wt to provide a user interface for manipulating the settings of the associated lights, bridges, and groups.

Lastly, for our program to work with the physical lights, we implemented calls to the Hue API in our program's front end which update both the local objects in our application and the physical hardware. For testing, we used the Philips Hue emulator developed by GitHub user "SteveyO." We used Atlassian's Bitbucket and JIRA software for version control and project management, respectively.

Key Accomplishments

What went right?

We were successful in creating a functional database that persistently stored user account information (usernames, a hashed version of the user's password, and the first and last name of the user) and bridge information. Our solution initially was to rely on a single text file that would contain all registered users, but it quickly proved to become unwieldy when dealing with a variable number of bridges attached to a given user. To solve this problem, we turned to the Boost C++ library to create directories for each individual user, with separate text files for user information and bridge information to be serialized before the session was terminated, or the user signed out.

We succeeded in creating a functional (if barebones) user interface with Wt. The majority of the base functionality required by the project is in place, and the user can navigate around the application in a way that is fairly intuitive by following our established hierarchy.

That is, a user has a vector of associated bridges, a bridge has a vector of associated groups, and so on. When opening the application, the user is immediately prompted to log in before any action can be taken, and all other pages follow a logical sequence when being presented to the user.

Ultimately, although we struggled in several areas (see also, the section entitled "*What technical challenges did you face?*"), we successfully implemented the Hue API in our application and were able to provide an interface for the user to manipulate the state, hue, brightness, and saturation of the lights.

What worked well?

In storing persistent user data, our team did a good job of securing user passwords and usernames using a hash function provided in the standard C++ library. This ensured that we as developers did not have direct access to user passwords (in the event that our application's database were to be hacked in a real-world scenario, this would also prevent hackers from accessing confidential user data), and that individual user files could not be searched by username. Instead, the users can be searched by a hashed version of their username, which solves the issue of using special symbols in directory path names (e.g. the @ symbol, or a period).

We created an efficient database solution that did not require extensive use of data structures and search algorithms to save and search for user information. We made good use of file streams to read from and write to file, and designed well-encapsulated classes for each of our basic objects (Users, Bridges, Groups, Lights, etc).

What was found to be particularly useful?

We found the third-party plugin Postman to be quite helpful in the process of developing our project. Postman provided us a friendly GUI to build and send requests to the Hue emulator. With its help, we were able to test different API calls to see what parameters could be accepted by the emulator and what results are being returned. Because there is a discrepancy between Philips' official Hue API guidelines and the emulator's actual behaviour due to emulator's limited functionalities, we had to follow the guidelines carefully and test with the emulator from time to time. Postman saved us time and trouble during this process.

Git and Jira were incredibly useful tools in the development of this project. Using Git ensured that all group members could work remotely from any system and have all up-to-date project files. Jira allowed us to storyboard the design of our project and organize tasks to be done. Having these tools available made it so every team member was up to date with the project and ultimately, the use of Git and Jira helped us work as efficiently as possible in a group dynamic.

Additionally, using Boost C++ libraries was a huge help. Boost libraries contain many tools that make developing in C++ a little bit more accessible and familiar to those of us who are proficient in other languages like Java. This library made string manipulation and accessing the filesystem very efficient and straightforward, and contributed greatly in our database design.

What design decisions contributed to the success of the project?

Beginning with user stories and a UML diagram helped to direct our developmental efforts. Creating the user stories gave us many specific functional goals to work towards while making the UML diagram provided for us an overview of the structure of our project and helped to break it down into smaller, more manageable parts to develop.

Early on in the project, we separated development into two discrete folders: one for the front end development, and one for the database. By dividing the two, we were able to work build the project together without needing to shackle our progress to the other side of things. Development of the database was not contingent on progress on the front end, which allowed us to be slightly more expedient.

Key Problem Areas

What went wrong?

At one stage in the project, it was discovered that not all members of the group were using the same version of the Wt framework. This was a large issue because it meant that those members who were using the wrong version of Wt were unable to build the project in order to test their code. Because Wt's versions 3.3.8 and 4.0.0 are considerably different in terms of programming standardizations (such as the regular use of the "`auto`" keyword, the "`make_unique`" function, and the syntax of `#include` statement declarations), it was essential to revert to using version 3.3.8 at that stage in the project.

Perhaps unsurprisingly, time management was an issue in our group. We are all students, and unfortunately we often found difficulty in budgeting our time appropriately amidst other commitments and deadlines. Allocating appropriate amounts of time to specific tasks was challenging. We focused too much time on the wrong details. When we wrote out the code for each class, we realised there was much to change after looking at the HUE API as we did not include the appropriate attributes or structure our classes to reflect what the HUE Api needs.

Most of our group members have past experience with developing programs in other languages, such as Java or C. It was challenging for the group members to learn C++ while not much time was available to truly experiment and understand its strengths and features. It may have been better if more time was allotted, or a familiar language was chosen (for example, Java or Python). On top of general C++ struggles, our group found that the Wt Framework was not the most intuitive web development kit for beginners to C++. It did take more time to figure out how things worked with Wt.

In designing a backend solution for persistent user data, a lot of time was spent trying to come up with a technically elegant solution; one that would make use of previous knowledge of data structures, efficient search algorithms and C++ containers. In hindsight, it is clear that much of this time would have been better spent working with the API and Wt to produce a better and more efficient light-manipulation system. As developers, we were focused on the technical aspects of the solution when we should have asked ourselves if instead of a perfect solution, we could make do with a solution that was good enough. Ultimately, the decision was made to use the Boost C++ library to put the burden of searching for and storing users on the OS. While it is possible that a cleaner database solution could have been implemented, this was not the focus of the software project. In this sense, the focus shifted away from features that would enhance user experience to ways that we could showcase our development abilities.

What project processes didn't work well?

Using JIRA and Git in harmony was sometimes an afterthought. Over the course of the project, the majority of our group found ourselves developing on a single "development" branch in our Git repository and warning each other through a communal chat space whenever we were pushing an update. In practice, creating a new branch for each individual user story was expected. However, we generally made sure to update our user stories on JIRA whenever we were working on or had completed a given task once the story was in an acceptable state and passed our tests.

What technical challenges did you encounter?

One technical hurdle of particular note that our group needed to overcome was our initial implementation of our API calls. Our first pass at a solution was to use a "busy wait" when receiving API responses, and having a boolean variable be set in the class when the API response was received.

As we later found out, Wt's framework actually handles things asynchronously. Consequently our application would be stuck waiting until our supposed "time out" condition was met, causing the boolean variable to never be set. The API response would only be received after it was expected. Figuring out what was happening cost us around a day's worth of development time. Development was stalled while trying to figure out why the API would send the response, but the program would never be able to use the data due to our control flow.

Another technical challenge we encountered was in managing the user and bridge information in the database. Initially, the database was designed to save user and bridge information concurrently. So, every time bridge information was saved as a result of an add, update, or delete operation, user information was also re-written to file. When this happened, a user's hashed password would be rehashed, meaning that a user would not be able to log in a second time with the same password. Eventually, user and bridge information were split into separate files so that user information would only be stored at registration and would never be modified after that point. This solution worked for our purposes, since we were operating under the assumption that a user would not need to change their username or password.

What design decisions made it more difficult to succeed in your project?

We encountered a self-inflicted issue in our code related to our implementation of the different classes: Bridge, Group, and Lights. Our idea was to handle these classes in a hierarchy (as mentioned above), so that all lights would be associated with groups, all groups would be associated with a bridge, and all bridges associated with a user. We had some difficulty mirroring this hierarchy in the front end.

Another issue of structure came up as we came down to the wire. Ultimately, when it came time to connecting our back end database with the front end user interface, we encountered more problems than expected. In theory, the backend database and the frontend GUI should have been linked with little to no issue. As a team, we communicated constantly regarding the design of both aspects to ensure that they could be integrated in accordance with our overall design vision. In reality, since both of these were developed and tested independently, integrating them proved to be a more difficult and time-consuming task than we had anticipated.

What were the effects / impact of these problem areas?

When managing an individual Group or Light, to construct an API request from the corresponding widget, we needed to pass the Bridge down the hierarchy of widgets in order to construct a URL to send the request to get or modify the target Group or Light object. We found this to be a problematic design pattern that could potentially make the application slightly less cohesive as a piece of software.

As a consequence of our lack of foresight, we had to implement our database as a global variable. We recognize in hindsight that there are other, more effective design patterns (such as a singleton, for example) that could have been used to initialize our database on every session rather than as an unprotected global variable. However, due to time constraints, we felt we had to make a compromise in this aspect of our design rather than take the time to rework the implementation.

Our initial thought was that a few lines of code added to the front end should make the database feature fully functional. Upon linking and testing the two however, we realized that the result of all the independent work gave rise to a few more bugs than expected. Ultimately, this cost us more time trying to rework the database feature by rewriting some code and reformulating the logic behind how the system would work.

Because we did not utilize the synergy of JIRA and Git to its full extent, we found ourselves having to constantly check in with one another before pushing our changes to the common development branch. As well, it made it difficult to keep track of who was working on what task at what time, because the commit logs in Bitbucket were often insufficient in delineating the progress being made on one feature or another.

What corrective actions did you take to resolve the problems?

The solution to our Git issue was to create separate development branches for cleaning up the main development branch. Although it was not as effective as creating individual branches via JIRA, it was a satisfactory band-aid solution for our purposes when we were in "crunch mode" at the end of the project.

To solve our database problem, we had to spend a good period of time tracking down some bugs (such as double-hashing the password, or adding an additional bridge to the user as a residual piece of code during initial testing of the front end while it was being built separately that added a blank bridge for testing). A mantra that we adopted was "when in doubt, '`cout`'".

Lessons Learned

A key lesson that this project stresses is the importance of project management in the software development lifecycle (SDLC). An important takeaway is the efficacy of project planning. Tools used such as JIRA allowed for efficient project management and progress diagnosis. JIRA would allow us to assign levels of importance to specific tasks, and this allowed for developer's to direct their attention to functionality of the highest importance, rather than functionality of lower importance.

In addition, another approach we would take differently for next time is choosing a different software development paradigm. For this project, we essentially used a "waterfall method." Simply speaking, this method is where concepts are initiated, designed, constructed, tested, and maintained largely in one direction. Due to there being not much time spent on revisiting previous stages due to the constraints of the course, this software development model is not iterative and it is the least flexible of all the possible paradigms available.

For a successive attempt, perhaps it would be better to use Agile development. Agile development allows for adaptive planning, evolutionary development, early releases and continuous improvement. Adaptive planning would allow us to revisit requirements and adjust accordingly. Evolutionary development and early releases would allow us to release a simple and functional prototype of the core features and use this early release as the foundation for subsequent releases. With immediate feedback received and addressed, this would allow for continuous improvement. Agile development would allow for prioritization of development of core functionality that is important to the respective game in the prototype and then allow us to branch out to functionality of lesser importance in subsequent releases. It would have been better to take a more industry relevant approach – after all, it is prevalent in the industry for a reason.

All in all, this experience stressed the importance of several roles that come together to bring an idea to life in the software world. A project manager, quality assurance testers and software developers are all equally important in bringing an idea to life. A project manager is required to have the project moving along adequately. A quality assurance tester is required because often times developers become accustomed to (and biased toward) their creation and are unable to pick out bugs. Agile development paradigm would have allowed for testing along the way.

We realize that there is a lot more that can be done with Wt, and expanding on our experiences with our hindsight would allow us to refactor our code and complete the features that were left unfinished. It would be a good learning experience for us to return to this project outside of the context of this course. Prior to this experience, none of us had any experience with web development using C++. It has been an eye-opening experience having worked on this project, and it goes to show the vastness of the field of Computer Science. This project reiterated the importance of teamwork, internal structure, and communication within a group setting. If we were to attempt such an undertaking again in the future, enforcing a stricter schedule with internal deadlines in the style of an Agile sprint would be essential.

We feel that it would be interesting to work on a project like this again in the future. However, a caveat to this is that we feel we would rather work on a team project without the added burden of other courses competing for our attention. Coupled with a less condensed timeframe, a group project like the Hue application would be a much more appealing prospect.