**CS3307 Group Project Specification**

Version 1.0

**Fall Session 2017**

**1.  Overview**

The [Internet of Things (IoT)](https://en.wikipedia.org/wiki/Internet_of_things) is leading to a proliferation of connected devices permeating virtually every aspect of modern life.  One of the main domains of this technological revolution is [home automation](https://en.wikipedia.org/wiki/Home_automation), leading to "smart homes" that allow remote or automated control over entertainment, security, heating and ventilation, appliances, and lighting within the household.  It is this last application area, lighting, that we will be focussing on in the group project for CS3307 this year.

[Philips](https://www.philips.ca/) has been a key player in the lighting space for smart homes, particularly through its [Hue](http://www2.meethue.com/en-ca) brand of lighting systems that gives users great control over the lighting in their homes, through apps and smart devices that can turn their lights on and off, adjust their brightness, and tune their colour.   The main purpose of this project is to create a web service that interfaces with the Philips Hue lighting system, enabling users to access the various features of the platform through any connected web browser.

This document outlines the general specifications for this project.  Separate documents within OWL will provide particular specifications for each stage and milestone for the project.

**2.  Objectives**

This project is designed to give you experience in:

* applying object-oriented analysis and design towards a real-world problem
* working with, interpreting, and following a detailed specification provided to you
* implementing your design in C++ and having to deal with decisions made earlier in the design process
* using the [Wt framework](https://www.webtoolkit.eu/wt" \t "_top) for building a web server in C++
* creating graphical, user-facing web content as well as more programmatical, [RESTful](https://en.wikipedia.org/wiki/Representational_state_transfer) web services
* writing robust and efficient code
* write good, clean, well-documented C++ code that adheres to best practices
* reflecting on good/bad design decisions made over the course of the project

The project is intended to give you some freedom in design and programming to explore the subject matter, while still providing solid direction towards reaching a specified goal.

**3.  Working with the Hue**

The Hue is a fairly robust lighting system, so it is best to take a look through consumer-oriented documentation and promotional material at the [Hue main site](http://www2.meethue.com/en-ca), particularly if you are not familiar with this system.  This will give you a good overview of product features and general operations before proceeding.  Once you have familiarized yourself with the Hue, you should proceed to the [Hue developer's site](https://developers.meethue.com/).  Here you will find getting started guidelines, documentation, and full API specifications for their [RESTful](https://en.wikipedia.org/wiki/Representational_state_transfer) interface.  (To access the latter, at least one member of your group will need to register on their website; it's free and appears to be mostly harmless.)

For this project, you may work with Hue hardware that is available, although that is not necessary.  (As mentioned in class, we are looking into acquiring and installing some of this hardware in the department for you to use, though you are free to use what you may have at home as well.)  For the most part, you will likely be wanting to use the [Hue emulator](http://steveyo.github.io/Hue-Emulator/) that provides more-or-less fully functional Hue bridge and virtual lights to play with.  The emulator adheres nicely to the Hue API specifications and provides all of the necessary functionality for this project.  Aside from some not quite fully implemented features, the main difference between the emulator and an actual Hue installation is finding and connecting to the bridge (real or virtual), and in this regard, the emulator wins out.  A real Hue bridge uses [UPnP](https://en.wikipedia.org/wiki/Universal_Plug_and_Play) for discovery, which will be difficult for us to do using Wt (and let's face it, isn't exactly peachy to do otherwise).  The emulator, on the other hand, at least partially supports UPnP, but also runs on a preconfigured IP address and port, meaning that we can directly connect to it without having to go through any kind of discovery process.

As noted above, we will be using the [Wt C++ framework](http://www.webtoolkit.eu/wt" \t "_top), putting the expertise and experience you gained from the individual assignment into use into putting together a server for controlling lights in a Hue environment.  Keep in mind that while you are building a server in Wt yourself from the user's perspective, your server will be acting as a client as far as the Hue system and services are concerned.  As a result, your Wt server will likely make use of [Wt's built-in HTTP client](https://www.webtoolkit.eu/wt/wt3/doc/reference/html/classWt_1_1Http_1_1Client.html" \t "_top) to connect with and communicate with Hue.

**4.  Requirements**

Your project will need to adhere to a collection of functional and non-functional requirements.  In essence, the functional requirements outline what your application will need to do, while the non-functional requirements specify how you're supposed to go about doing things.

**4.1  Functional Requirements**

Functional requirements include required functionality and additional functionality, as discussed in the sections below.

**4.1.1  Required Functionality**

Implementing the required functionality will earn you to a base grade of 80% towards your project mark.  Increasing your grade beyond that point will require the implementation of additional functionality, as discussed below.  How the below functionality is delivered is for the most part up to you.  While we will not be grading visual appeal or aesthetics directly, if things slide to the point where your service is unintuitive, difficult to use, or unreadable, then this could impact your overall grade.

**Welcome / Login Screen**

Upon connecting to your server, the user should be presented with a welcome screen.  This should give a name and brief description of the server they have connected to; a logo and other visuals can be given too.  The user will also be prompted to authenticate with the server, typically by signing in to the service with a user name and password, or creating an account if they have yet to do so.

**User Account Management**

Each user of your server will have their own account, and all information managed by the server on Hue bridges and so on will be done on a user-by-user basis.  (So one user can not interact with the Hue setups of other users.)  On first access, a user will need to create an account whereby they give their first and last names, a contact e-mail address, and a password (twice, with the second time to double-check things).  Their e-mail address doubles as their user name and is used to sign in to the service on subsequent visits.  (As a result, you will need to ensure uniqueness on e-mail addresses entered by the user.  When a user has signed in, from any point in their session, they should be able to see who they are signed in as (by first or first and last name) and should be able to sign out.

Note that Wt has mechanisms built-in for authentication, but it might be easier to do this yourself.  As long as can sign in and you have some method for storing basic profiles (see the section on Persistence below for details), then you are fine.  Also note that you should not store passwords directly for security reasons, and should at least hash them (with a one-way hash of your choosing) and store only the hashes.  (When a user signs in, you re-hash their password and compare this to the stored hash to let them in.)

**Bridges**

Each user will need to be able to register and manage their bridges through the server.  Each registered bridge should have a name for the bridge, and a location (both user supplied strings), an IP address or host name (a single string to hold one of these), and a port number.  The user can also optionally specify a user name to use in connecting to the bridge, and this will default to newdeveloper (as used in the emulator) if one is not provided.

You will need to provide the ability to add, edit, and remove bridges on your server.  When a bridge is added, you should verify the information provided by making a test request to http://address:port/api/user where address, port, and user are replaced with information provided when adding the bridge.  Likewise, if any of this data is modified when editing a bridge, a similar test request should be made.  When a bridge is removed any persistently stored data making reference to that bridge should also be purged.

**Lights**

Manipulating Hue lights is the core of what you are doing in the application.  As the full light API is not supported by the emulator, you can limit your server to only support the functionality supported by the emulator.  (The emulator has its own mechanism for adding lights, and so requests to search for lights or delete lights are not supported.)  In other words, your server must be able to get all lights for a selected bridge, get light attributes and state, set light attributes (rename a light), and set light state.

In manipulating light state, your server must allow lights to be turned on and off, allow their brightness to be adjusted, and allow their colour to be adjusted.  You do not need to explicitly support every colour mode supported by the Hue system, but you need to support at least one that lets the user select from a full range of colours.  When changing light state, users should also be able to specify a transition time for the change to take effect.  Colour looping can be supported but is not required.

**Groups**

The Hue system allows lights to be grouped together and adjusted all at once.  As in the Hue API, for a selected bridge, your server will need to be able to list groups, create new groups, get and set group attributes, set group state, and delete groups.  Your server will need to support group state setting to the same extent as individual lights, discussed above.  You do not need to support groups that span more than one bridge, as that is not officially supported by the API.

**Schedules**

Instead of putting changes into effect immediately, it is possible to schedule such changes to occur at a set time in the future.  As the emulator allows you to create, update, and delete non-recurring schedules, this is the extent that you need to support in your server as well.  All of the supported light and group state changes from the sections above need to be schedulable through your server.

**Persistence**

Your server will need to be able to store some data persistently so that it is available from session to session.  In particular, this includes user account data and bridge data.  Your server does not need to make light, group, and schedule data persistent as this data is available from the bridges and is persistent as necessary there.  (Such data could also be readily changed by other apps accessing the bridge, so it would not generally be wise to assume that any data your server has stored locally is good for any extended period of time.)  For instance, if the user wants to list all lights on a selected bridge, you can simply query the bridge and construct the list on demand.  If the user wants to update the state of a given light, you can pull the current light state from the appropriate bridge, allow the user to adjust the state, and then push the new state back to the bridge.  None of the light data needs to be stored in either case.

Note that it is possible, as noted above, for other apps to be working with the same bridge at the same time as your server.  Consequently, from time to time, your server might find itself in a situation where otherwise seemingly reasonable requests to a Hue bridge will fail.  For example, suppose a user wants to update the state of a light, following the workflow as above.  Some time after the light's state is queried from the bridge, but before the new state is pushed, suppose that someone else removes the light.  (This couldn't be done using a server for this assignment, as light deletion is not supported, but it could be done directly within the emulator.)  The request to update the light's state will now fail.  Such situations cannot be readily prevented in general, so your server must be able to respond to such failures or errors in as graceful a fashion as is possible.  (Crashes are definitely not graceful!)

Persistence can be accomplished through either simple file storage or through the use of Wt's database services.  Reading from and writing to local files is a straightforward process to your Wt server, as it is no more complicated than in a traditional C++ application using file streams.  Using a database is more elegant, and is a key feature of Wt, but it is more complex and requires Wt to be configured properly and aware of the database package being used.

**4.1.2  Additional Functionality**

To bring your project grade to a possible 100%, you will need to add features that were not discussed above.  These will be evaluated subjectively as they come along.  If you have questions about how these might be graded, please ask your instructor.  Along the way, they will need to be documented as the rest of your project is.  (You will need to write user stories for them, adjust your models, and so on.)

To get inspiration for ideas, think about other useful functions that the user might be interested in seeing.  Feel free to look into other Hue apps to see features and functionality that they deliver that you are not already asked to do above.  Also, feel free to look through the Wt documentation and examples to see what other things it supports; perhaps something interesting can be found there.

Some ideas to get you started:

* When registering bridges, users need to provide a location for the bridge.  Above, this is simply text and could be as simple as "My house" or "Bob's bedroom", or it could be a full street address in a single string.  If you wanted to be fancy, you could provide a map view to show or pick locations.
* A user might need to have multiple Hue bridges in a single house or building, and so a user might want to have lighting groups spanning multiple bridges, even though the API does not support this.  You were told that you did not have to support this functionality for the project, but it could make for an interesting addition.
* The current Hue emulator does not support recurring schedules, as noted above and on the emulator's website.  That does not make it impossible.  For example, your server could make multiple, separate schedule entries in a bridge to make something recurring.  Alternatively, your server could manage a recurring schedule on its own, using [timers](https://www.webtoolkit.eu/wt/wt3/doc/reference/html/classWt_1_1WTimer.html) on your server to schedule state changes that are then executed immediately on the Hue bridge when instructed by the timer.

Please note that more than one piece of additional functionality might be required to bring your maximum potential grade up to the full 100%.  This depends on the creativity, effort, and technical difficulty of the features in question.  Again, if you have questions about this, please ask.

**4.2  Non-Functional Requirements**

Your application will need to adhere to the following requirements, and these requirements will be taken into consideration in the assessment of your project.

* The application must be developed in C++.
* The application must use the Wt framework.  It is recommended that you use Wt 3.3.8, however you may use Wt 4.x as well.  Your group must standardize across a single version of the platform.
* The application may use the Standard Template Library (STL) and/or the Boost C++ Libraries.
* The application should not need to use any other libraries.  If you do wish to use another library, you must obtain written consent (in e-mail form) from the instructor prior to its use.
* All code in the application must be commented using the [Doxygen documentation system](http://www.doxygen.org/" \t "_top).
* You may choose as a team the conventions and styles you wish to adopt in your code.  (You are free to follow [those given for the first assignment](https://owl.uwo.ca/access/content/group/df1797bc-56ba-4230-9065-cdbd30035a7d/Coding%20Guidelines), if you like.)  However, you must remain consistent in applying those conventions and styles across all files in the application.
* The application is not required to run on department system, but each team member must be able to compile it and run it from a development environment they have ready access to.  Please note that for acceptance testing, at least one member of the group may be required to demonstrate the application running live to the instructor and/or teaching assistants.
* The application must not create, modify, or delete files outside of the directory in which the application is installed, and subdirectories of this directory.  The exception to this would be any database access, which would by necessity work with files outside of your application's directory.
* The application must present a visible response to every user action.  Erroneous actions or actions that could not succeed for some reason must be met with a useful, professional error message.
* The application must be designed with object-oriented principles and an eye for good object-oriented design.
* Project code must be checked into the designated repository assigned to your team and members must commit and push code to the repository regularly.  Once groups have been set, we will be posting more information on the project management software to be used for the course, and this document will be updated accordingly.