**Music-Synced Smart Lighting System**

A dissertation submitted in

partial fulfilment of  
The requirement for the degree of  
**MASTER OF SCIENCE**

in

**Software Development**

in  
**The Queen’s University of Belfast**  
By

**Ryan Urquhart**

**40099112  
06/05/2025**

**Acknowledgements:**

* Bee-Yen Toh – My supervisor who was with me every step of the way
* Cherise McClean – My wife, a constant (and welcome) distraction
* SoDes – On GitHub, created a package that made this whole project possible on windows (PyAudioWPatch)
* aiXander – On GitHub, created a repo that I adapted, it provided fast Fourier transformations that I built the beat detection algorithm on top of.

Abstract:

This report covers the development of a Music-Synced Smart Light System. The development of this system was successful, however, due to time constraints the system is missing many aspects that would be desirable should the project be continued on. This is aided by the style of development of the system, the system has been developed in such a manner as to allow easy continued development. The system takes advantage of FFT of audio data from PyAudio to allow the system to detect the beat of music and change bulb brightness in time with that beat.

Table of Contents

[Introduction: 5](#_Toc195705077)

[1. Understanding the Problem 6](#_Toc195705078)

## Introduction:

This report is focusing on the development of a Music-Synced Light Control System (hereafter referred to as, Bulb Bop). The report will consist of six chapters, Chapter One, Understanding the Problem, Chapter Two, User interface-design, Chapter Three, Architecture design and algorithm explanation, Chapter Four, Experimentation, Chapter Five, Testing, Chapter Six, Evaluation and Conclusion. The content of these chapters will be explained in the rest of the introduction.

Chapter One will contain my exploration of the problem and the basis on which I started undertaking the development of the software system. This will include my initial understanding of the project, based on the title and the preliminary meetings with my supervisor. It will then progress into my initial investigations about the potential solutions to the problem at hand. Finally, this chapter will end with the decided upon plan of action for development.

Chapter Two will contain the plan and the development of the user-interface. This will include the early iterations of the user-interface, and how these early iterations influenced the final design. It should be noted however, that the bulk of the work in this project was the beat detection algorithm and therefore, there was less thought put into the user-interface than there was into the back end of the system.

Chapter Three will contain the plan and the development of the architecture and algorithm design. This section will explain the architecture upon which the system can be ran, and the algorithms that allow it to run. This is the chapter in which the bulk of the work of this project will be explained.

Chapter Four will contain the experimentation that I have conducted due to this project. This will include looking at different languages that I attempted to utilise, as well as the research and experimentation that went into the development of the beat detection algorithm. Various iterations and experiments will be present in this chapter.

Chapter Five will contain the testing carried out on the project, during its development, and before the submission of the project. This will include risk assessments of the project, and a discussion of the rationale of the tests that were designed, as well as an examination of the areas of the project that were not thoroughly tested and an explanation as to why.

Chapter Six will be the final evaluation and conclusion of the project. Discussing the success or failure of the development of the system. It will evaluate different technologies utilised and how they have impacted the final system shape. There will be a reflection on my own development of the project, highlighting areas that were handled well, and areas where there could have been further development.

# Understanding the Problem

The initial description of the project is as follows,

“Develop a system that integrates music with dynamic lighting displays to create an "instant smart disco light" function or a “smart lighting solution for improved well-being.” The system should control a Wi-Fi-enabled smart light bulb, adjusting its brightness and colour in sync with music rhythms, enhancing the atmosphere in any setting.”

As well as this, there were four main functional criteria for the development of the system.

1. The system should connect to and control a Wi-Fi enabled smart light bulb
2. The system should detect music rhythm and tempo from an audio source and synchronise the light settings accordingly
3. The system should provide a primary control interface
4. The system should offer multiple lighting modes

There was no restriction placed on the technologies to be utilised in this project. No specific bulbs, programming languages, or web frameworks. However, due to having to use third-party hardware (The smart bulbs) there may be a specific API dictated by the manufacturer that would have to be used.

The initial prompt has no definition of the user the application is supposed to be designed for. However, during meetings with my supervisor the intended user was defined in a varied way. From families, to DJ’s, to people hosting a party. The decided goal was that the user would be someone hosting a party in their house, this user would be either using a microphone attached to the computer running the system, or the computer running the system would be playing the music from its speakers.

From the initial description of the project, research was carried out, this research covered two main areas. The first was the bulb, which brand would be appropriate for the project. The second was the programming language, this programming language had to fulfil two criteria, it needed access to the audio from the operating system I was developing on (Windows and Linux), it also needed to be able to host a web server which would provide the user interface.

The research into the bulbs brought up many different options for use, Wiz, TP-Link Kasa, TP-Link Tapo, YeeLight, Shelly, Philips Hue, Athom.tech, and switchbot. From all of these bulbs I decided to use Wiz bulbs. The reasons for this were the cost, the open API (*bulbs like TP-Link Tapo have closed API’s so users must use their app*), the app does not require a hub to operate (*Philips Hue does*), and from my reading of various forums wiz was highly recommended as a bulb brand. Based on this research I purchased a WizBulb A60.E27 bulb.

The research into the programming language to use initially started with Java, the reason for this simply being that much of the CSC7063 Computer Programming course has been taught in Java. Two libraries I looked at were TarsosDSP(Six, 2011/2025), and Jipes by Tagtraum(Schreiber, 2013/2024), both of which I was unable to gain access to audio data from the host machine using. Jipes was unsuited as it was not made for real-time processing of audio information. With regards to TarsosDSP, it used Gradle to build it, this was something that, given the time I needed to get the project started in, I was unable to figure out in time.

As such, I looked for another programming language that would allow me access to this host audio data. For this I turned to python, there is a python package names, PyAudio, and it reliably and easily allows developers access to the host machine audio data. The issue however, is that PyAudio does not allow access to the loopback api present on windows machines that allows the developer access to real-time speaker audio data. This was solved when I found a windows specific patch of the PyAudio library, called PyAudioWPatch. This allows PyAudio the ability to access the loopback api, this with the Django library that is able to serve web apps with a great deal of flexibility made python the perfect choice.

# User Interface Design

A black and white image of a person standing in front of a white sky

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

Figure 2 - Index Page (Mobile View)

Figure 1 - Index Page (PC View)

The user interface for this system is quite simple. There are a total of four different views that the user can access. Each of these views are dynamic to varying degrees, Django gives the option to develop html that will dynamically change based on a context provided by the developer. For instance, the bulbs on the index page render depending on if they are present in the database or not.

The first part of the user interface that we will discuss are the parts common to all views. The main part of this is the navbar, this contains the title, links to other views, buttons to activate functions of the system, and a light/dark mode toggle button. There is also a footer on the page, a line that contains the text, ‘2023-25 : Bulb Bop : Ryan McClean : 40099112’. Part of these aspects are dynamic, the audio sync option for instance, only shows when bulbs are recorded in the database.

The second view we are going to look at is the home page, the index page (figure 1 and 2), this is the main page that the user will be engaging with. This page displays icons for the bulbs that are recorded in the database, their names, as well as graphically displaying their state. There are tooltips that appear when the user hovers their mouse over the bulbs, this informs the user that should they click on the it will toggle the state of the bulb. Should the user not yet have connected the system with any bulbs then the index page will display a prompt to the user to use the discover function of the system (figure 3).

A screenshot of a computer

AI-generated content may be incorrect.

Figure 3 - User prompt when bulbs aren't saved

A screenshot of a computer

AI-generated content may be incorrect.

Figure 4 - FAQ Page (PC View)

The third aspect of the user interface we will discuss is the ‘FAQ’ (frequently asked questions) page (figure 4). This page covers some basic setup questions that a user may have, from connecting lights to troubleshooting, and a contact email for support (not a real email). This page is referenced in the tooltip to help new users get started with the application. It also links to the WizBulb app, to show users how to connect their bulbs to the local network (unfortunately using the system to connect the bulbs to the local network was impossible as it requires Bluetooth and is undocumented).

The fourth page of the system to discuss is the ‘About’ page. This is a simple page that just displays information about the project, it’s goals, and some information about the author if the project. Like every other page, this page has the navigation bar which allows the user, should they have bulbs saved in the database, to activate the audio sync. As this page is mostly static, apart from the navigation bar, there is nothing else to say about it.

The final page to discuss in this section is the ‘Edit’ page, for the bulbs. This page allows the user to edit or delete the bulbs that they have saved in the database. The user can edit the bulbs name and Ip address, but that is all. As when the system has a bulb saved with a name and an Ip address, then it will query the bulb and update its status in the database itself. When the user clicks on edit, it will show a modal that the user can interact with to edit the bulb. When the user clicks on save edit, or delete, the page will display a success or error message in the bottom right corner.



Figure 5 - Edit page (PC view)

A green screen with white text

AI-generated content may be incorrect. A red sign with white text

AI-generated content may be incorrect. Figure 6 - Delete Success Message Figure 7 - Edit Error Message

# Architecture Design and Algorithm Explanation

In this section of the report I will be going over the ‘backend’ of the system. This will include a description of the api endpoints, how the system was configured to interface with bulbs on the network, a description of how I gained access to the audio data of the host machine, and finally an explanation of the algorithm to detect beats in music. This section will encompass most of the work done on this project and therefore will be the longest section in this report.

This project has 14 different API endpoints, not including requests that get images and JavaScript code. Four of these endpoints are for different views or pages, of the user interface, one is reused for bulb form submission, therefore, these will not be discussed in this section, as the views were mentioned in chapter 2. There are three endpoints for bulb control, four (including the reused view endpoint) for database operations and, two for error and success handling.



Figure 8 - API Endpoints

The endpoints for bulb control are as follows, “/discover/”, “/toggleBulb/”, “/queryBulb/”, and “/colorBulb/”. Of these four, only three are currently utilised in the system, “/colorBulb/” was developed for use, but when testing with a colour changing bulb, and a non-colour changing bulb this endpoint caused issues, so was left for future development, which ultimately ended up being scoped out of the project. “/discover/” is used to discover new bulbs on the network, it sends a broadcast UDP packet, to which bulbs connected to the network respond with their current state. This response is shown to the user, who can give the bulb a name, which is then sent to another endpoint, which will be mentioned later. “/toggleBulb/” is the main endpoint the user will trigger from the user interface, when they click on the icon of a bulb it will trigger this endpoint, which sends a packet to the specific bulb and instructs it to turn off, or on, depending on the state of the bulb recorded in the system. “/queryBulb/’ is similar to “/discover/”, except rather than sending a broadcast packet to all Ip addresses, it send packets to specific addresses to query the state of specific bulbs.

The endpoints that control database operations are, “/”, “/delete/<str:ip>/”, and “/edit/<str:ip>”. The “/” endpoint is the index endpoint, when this endpoint is triggered with a GET request, then it renders the home page, when it’s triggered with a POST request, then it submits a Django model form. This Django model form is how the system saves new bulbs in the database. When the form is submitted, it is checked to ensure that it contains no errors. The criteria for this check is that the Ip address is valid, and that the name of the bulb is not null. The other information for the bulb is filled in from the return from the broadcast packet that was sent during discovery. The next two endpoints are accessible to the user through the edit bulbs page, “/delete/<str:ip>/” and “/edit/<str:ip>/” both of these endpoints require the Ip address of the bulb in which they are editing, as a unique identifier. “/delete/<str:ip>/” will delete the bulb with the specified Ip address, and “/edit/<str:ip>/” will submit a form, that is verified, of a new Ip address or name for the bulb, this then updates the model in the database for that bulb.

There are two endpoints that control error and success handling, these clear the error and success messages, both are to prevent error, and success messages being displayed multiple times. These endpoints are never interacted with by the user manually, they are triggered through JavaScript that runs when every page loads. While this does not interfere with a message that is being displayed in the page that has just been loaded, it will prevent a message from being displayed in another page that the user navigates to.

There are two endpoints for controlling the audio synchronisation, “/activateSync/”, and “/stopsync/”. These are fairly self-explanatory, “/activateSync/”, this endpoint accepts a POST request that contains the index number of the audio device selected by the user. This then activates the audio sync in the system for that audio device, the system waits thirty seconds to ensure that the sync program has activated, then it sends a status message to the user interface.

This concludes the endpoints of the API that are activated by the user, or the JavaScript the pages contain.

**SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING and COMPUTER SCIENCE**

**CSC7058 – INDIVIDUAL SOFTWARE DEVELOPMENT PROJECT**

A signed and completed cover sheet must accompany the submission of the Individual Software Development dissertation submitted for assessment.

Work submitted without a cover sheet will **NOT** be marked.

**Declaration of Academic Integrity**

Before signing the declaration below please check that the submission:

1. Has a full bibliography attached laid out according to the guidelines specified in the Student Project Handbook
2. Contains full acknowledgement of all secondary sources used (paper-based and electronic)
3. Does not exceed the specified page limit
4. Is clearly presented and proof-read
5. Is submitted on, or before, the specified or agreed due date. Late submissions will only be accepted in exceptional circumstances or where a deferment has been granted in advance.
6. Software and files are submitted via Canvas.

**I certify that that the submission is my own work, all sources are correctly attributed, and the contribution of any AI technologies is fully acknowledged.**

**I declare that I have read both the University and the School of Electronics, Electrical Engineering and Computer Science guidelines on plagiarism - http://www.qub.ac.uk/schools/eeecs/Education/StudentStudyInformation/Plagiarism/ - and that the attached submission is my own original work. No part of it has been submitted for any other assignment and I have acknowledged in my notes and bibliography all written and electronic sources used.**

**I am aware of the disciplinary consequences of failing to abide and follow the School and Queen’s University Regulations on Plagiarism.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name: (BLOCK CAPITALS)** |  | | |
| **Student Number:** |  | | |
|  |  |  |  |
| *Student’s signature* |  | *Date of submission* |  |