NAIT

Edmonton, Alberta

**IZ-KIT**

As a submission to

Mr. Kelly Shepherd, Instructor

English and Communications Department

&

Mr. AJ Armstrong, Instructor

Computer Engineering Department

Submitted by

Isaac Wittmeier, Student & Ezekiel Enns, Student

CMPE2960

Computer Engineering Technology

April 16, 2021

Mr. Isaac Wittmeier, Mr. Ezekiel Enns

Students

NAIT

11762 – 106 Street NW

Edmonton, AB T5G 2R1

April 16, 2021

Mr. Kelly Shepherd, Mr. AJ Armstrong

Instructors

NAIT

11762 – 106 Street NW

Edmonton, AB T5G 2R1

Dear Mr. Shepherd and Mr. Armstrong,

As per the requirements of CMPE2960, we are submitting the report titled IZ-Kit for evaluation.

This report describes our design and implementation processes, including challenges we faced and how we overcame these obstacles. It describes how our project’s devices function and how they interact with each other including detail into how the software and hardware were created and how their processes interact with each other.

We would like to take this opportunity to express our gratitude towards all of our capstone instructors for the advice and support they gave us throughout the duration of this project. We would also like to thank every instructor that taught and encouraged throughout our NAIT career. We really appreciate their advice and support. We learned many valuable concepts and practices that will be beneficial in our future careers. Thank you!

Sincerely,

Isaac Wittmeier Ezekiel Enns

CNT Student CNT Student

Table of Contents

[List of Figures and Tables iv](#_Toc69768370)

[Abstract v](#_Toc69768371)

[1.0 Introduction 1](#_Toc69768372)

[1.1 Topic 1](#_Toc69768373)

[1.1 Report Overview 1](#_Toc69768374)

[2.0 Overview 2](#_Toc69768375)

[2.1 Overall Design Choices 4](#_Toc69768376)

[3.0 Wi-Fi Modules 5](#_Toc69768377)

[3.1 Hardware Technical Specifications 6](#_Toc69768378)

[3.2 Module Circuit Types 7](#_Toc69768379)

[3.3 Virtual Devices 10](#_Toc69768380)

[3.3 Server Communications 11](#_Toc69768381)

[3.4 Challenges 11](#_Toc69768382)

[3.5 Reflections 12](#_Toc69768383)

[4.0 Front-end Website 12](#_Toc69768384)

[4.1 User Accounts 14](#_Toc69768385)

[4.2 Admin Panel 16](#_Toc69768386)

[4.3 Programming Page 17](#_Toc69768387)

[4.3 Challenges 20](#_Toc69768388)

[4.4 Reflections 21](#_Toc69768389)

[5.0 Back-end Server 21](#_Toc69768390)

[5.1 Database 22](#_Toc69768391)

[5.2 Socket Server 24](#_Toc69768392)

[5.3 Rest API 26](#_Toc69768393)

[5.4 Challenges 27](#_Toc69768394)

[5.5 Reflections 27](#_Toc69768395)

[6.0 Conclusion 28](#_Toc69768396)

[References 29](#_Toc69768397)

[Appendix A. Bill of Materials 31](#_Toc69768398)

[Appendix B. Circuit Schematics 32](#_Toc69768400)

# List of Figures and Tables

**Figures**

Figure 1: IZ-Kit Network Communications…………………………………………………………………………….……3

Figure 2: Difference Between Star and Mesh Topology………………………………………………………….....4

Figure 3: ESP8266 NodeMCU……………………………………………………………………………………..……...........6

Figure 4: IZ-Kit Devices..…………………………………………………………………………………………………………….8

Figure 5: Pushbutton Module Circuit…..…………………………………………………………………………………….9

Figure 6: Python Virtual Timer Module….............……………………………………………………………………..10

Figure 7: Website Login Page........................................................................................................13

Figure 8: Website Create Account Page.......................................................................................15

Figure 9: Website Admin Panel Page............................................................................................16

Figure 10: Website Programming Page........................................................................................17

Figure 11: Sample IZ-Kit Program.................................................................................................19

Figure 12: Backend Components..................................................................................................22

Figure 13: Database Schema.........................................................................................................23

Figure 14: Database Dataflow.......................................................................................................24

Figure 15: Led Module Schematic.................................................................................................32

Figure 16: LCD Module Schematic................................................................................................33

Figure 17: Light-Sensor Module Schematic..................................................................................34

Figure 18: Pushbutton Module Schematic....................................................................................35

Figure 19: RGB LED Module Schematic.........................................................................................36

**Tables**

Table 1: Bill of Materials…………………………………………………………………………………………………………..31

# Abstract

A project that aims to develop a flexible framework for home automation. It utilizes a fully functioning configuration website and a powerful backend. The project comes complete with an HTML/JavaScript/CSS/Bootstrap frontend, a PHP/Python backend, user accounts, a REST API, and even its own programming language. Insight is given into how several hardware and software modules can be rapidly designed and produced to work with a dynamic data routing system. The back end of this system also covers database design concepts in relation to a complex system with several user configurations, and how to minimize latency in a web-based project using sockets.

# Introduction

Home automation, and its umbrella of Internet of Things (IoT) devices, a collection of internet-enabled devices that make life easier, is an exploding industry. The demand for IoT devices has been expanding at an alarming rate. As of 2020, there are more IoT devices connected to the internet than non-IoT devices such as personal computers and smartphones (Lueth, 2020) and this number is still growing. By 2025, it is expected that over 30 billion IoT devices will be connected to the internet, an average of almost four devices per human on Earth (Lueth, 2020).

## 1.1 Topic

The home automation system discussed in this report is designed to be a home automation setup that grants the user full control and customization over its functionality and input/output modules. It is fully customizable; with a custom programming language and the possibility of adding unlimited types of input and output modules into the system to perform any automation task required.

## 1.1 Report Overview

The purpose of this report is to document the experiences, decisions, and challenges in the process of creating a home automation system christened the IZ-Kit. It will begin with a high-level overview of the project. It will then move on to the project’s major systems: the input/output modules, the management website, and the backend web server. For each system, the report will include a detailed description, (including technical info) of each subsystem, the challenges faced, and the reflections gathered. The report will then finish with a conclusion.

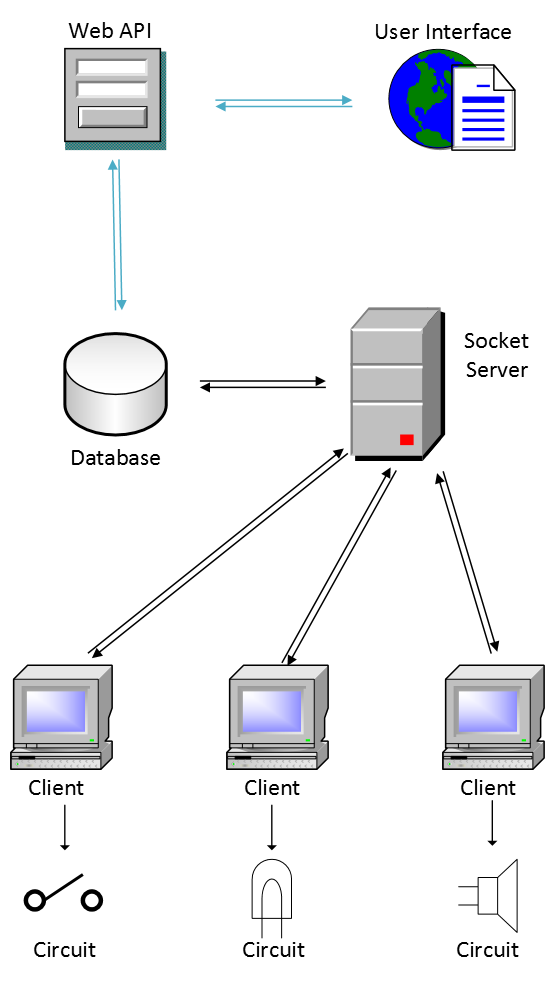
The report will focus on the major components of the project and delve into the technical details only as far as needed to explain the project.

# 2.0 Overview

The IZ-Kit is a combination of a server and modules that communicate with the server. Comparable to an electronics engineer soldering electronic components together to form any type of circuit, users can craft the module connections into a virtual circuit however they desire, using a custom programming language. The illustration below presents a visual overview of how the system operates.

**Figure 1**

*IZ-Kit Network Communications*



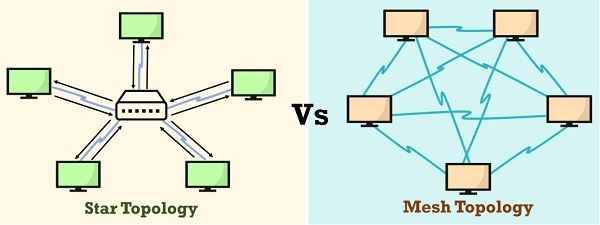
*Note.* Visio illustration of IZ-Kit communications. Own work.

As seen in the illustration, the modules all connect to a central server. This server and a web API interface simultaneously communicate with a database to exchange information. The web API in turn communicates with the user through a web interface to allow user programming and control of the network. The system is highly modularized, allowing for any combination of input and output modules to connect through the server’s routing capabilities. New modules can easily be created using the modular base code of every circuit. This allows for limitless possibilities of automation machines. The IZ-Kit can also be used as a fun way for children to learn coding. Using the system and its simple programming language, children can create any toy or machine that they imagine.

## 2.1 Overall Design Choices

A choice had to be made whether to implement a Star network topology where devices (modules) connect to a central switch (or server in this case) (Keary, 2020) or a Mesh topology in which each is connected to every other module with a direct link (Keary, 2020), as seen in the illustration below.

**Figure 2**

*Difference Between Star and Mesh Topology*

*Note:* Source: TechDifferences.com

After discussion, it was determined that a Star network would be easier to manage through a central server and would require less storage space on the modules, which have limited memory.

In the prototype version of the IZ-Kit, the server is hosted on remote NAIT server infrastructure. This was done to avoid the large amount of setup required to set up a webserver on a Raspberry Pi. In a production version of the kit, the server software would be ported to a Raspberry Pi with wireless capabilities to give the user physical control of the server and allow the user the choice of keeping the IZ-Kit infrastructure local rather than connected to the internet.

# 3.0 Wi-Fi Modules

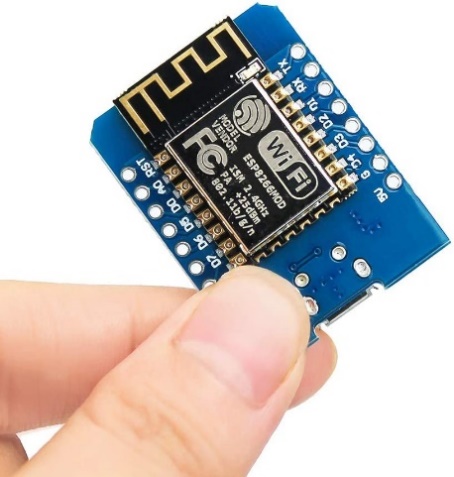
The IZ-Kit relies on a network of hardware and software modules connected to a server to allow interfacing with the real and virtual world through sensors, actuators, and software programs. Each module has its own distinct function as either an input or an output. Design choices have been made such that each module will perform one task and that task only to allow for easy addressing and modularity. Modularity has been paramount in the design of these modules. Each one has a similar base structure, and they all communicate with the server in the same way. They only differ in how they output or receive information from the world. Multiple modules can easily be connected to the server in a way that allows them to function together as a unit and create complex systems.

## 3.1 Hardware Technical Specifications

At the heart of each hardware module lies a simple but powerful microcontroller, the Wi-Fi enabled Wemos D1 Mini, also known as a NodeMCU, seen below.

**Figure 3**

*ESP2886 NodeMCU*



*Note.* Source: Amazon.com

Prototypes for this project have been made using a Wemos D1 Mini clone due to cost; however, the functionality is the same as the original. The NodeMCU is based on an ESP8266 chip and features 4MB of flash memory, 11 digital IO pins, 1 analog pin (3.2V max input) (Wemos, n.d.) and 2.4GHz 802.11b/g/n Wi-Fi connectivity. With an operating voltage of 3.3V, the D1 Mini runs every GPIO pin at 3.3V but has a switching power output of 5V for IO devices that require more power.

## 3.2 Module Circuit Types

With the versatility of the D1 Mini’s analog input pin, 5V voltage output, and multiple GPIO pins—including some with I²C support (Wemos, n.d.), innumerable types of devices can be created to work with the IZ-Kit’s system. Possibilities are endless, ranging from simple LED outputs to I²C LCD displays, to relay circuits which could control Space Shuttle launches. Every module will have a globally unique ID that will be referenceable in the programming area of the IZ-Kit’s website. For prototype purposes, the kit only contains a button input module, an analog light sensor input module, an LED output module, an RGB LED output module, and an I²C LCD display output module, shown in figure 4 below.

**Figure 4**

*IZ-Kit Devices*

A picture containing electronics, circuit

Description automatically generated

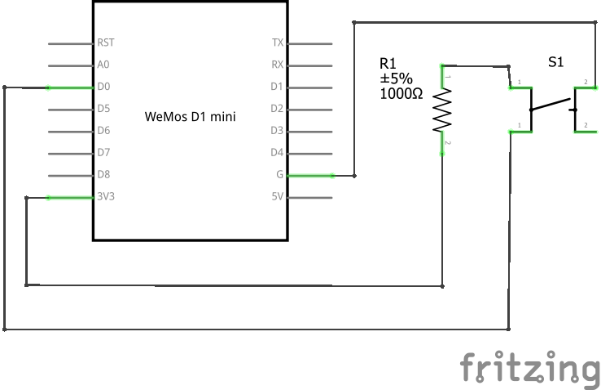
*Note.* Picture of IZ-Kit modules. Own work.

From left to right, the modules in the image are a pushbutton module, an RGB LED module, an LED module, an LCD module and a light-sensor module. These protype modules are created from common Arduino kit components, including a basic four-pin pushbutton, a typical blue LED and a common photosensitive resistor. An HW-479 module is used for an RGB LED output and a 16x2 character LCD is used with an I²C backpack for an LCD output.

As seen in figure 5 schematic of the pushbutton module below, the circuits are simplistic, but easily reconfigurable.

**Figure 5**

*Pushbutton Module Circuit*



*Note.* Schematic of pushbutton circuit. Own work.

The same pins used for the pushbutton could easily be repurposed for a switch or other binary input. An LED module could easily become a relay circuit and activate virtually any electric device. Using the Wemos microcontroller’s I²C pins, any I²C device could be managed through the IZ-Kit.

In terms of software, the module code is written in C++ using Arduino and ESP8266 Wi-Fi libraries. An additional LiquidCrystal\_I2C library was required to control the LCD through the microcontroller’s I²C pins. They all utilize the same base code but have different functionalities when converting input and output data to physical world device control. To communicate with the IZ-Kit server in a simple and module manner, a generic, C++ socket class has been created and implemented into every module.

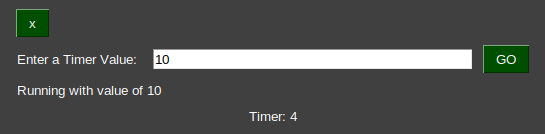
## 3.3 Virtual Devices

In addition to the hardware modules, the IZ-Kit server also allows for the connection of virtual, software devices. These can be programs locally on the server, or any internet enabled piece of software. Examples include text message alert connections, timers and Discord bots. If the software follows the socket protocols required on the server, it can interface with the IZ-Kit. Users can elect to create their own software modules or use modules that would be created for the kit in a production version.

A simple Python timer has been created for the IZ-Kit to demonstrate its virtual device functionality, seen below.

**Figure 6**

*Python Virtual Timer Module*



*Note.* Screenshot of a Python virtual timer module. Own work.

To start the application, the user must run the application and enter the timer length desired in seconds. Once the desired time is entered and submitted, the device will connect to the server and update it with its new state every second. This timer cycles a from zero to value inputted.

## 3.3 Server Communications

Due to the modular design of the project, a generic socket client class was created to handle all the communications between the nodes and the server. The class handles connection establishment, sending data, and receiving data. Design and functionality of the class is particularly made to interface with the server, having special functions for getting updates on events, registering the module on the server and sending information back to the server. The reasoning behind this was to allow for variations in the type of modules. Since the class was generic, it was easy to port over to other programing languages, allowing the creation of software-based modules as well.

## 3.4 Challenges

The first generic socket class for the hardware modules was written to run with the ESP2886 chipset and therefore had to use the Arduino Wi-Fi libraries. This documentation was vague and hard to understand with a simple description of what the functions were but no information on how they worked (*Arduino, n.d.*). Online Arduino forums were also unhelpful, with most of the users not having direct experience in programming or just being hobbyists. The best success resulted from going into the Arduino library code and reverse engineering the functions.

## 3.5 Reflections

The prototype versions of the modules are only powered by micro-USB power and they do not have an on/off switch. For the scope and time allotted of the project, it was determined that the modules would not require a battery circuit to be complete prototypes.

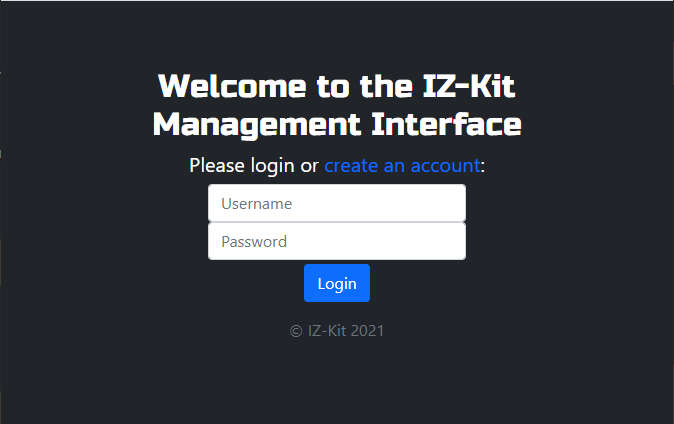
The socket connection class in the modules uses blocking code to wait for data to be received. Since a non-blocking solution is possible but was not implemented due to time constraints, the code should ideally be converted to its non-blocking counterpart. This is to allow for output modules to do side processing while waiting to receive data, which could possibly lead to faster response times and added productivity from the module.

# 4.0 Front-end Website

The IZ-Kit system requires an interface that allows users to create code that will control the modules. A website hosted on the server where the modules connect to serves this purpose. Using a LAMP format, the backend is written in PHP and the frontend consists of a typical combination of HTML, JavaScript, and CSS. Bootstrap has been integrated as well to allow for basic mobile functionality and to assist with quickly generating the website’s design. figure 7 below, of the website’s login page, represents a sample of the website’s overall style.

**Figure 7**

*Website Login Page*



*Note.* Screenshot of website’s login page. Own work.

As seen in the figure above, the website uses Bootstrap’s default, dark, theme to generate a visually appealing site.

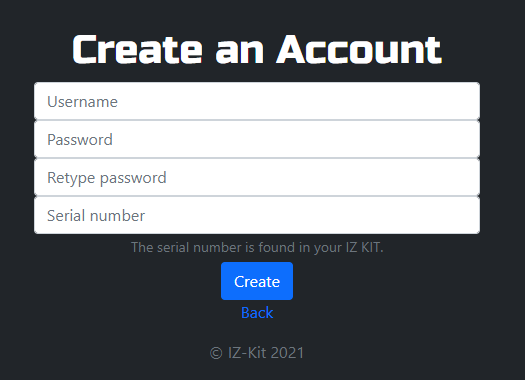
The website features an administrative panel where device info can be queried and modified and where users can directly control the modules and read their input values directly. It additionally features a main programming page where users can create and upload code to the modules with basic code debugging and save/load functionality for code files. The website includes a simple page to monitor the server’s status and user accounts are incorporated into the website as well.

## 4.1 User Accounts

For security purposes and the ability for different users to create separate programs, user accounts have been integrated into the website. Should a user choose to port-forward their website into the world-wide-web, the accounts will create a level of security between the outside world and the system’s modules. User accounts also allow users to save and modify configuration files that are separate from other users. Each account is granted the ability to save and store one code file which can be retrieved and uploaded to configure the device network. A create account page, see figure 8 below, allows a user to create an account.

**Figure 8**

*Website Create Account Page*



*Note.* Screenshot of website’s account creation page. Own work.

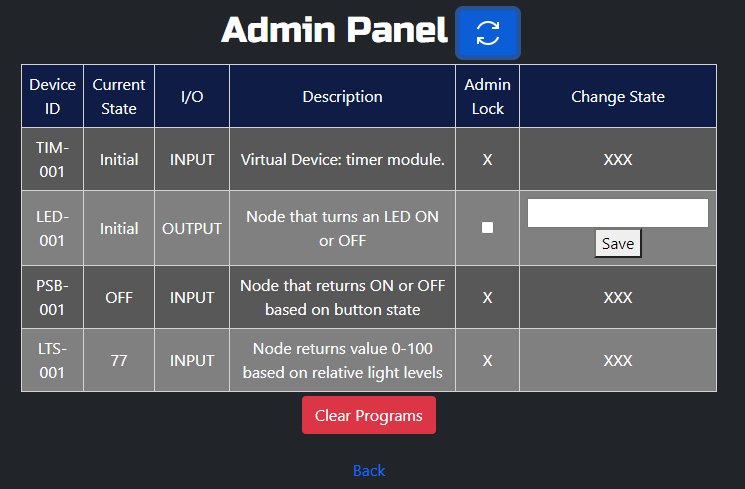
After the account creation page sends an account creation request to a PHP webservice, the webservice validates the inputted data including password length. One of these validations is a password length requirement of eight characters. Due to scope and time constraints, extra password security requirements were not implemented; however, the account creation includes another security feature to prevent unauthorized users from creating accounts. Creating an account requires the serial number of the IZ-Kit to be inputted. The serial number is physically attached to the kit, requiring users to have physical access to the kit in order to create an account. Accounts can be deleted from an account’s management page, accessible once a user logs in.

## 4.2 Admin Panel

An admin panel, seen in figure 9, has been implemented into the website to allow manual control and monitoring of the connected hardware and software modules.

**Figure 9**

*Website Admin Panel Page*



*Note.* Screenshot of website’s admin panel page. Own work.

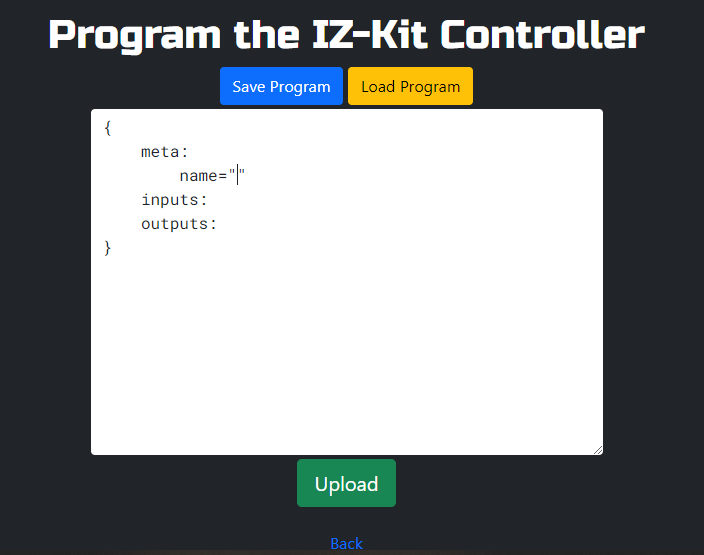
The panel displays the stats of the currently connected devices, including their ID and state. It gathers this information via a REST API on the server. Additionally, the panel allows users to change the state of output devices and deletion of all the programmed circuits uploaded onto the server. The admin lock, seen in figure 9 alerts the server that the state of an output will be changing and prevents any programmed circuits from modifying the state value in the meantime.

## 4.3 Programming Page

The programming page is the central component of the website. Seen in figure 10 below, it allows users to program virtual connections between the modules, using a programming language custom-designed for the kit.

**Figure 10**

*Website Programming Page*



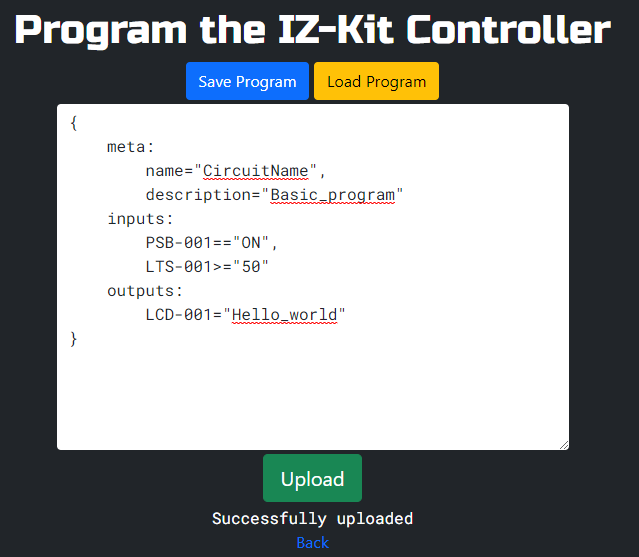
*Note.* Screenshot of website’s programming page. Own work.

The programming page includes a saving and loading feature for each account to save one program. A few IDE-style shortcuts have been implemented for as well, including a ##[Enter] command which generates a template as seen in figure 10, quote completion to create a closing quote character for each opening quote typed, and some spacing assistance. Pressing the “Upload” button sends an upload request with the program text, parsed in JavaScript via Regex, to the backend server via a REST implementation.

The figure below presents a sample program that can be uploaded to the IZ-Kit. When uploaded to the server, an LCD module displays “Hello\_world” when a pushbutton module is depressed, and a light sensor reads a value greater than or equal to 50 (light sensors read between 0 and 100).

**Figure 11**

*Sample IZ-Kit Program*



*Note.* Screenshot of a sample IZ-Kit program. Own work.

Meta data in the meta section can include any combination of tag and string value separated by commas, if “name” is one of them. Every other tag will not be saved to the server, but it allows for human-readable meta information on code files. The input section is where input module state conditions can be programmed to act as triggers for the outputs changing state below. An input module ID is required along with a comparison symbol and a state value in quotes. Output states are coded in the outputs section. When all the input state conditions are true, the output modules, denoted by their ID, set their state to the value after a single equal sign, surrounded by quotes. An interesting feature of the programming language is that output states do not revert to their previous state when the input conditions become false. This allows for greater flexibility in the code as the user can program the setting and resetting conditions independently.

## 4.3 Challenges

The programming input area is a simple HTML <textarea>, meaning its text capabilities are very limited and the default operation of a tab input will move the webpage’s focus to a different element on the page. A solution was found on ExceptionsHub.com, which involved creating a JavaScript function that checks for a tab key and manually inserts spaces into the <textarea> at the current index *(ExceptionsHub Admin, 2017)*. It also involves disabling the default action of the tab key. The same principle was used to create the programming shortcuts and template creation.

Another challenge faced was the parsing of the code file. Initial programming used JavaScript to process the file through Regex on the client side; however, the team later attempted to parse the file on the PHP server side instead using the same Regex commands from the JavaScript version. The commands did not function the same in PHP, however, and it was determined that parsing the file on the client end would be acceptable rather than expending valuable time reworking all the parsing.

## 4.4 Reflections

In its current version, the admin panel does not allow input states to be changed. This occurred due to scope and time restrictions. It was not required to prove functionality of the kit, but it would be a useful feature for a user debugging their circuit code.

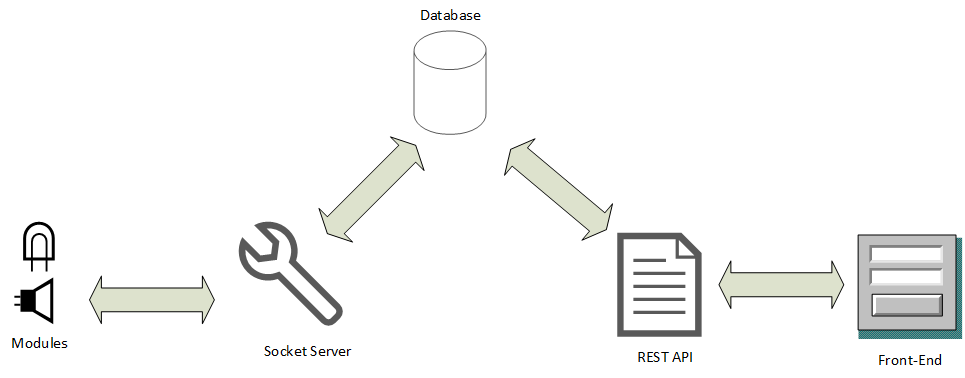
Version one of the programming language used on the programming page has a quirk in that all spacing is removed via Regex when parsing. Due to this, device states and meta information do not allowing spacing. For example, an LCD module would not be able to display words separated by a space. This would be fixed should a second version of the language be created.

# 5.0 Back-end Server

In previous sections, the IZ-Kit has been described as a series of connections between input and output modules. This creates a system in which the states of inputs go through a “black box” that leads to the change in state of an output; the backend is that “black box”. When designing the backend, several questions needed to be answered. How will modules be able to interact with the server? Where and how will the modules previous states be sorted? How will the user control the end results of systems they design? How will the systems made by users be stored and reproduced by the server? Finding the answers to these questions shaped the design of the backend. There are three main components that all interact with each other to form the backend: a REST API that acts as the server's communication with the end user, a socket server that allows for communication with the modules, and a database, which shares data between the previous two parts. These three main components are presented in the figure below.

**Figure 12**

*Backend Components*



*Note.* A diagram of the parts of the backend. Own work.

The front-end website communicates to the backend database through a REST API while the input and output modules simultaneously interact with the database through a Python socket server.

## 5.1 Database

Conceptualizing how the data in this complex system would be stored was a difficult task. In the book, *Theory of Modelling and Simulation,* the authors describe the external behavior of a system as a collection of its inputs and past output states. The internal behavior is how inputs transform the current state into the successor states and how the state is mapped to outputs *(Zeigler et al., 2018)*. A backend system design was created around these principles and resulted in the database shown below.

**Figure 13**

*Database Schema*

Graphical user interface, application

Description automatically generated

*Note.* IZ-Kit Database schema. Own work.

In the database, the external behavior of the modules is stored within the “Stats” table, which is constantly updated whenever the state of a module changes. The internal behavior of the module system is stored as links between three different tables, namely the “Circuits” table, the “CircuitToInput” table, and the “CircuitToOutput” table. The “Input” table and the “Output” table map the relationship between the external and internal states of the modules. The “Circuits” table is used to transform the current output state into its successor states.

As seen in figure 13, the tables “Input” and “Circuits” both have a column called “Trig”. These columns store Boolean values that represent whether or not certain conditions have been met. For the “Circuits” table, the “Trig” column is set to true when all the inputs linked to it, using the “CircuitToInput” table, have had their local “Trig” columns activated. A triggered column in the “Input” table will only be set to true when a comparison operator, defined in the “Sign” column, returns a true value when comparing the “OnValue” column with the corresponding “Stats” table “Value” column. The two tables are linked using the “DevID” foreign key. This database input to output data flow is represented in the figure below.

**Figure 14**

*Database Dataflow*

Chart

Description automatically generated

*Note.* A diagram showing how data flows in database. Own work.

The data flow discussed in this section and demonstrated in the illustration above, by the table linking arrows, is enforced by the Python socket server.

## 5.2 Socket Server

A lot of effort went into determining the best way to facilitate the communication between the modules and the server, leading to several weeks of research and testing. The first design involved using HTTP requests but upon further research it became clear that this methodology would result in extreme latency. A Request for Comments publication by Loreto, et al. describes the issue:

In the standard HTTP model, a server cannot initiate a connection with a client nor send an unrequested HTTP response to a client; thus, the server cannot push asynchronous events to clients. Therefore, to receive asynchronous events as soon as possible, the client needs to poll the server periodically for new content. However, continual polling can consume significant bandwidth by forcing a request/response round trip when no data is available. It can also be inefficient because it reduces the responsiveness of the application since data is queued until the server receives the next poll request from the client.

This led to a new design using TCP sockets. This would allow for a constant host to host connection where the server could service multiple connections and have bidirectional communication with each of the modules *(Information Sciences Institute, 1981, p. 5)*. A PHP socket server was created as the first version but due to PHP being primarily focused on rendering hypertext *(The PHP Group, n.d.)* the final iteration of the server was developed using Python3.

The server backend uses the Python socketserverframework which simplifies the process by using a collection of concrete server classes (*Python Software Foundation, n.d.*). The framework keeps code organized and it eased the designing of how the sockets interact. When a client module connects, it must first register itself by sending its identifier, description, and IO type. Based on the IO type, the server initiates the communication processing in either an input or an output processing thread. Each processing thread has slight differences.

Input processing in the socket server starts by taking input data from the client module which until a newline character is received. The newline character represents the end of a message from the client. This design choice was inspired by strings in the C family of programming languages, which are terminated by a null character *(Ritchie D., 1993)*. When a full message, which consists of the module’s new state, has been received, the server queries the database and gets all the “Input” table rows linked to the input module’s identifier. The server then updates all each row’s “Trig” value by comparing the module’s new state to the required “OnVal” value by using the comparison symbol saved in the “Sign” column. The server then checks if any triggers in the “Circuits” table need to be updated as well based on groups of “Input” table triggers being activated. If the input module’s new state no longer fulfils the trigger requirements, the trigger is deactivated.

Output processing in the socket server revolves around the data connection routes formed in the database between the “Circuits”, “CircuitToInput”, and “CircuitToOutput” tables. When an output module requests the server to check if its state is destined to change, the server queries the database for any “Circuit” table entries with active trigger states linked to the output module’s identifier. If a valid row is found, the output module’s state is updated to be the value in the “SetValue” column of the “Output” table entry.

## 5.3 Rest API

A PHP REST API allows interactions between the client website and the backend server. Via Ajax requests, the client accesses data through one of two REST directories; one being for module information management, including the program uploading, and one for a few user account functionalities, such as the saving and loading of the program files. The create account and login pages were isolated from the REST API to keep them simpler and avoid the need for Ajax on those pages.

## 5.4 Challenges

The biggest challenge with creating the socket server was time constraints on research and making sure design decisions were flexible should the need to adjust them arise.

When building the database, generating a robust design for data storage and routing was a challenge. There are countless possibilities for the database to be configured meaning it took many days of research and discussion to determine a suitable design.

While creating the REST API, an interesting bug was detected. The REST API was calling functions from another folder on the website which in turn was linked to a database connection PHP file. The database file was not linking properly, and it was determined that the path was incorrect as it was based off the relative path from the functions file (ie. “../db.php”), rather than the relative path to the REST file that accessed the functions (ie. “../../db.php”). This was remediated by modifying the relative path to the correct version. In hindsight, an absolute path would have been a wiser option.

## 5.5 Reflections

In its current form, the backend has some design flaws that prevent it from scaling well. One of the biggest offenders is the constant polling of the database and altering done by the server, this can be made more effective by using SQL Triggers *(Oracle Corporation and/or its affiliates, n.d.)*.

No protocol exists in the prototype version to determine what state an output module should update to if there are conflicting activated circuits leaving the system’s path of action indeterminable. It’s currently up to the user to write programs that will not conflict. Should a future revision of the project be created, this issue would be studied in more detail.

# 6.0 Conclusion

The IZ-Kit is the perfect home automation setup kit for automation enthusiasts. It is extremely modular, allowing for unlimited input and output possibilities in both hardware and software forms. Multiple pins and varying types of pins, such as the GPIO, I²C, and analog pins on the microcontrollers add to the vast assortment of sensors and actuators that can be used. The implementation of a custom programming language to the kit allows the user to precisely determine the routing between input and output modules resulting in home automation system set up exactly to the specifications of the user. The front-end website is packed with helpful features to support the development of automated systems, and the socket server and database allow the project to route communications of these systems quickly and efficiently. The IZ-Kit has great potential, and with a few optimizations to the prototype version it would make an excellent home automation setup or education system.

# References

Arduino. (n.d.). *WiFiBegin*. Arduino. https://www.arduino.cc/en/Reference/WiFiBegin.

ExceptionsHub Admin. (2017, November 19). *How to handle in textarea?.* ExceptionsHub. https://exceptionshub.com/how-to-handle-tab-in-textarea.html.

Information Sciences Institute, TRANSMISSION CONTROL PROTOCOL (1981, September). <https://tools.ietf.org/html/rfc793>

Keary, T (2020, December 10) *Network Topology: 6 Network Topologies Explained and*

*Compared*. Comparitech.

h<ttps://www.comparitech.com/net-admin/network-t>opologies-advantages-disadvantages/

Loretto et al. (2011, April). *Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP*. http://www.rfc-editor.org/rfc/rfc6202.html.

Lueth, K. L. (2020, November 19). *The State of IoT 2020: 12 Billion IoT Connections, Surpassing*

*Non-IoT*. IoT-Analytics. <https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/>

Oracle Corporation and/or its affiliates. (n.d.). *MySQL 5.6 Reference Manual:: 20.3 Using Triggers*. MySQL. https://dev.mysql.com/doc/refman/5.6/en/triggers.html.

Python Software Foundation. (n.d.). *Socketserver - A framework for network servers*. https://docs.python.org/3/library/socketserver.html.

Ritchie, D. (1993, April). *The Development of the C Language*. Bell Labs. https://www.bell-labs.com/usr/dmr/www/chist.html.

The PHP Group. (n.d.). *What is PHP?.* <https://www.php.net/manual/en/intro-whatis.php>.

Wemos. (n.d.). *LOLIN D1 mini — WEMOS documentation*.

<https://www.wemos.cc/en/latest/d1/d1_mini.html>

Zeigler et al. (2018). *Theory of Modelling and Simulation: Discrete Event and Iterative System Computational Foundations*. ProQuest eBook Central. https://ebookcentral.proquest.com.

[Difference Between Star and Mesh Topology] [Illustration]. (n.d). TechDifferences.

<https://techdifferences.com/difference-between-star-and-mesh-topology.html>

[Hand Holding ESP2886 NodeMCU] [Image]. (n.d.). Amazon. <https://images-na.ssl-images->

amazon.com/images/I/61sU-gLZncL.\_AC\_SL1001\_.jpg

# Appendix A

## Bill of Materials

The materials required for IZ-Kit’s circuits are as follows in the table below.

**Table 1**

*Bill of Materials*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Manufacturer Part Number | Description | Quantity | Reference Designator | Cost | Extended |
| N/A | WEMOS D1 Mini MICROCONTROLLER | 5 | U1 | $5.67 | $28.35 |
| RL5-B5515 | 5mm BLUE LED | 1 | LED1 | $0.28 | $0.28 |
| CF1/4W221JRC | 220OHM, 1/4W, 5% | 1 | R1 | $0.08 | $0.08 |
| CF1/4W102JRC | 1KOHM, 1/4W, 5% | 1 | R1 | $0.08 | $0.08 |
| CF1/4W103JRC | 10KOHM, 1/4W, 5% | 1 | R1 | $0.08 | $0.08 |
| B3F-4055 | Tactile Switches 12x12mm | 1 | S1 | $0.61 | $0.61 |
| GM5539 | Photoresistor 5mm | 1 | R2 | $0.25 | $0.25 |
| HW479 | RGB Module Board | 1 | U2 | $0.38 | $0.38 |
| TM162A-3 | 16X2 Character Monochrome LCD | 1 | U2 | $1.91 | $1.91 |
| N/A | LCD I2C Backpack | 1 | U3 | $0.74 | $0.74 |

*Note.* Table of IZ-Kit materials and prices. Own work.

# Appendix B

## Circuit Schematics

The circuit schematics of the IZ-Kit modules are included below.

**Figure 15**

*LED Module Schematic*

Table

Description automatically generated

*Note.* Circuit schematic for the IZ-Kit’s LED module. Own work.

**Figure 16**

*LCD Module Schematic*

Table

Description automatically generated

*Note.* Circuit schematic for the IZ-Kit’s LCD module. Own work.

**Figure 17**

*Light-Sensor Module Schematic*

Table

Description automatically generated

*Note.* Circuit schematic for the IZ-Kit’s light sensor module. Own work.

**Figure 18**

*Pushbutton Module Schematic*

Diagram, table

Description automatically generated

*Note.* Circuit schematic for the IZ-Kit’s pushbutton module. Own work.

**Figure 19**

*RGB LED Module Schematic*

Table

Description automatically generated

*Note.* Circuit schematic for the IZ-Kit’s RGB LED module. Own work.