Milestone 4: Final Paper

CSCE 838 – Internet of Things

University of Nebraska-Lincoln

December 16, 2022

Course Project: IoT Escape Room

Team 3 Members

Marcus Gubanyi

Rehenuma Tasnim Rodoshi

Rezoan Ahmed Nazib

Samuel Murray

Sepehr Tabrizchi

Version	Date	Changes	Comments
MS0	10/19/22	Initial version of the project specification: • Introduction • Initial customer requirements • User Guide • Bill of materials	Project proposal
MS1	11/01/22	Added the following: • Engineering requirements • Implementation Style Sheet • Test Plan • Schedule and Staffing Plan	Gate Review
MS2	11/22/22	Updated sections from prior versions. Added the following: • Test Results • Architecture • System Design • Software Development Plan • Individual Progress Reports with Bug List	Mid-Term Gate Review
MS3	12/05/22	Updated following sections from prior versions: • Test Plan and Results • Architecture • Software Development Plan	Pre-Demo Update
MS4	12/16/22	Finalized all sections of paper based on final system specification and implementation	Final Paper

The project repository, containing code, videos, and a 3D model, can be found at the following GitLab URL: https://git.unl.edu/mgubanyi/csce-838-escape-room

Email mgubanyi2@unl.edu if you cannot access the repository.

1. Introduction

We will develop an IoT system to enhance Escape Rooms. According to Wikipedia, an escape room is a game in which a team of players discover clues, solve puzzles, and accomplish tasks in one or more rooms to accomplish a specific goal in a limited amount of time. Typically, the goal is to escape. A key element of high-quality escape rooms is an immersive environment which captivates the players. Players might be stuck on a dysfunctional submarine deep in the ocean and needing to find a way home, or players could be trapped in a haunted house, desperate to escape.

Our IoT system will incorporate Enchanted Objects into Escape Rooms to serve two purposes: to create an immersive, magical environment, and to track data relevant to gameplay. By building a variety of embedded devices that communicate with a cloud-based controller, all the enchanted objects in the escape room are connected and data can be easily harvested.

Escape Room businesses oftentimes do not have expertise to implement such IoT systems. Our system will provide plug-and-play Enchanted Objects that can be incorporated into existing Escape Rooms or built into new Escape Rooms. Two kinds of Enchanted Objects are Trigger devices and Activator devices. Businesses will purchase specific Enchanted Objects that can interact with each other, along with a subscription. This TaaS (technology as a service) business model will help Escape Room businesses enhance their rooms without stressing about the technology.

In this document, we provide the customer requirements of the system, a user guide for businesses seeking to use our system, engineering requirements, the implementation style guide, the testing plan, and the schedule and staffing plan. The document will be updated throughout the semester.

2. Customer Requirements

Our two deliverables are Enchanted Objects and the Cloud Infrastructure that connects and controls the objects. In this section, we provide high-level descriptions for these subsystems and verifiable customer requirements in the format of *C.# Customer Requirement*.

2.1 Enchanted Objects

The enchanted objects can be classified into two generic categories: triggers and activators. Trigger devices collect data from sensors and send the results to the cloud. Activator devices receive commands from the cloud and perform some action to change the escape room environment or give information to the players. These Triggers and Activators will be incorporated into the Escape Room's clues, puzzles, and tasks, governed by a predetermined procedural flow set by the escape room designer. The devices for the enchanted objects do not communicate directly to each other. Instead, they connect to a cloud-based controller.

C.1 – Light Sensor Device (Trigger)

A light sensor sends a signal to the cloud when a configuration light level is sensed. An example way to trigger this device would be to shine a flashlight on a portrait with the sensor mounted behind the paper.

C.2 – Switched Outlet (Activator)

A switched outlet is normally open but can be triggered to turn on when receiving a command from the cloud. An example way to use this device would be to turn on a lamp after a cloud command.

C.3 – Push Button Device (Trigger)

This device is as simple as it gets – when the button is pressed, a signal is sent to the cloud. An example way to use this device is to fasten the button to an object, such as a book. When the object is moved, the button is pressed/unpressed.

C.4 – LEDs/Candles (Activator)

These simple LEDs (possibly in the form of electric tea candles) can light up in response to triggers that a player induces. The LEDs can be configured together to give clues as well, depending on the pattern of their on/off cycles together.

C.5 – Step Device (Trigger)

The step device is a floor mat with electrode plates beneath it (concealed from the players). When the players step on the floor mat, the electrodes close a circuit in the detecting electronics, which can report to the cloud when the mat is stepped on. With several step devices, a puzzle can be set for the players to step out a particular pattern to trigger the next part of the sequence.

C.6 – Raspberry Pi or Any Computer (Activator or Trigger)

A raspberry pi connected to the monitor and keyboard can act as a Trigger and/or an Activator device. Use cases include showing clues or receiving codes from the keyboard.

2.2 Cloud Infrastructure

The core of the system lives in the cloud. Both setup configuration and flow configuration are specified in the cloud infrastructure. This allows for adjustments to be made to the system more easily, without editing, compiling, and uploading code on the actual devices. It also allows the system to be modular such that different setups, triggers, activators, sequences, and puzzles can be implemented without changing the software on the devices.

During device setup, the device requests setup configuration from the cloud. This configuration determines the sensor readings for which a trigger device must send a signal to the cloud. This setup configuration minimizes communication between the sensor devices and the cloud. Each device, both

trigger and activator devices, send system data to the cloud periodically. These features allow the technical aspects of the system to be managed and monitored remotely.

Escape room businesses design escape rooms that flow from one state to the next. Enchanted objects will play a critical role in this flow as they are user-facing, but the cloud determines how trigger devices connect with activators. Trigger devices send signals to the cloud, the cloud parses it's flow configuration to determine which activators to activate, and lastly the cloud sends commands to the appropriate activators.

An example scenario for this system is outlined below:

1) Players are given clues to shine a flashlight on a picture hanging on a wall.
2) A light sensing trigger device (C.1) sends a signal to the cloud.
3) Using the flow configuration, the cloud sends a command to an LED/Candle activation device.
4) The LED/Candle device blinks the LED in a specific pattern as a new clue for players.

Typically, clues in the escape room lead players to trigger devices and activation devices provide players with more clues.

C.7 Setup Configuration

During start-up for any device, the device sends a request to the cloud to send configuration for that device. The configuration includes: frequency of heartbeat and sensor thresholds for when to trigger.

C.8 Heartbeat

Each device periodically sends a heartbeat to the cloud to track the device's runtime behavior. Device data sent to the cloud includes: last start-up time, error logs, and sensor readings.

C.9 Data Log

The cloud logs data for each enchanted object. Data includes trigger times and heartbeat data. See section 2.5 Usage Data for more details.

C.10 Flow Configuration

Unique to each escape room, the customer provides a configuration for how trigger devices interact with activator devices. The configuration format is a set of ordered pairs of the form *(TriggerID, ActivatorID)*.

C.11 Cloud Triggered Activation

When a trigger device sends a signal to the cloud, the cloud sends a command to the appropriate activator device. An example is the push button signaling to the cloud and the switched outlet receiving a command to turn on power.

2.3 Customer Q&A

We conducted a Q&A session with Jacquelyn Schranz, an escape room specialist from Seward Library. She builds escape rooms multiple times per year in the library, free for the Seward community. Jacquelyn, who is a prospective customer for our IoT system, answered the following questions.

After reading the introduction and enchanted objects in our project proposal, do you think our proposed tech system would enhance an escape room experience? Do you have any concerns?

> "I really like this idea! (the name is pretty cool too) I have kept our escape rooms pretty low tech since we don't have a large budget for the rooms, the ease of use for staff members running the rooms, and also because I didn't want to run into major issues with something not working and we can't fix it before more groups come in. I think stressing the plug and play idea would be a big selling point for these – easy to set up and easy to maintain! Am I correct in thinking that these Enchanted Objects would be purchased outright, not "rented?" Just curious.
>
> I've been to some escape rooms in Orlando made by the company Escapology that were fantastic and then I have been to others here in Nebraska that weren't so great. They all used technologies of some sort such as RFID tags, electromagnets, lights etc. but I think what made Escapology's rooms so much better is that they did a great job of "embedding" the technology into the theming of the room and you never knew what "effect" was going to happen after solving a puzzle (door opens, panel in wall opens, light starts flashing). The tech also worked flawlessly whereas the one room we went to in Lincoln had a puzzle near the end that used morse code and although we did the morse code right, the light never came on. The gamemaster had to come in and fix the issue which made the end of the game anti-climatic. It's great that you have this variety of objects to use, but it is also important that they work well all the time."

As someone who builds and runs escape rooms, what data would be useful to track for an escape room? For instance, would it be helpful to track usage data such as the time between each phase? What about user data like age, gender, etc.?

> "For our previous rooms, I have only tracked time remaining and how many people went through the room. For professional escape rooms, keeping track of escape rate percentage is important so that people who are wanting to go through a room have a rough idea of how easy the room is to escape. It may also help to know how many people were in the group compared to their time left. I also like to know who the gamemaster is each day so that if I for example keep seeing a lock not fully locked or something wrong, I know who to talk to. I wouldn't use user data for much, though if I were to run a room longer term, knowing the ages might be helpful for making minor room adjustments for difficulty."

Do you have any interesting ideas for using technology in an escape room?

> "Electronic locks – all of the locks we use are good old metal locks with dials or keys. Perhaps a lock could be created that can only be unlocked once a puzzle is solved (and it would automatically unlock itself)

I always appreciate puzzles that require different materials from around the room to solve such as your monitor and glasses example.

It would be great to have a device that would change lighting in the room based on what a player manipulates. For our children's pirate room, I had one puzzle that used glow in the dark stars and a black light so I had to turn off the lights in the room, turn on a wave light / sound projector, and then turn off the music on the cd player. It causes an awkward pause in the game and players just had to stand there with their eyes closed. Having this done automatically would be nice. A color change in light could also correspond with the storyline of the game.

I have wanted to try some kind of puzzle that involves plugging in wires and then pressing a button to see if the puzzle is correct – like a light turns green. Could any of your objects do something like this?"

Would you be interested in incorporating our system into your escape rooms at the Seward Library?

"Yes, I would love to test it out and see how it works! I have not started on next summer's themes yet so I'm willing to try some new tech and work it into a story."

If you are willing to, please send an example flow chart for an escape room.

"See the attached PDFs*. It includes the flow chart and puzzle descriptions for our submarine room "Under Pressure" from summer 2022."

**Jacquelyn shared PDFs for detailing puzzles, story narrative, and flow chart for an escape room. These documents are helpful for understanding and can be provided to the instructor upon request.*

2.4 User Data

The initial version of the system will not track user data. Future versions may track relevant user data such as age category of players.

2.5 Usage Data

Our proposed system will track a variety of usage data. The usage data will serve two purposes: inform escape room designers and debug/fix issues with the technology. Escape Room designers can use data to help make the experience as fun as possible. An Escape Room is not fun if it is too easy, too challenging, or takes too long.

Usage data relevant to escape room designers:

1. Time between phases of the escape room
2. Number of failed attempts for a puzzle or task

Usage data relevant to management of IoT system:

- Power on timestamps
- Reset data and timestamps
- Sensor readings to inform configuration

3. User Guide

3.1 Example Escape Room Narrative

In high-quality escape rooms, the puzzles are not linear. But for the sake of this project, we developed a simple, linear escape room to illustrate the use of our IoT technology in an escape room setting. The following steps depict how the Enchanted Objects could be used.

- To begin, the players are in a dark room with flashlights.
- Players are given clues leading them to shine a flashlight on portraits hung on the wall.
- Light sensor sends a trigger signal to cloud.
- Cloud sends command to Switched Outlet to turn on.
    - Unpolarized monitor turns on as a white screen.
    - The lamp turns on to brighten the dark room. The lamp reveals two chests, but they are locked and require codes.
- Clues lead players to pull a book off a shelf. A push button device sends a signal to the cloud.
- Cloud sends command to LED/candle devices:
    - The book pulled off the shelf is a book on binary. Book gives 4-bit binary numbers associated with 10 decimal digits. Hints that binary bits 0 & 1 correspond to off & on.
    - A row of LED's gives the code for one of the chests by cycling through digits in binary.
- Opening the chest with this code reveals a pair of sunglasses.
    - The sunglasses polarize the light from the monitor.
    - The monitor provides a clue.
- In the corner of the room, a player stands on the Step Device triggering a signal to the cloud.
    - Cloud sends signal to the monitor to Raspberry Pi (connected to the monitor) to change its display.
    - While the step device is stepped on, the monitor displays the code that opens the other chest which reveals magnificent treasure!

3.2 Guide to Incorporate IoT System into an Escape Room

Whether a business has an existing Escape Room or is designing one from scratch, follow these steps to incorporate our IoT system:

1. Create an account on the website.
2. Explore the catalog of available Enchanted Objects that fit with your current escape room.
3. Add desired Enchanted Objects to your cart and go to checkout.
4. Specify the configuration of how each Activator device is triggered by Trigger devices.
5. Select a subscription for cloud support based on how long the Escape Room will be deployed.
6. Place order after confirming purchase of hardware and cloud software subscription.
7. Once Enchanted Objects arrive, follow the instructions to turn them on and connect to WiFi.
8. Activate the cloud by logging into your account.
9. Verify the Enchanted Objects are connected to the cloud and test the system.
10. The IoT system of Enchanted Objects is now operational.

4. Engineering Requirements

E.1 Generic End-Device Requirements

The role of the Generic End-Device is to handle LoRaWAN communication within the system. In the initial version of this system, each Generic End-Device is a SparkFun Pro RF-LoRa (SamD21) board. A Generic End-Device could contain multiple trigger or activation devices. For example, one Generic End-Device could be connected to a 4-channel switch outlet and a light sensor, resulting in 4 activator devices and 1 trigger devices all in one Generic End-Device.

E.1.1 Network Membership

Each end-device must join the LoRaWAN network upon setup.

E.1.2 TX Message Identification

Each end-device must have a unique end-device ID that is included in every TX message the device sends.

E.1.3 RX Message Identification

Each cloud RX message includes an end-device ID for the intended recipient. End-devices only process messages if they are the intended recipient.

E.1.4 Heartbeat

End-devices send heartbeat messages to the cloud, through the gateway, periodically. The heartbeat message includes power-on timestamps, reset data and timestamps, sensor readings, and error logs.

E.1.5 Contained Devices

End-devices must be able to contain one or more trigger devices and/or one or more activator devices.

E.1.6 Contained Device Identification

Each trigger and activation device attached to the end-device must have a unique ID. During setup, the end-device sends a list of all attached devices with their respective IDs to the cloud.

E.1.7 Generate Errors

End-device generate errors to send as part of heartbeat. Errors include: watchdog timer reset, missing packing, trigger device errors, and activation device errors.

E.1.8 Watchdog Timer

Each end-device must include an active watchdog timer.

E.1.9 Reset

End-devices receive a reset command from the cloud, when resets the system to its original state.

E.2 Gateway Requirements

A single gateway provides the bridge between end-devices and the could server. It establishes the LoRaWAN network, allows end-devices to join the network, connects to WIKI, forwards messages between the end-devices and the cloud, and sends its own heartbeat data to the cloud. An alternative version of the system could work without a gateway if each end-device connects to WIFI. This alternative system could be achieved with Raspberry Pi Pico W boards.

E.2.1 Gateway Network Setup

The gateway device must establish a LoRaWAN network for end-devices.

E.2.2 WIFI Connected

The gateway device must connect with a WIFI network.

E.2.3 Message Forwarding

The gateway device must forward messages from the cloud to the end-devices and from the end-devices to the cloud.

E.2.4 Gateway Heartbeat

The gateway device must send a heartbeat to the cloud for itself, periodically. The heartbeat message includes power-on timestamps, reset data and timestamps, and error logs.

E.3 Generic Trigger Device Requirements

A generic trigger device is one that reads sensory data and reports it to the cloud via the LoRaWAN network. The trigger transmissions to the cloud must indicate the parent end-device ID and the specific trigger device ID (since there may be more than 1 trigger per end-device). The trigger transmission must also contain the sensor type (e.g. “button”), and the sensor data (e.g. “pushed” or “released”). The trigger device must wait for the command from the cloud to begin the escape room, and then transmit the initial sensor data. When the command arrives that ends the escape room, the trigger stops sending data.

E.3.1 Cloud-based Trigger Configuration

During setup, the trigger device sends a request to the cloud for its configuration. The cloud responds with configuration for when to send a trigger signal. The configuration includes the frequency of sensing and the conditions on which to send a signal to the cloud.

E.3.2 Trigger Signals

Trigger devices periodically read sensory data and sends a signal to the cloud when conditions are met.

E.3.3 Stop Command

Trigger devices cease sensing data and wait to be reset by their end-device. The stop command is used when the trigger device is no longer needed for the remainder of the escape room.

E.4 Generic Activator Device Requirements

A generic activator device is one that receives a command from the cloud via the LoRaWAN entwork to take some action. This activation is modeled as a state-based system with the cloud having the capability to both get and set the state of the activator device.

E.4.1 Set State Command

Receive cloud command to set state of activation device.

E.4.2 Get State Command

Receive cloud command and return current state of activation device to cloud.

E.5 Light Sensor Trigger Device Requirements (from C.1)

The light sensor inherits the properties of a generic trigger device (from E.3). It begins reporting the light sensor data to the cloud when the escape room begins. The software must include drivers to communicate with the light sensor on-demand to read off the data via a serial connection.

E.5.1 Signal at Light Threshold

Based on its cloud-based setup configuration, send a signal to cloud when light level condition is met.

E.5.2 Adapt to different room light condition

The device triggering condition should adapt with the lighting condition of the escape room.

E.5.2 Suitable housing for triggering

The device should trigger and detect the presence of the flash light even though it will be hidden behind a picture.

E.6 Switched Outlet Activator Device Requirements (from C.2)

The switched outlet inherits the properties of a generic activator device (from E.4). It waits for commands from the cloud that tells it to turn off or on its outlet and toggles a GPIO pin to control the relay attached to the outlet accordingly.

Each individual socket constitutes one switched outlet activator device. Hence, an outlet box with 4 sockets constitutes 4 unique activator devices.

E.6.1 Set Power State

Set power state of outlet by toggling GPIO pin on end-device.

E.7 Push Button Trigger Device Requirements (from C.3)

The push button inherits the properties of a generic trigger device (from E.3). It begins reporting the state of the button to the cloud when the escape room begins.

The software must debounce the state of the pushbutton continually. When a debounced state change is detected, the device sends a message to the cloud with the new state.

E.7.1 Signal on Pushdown/Release

Send signal to cloud upon button pushdown, release, or both, based on setup configuration.

E.8 LED/Candle Activator Device Requirements (from C.4)

The LED/candle inherits the properties of a generic activator device (from E.4). It waits for commands from the cloud that tell it to turn off or on and toggles a GPIO pin connected to the LED. When

Each individual LED/candle constitutes one switched outlet activator device.

E.8.1 Blink Configuration

Configuration received from the cloud at configuration will determine whether the LED blinks in its on state. If so, configuration will specify the blink pattern.

E.8.2 Set State

Set the on/off state of LED by toggling the LED's GPIO pin on the end-device. On state includes blink patterns.

E.9 Step Trigger Device Requirements (from C.5)

The step sensor inherits the properties of a generic trigger device (from E.3). Each step device constitutes 1 trigger device. The software must debounce the state of the step continually. When a debounced state change is detected, the device sends a message to the cloud with the new state.

E.9.1 Signal Step

When a step is sensed, send a signal to the cloud with the new state.

E.10 Raspberry Pi Requirements (from C.6)

An application running on a Raspberry Pi or other computer will emulate an IoT device from the perspective of the cloud infrastructure. The python program can send data to the cloud based on user input from a peripheral such as the keyboard. The program can also receive commands to the cloud to change its display. Note that this enchanted object does not inherit from the base classes.

E.10.1 Cloud-based Application Configuration

Receive configuration from the cloud on when to send heartbeat and user input signals.

E.10.2 Graphical User Interface Activated

Display a graphical user interface to the user through a monitor. The user interface would have visual components that change based on commands from the cloud, simulating an activation device.

E.10.3 User Input Signaling Cloud

Provide means of receiving user input via keyboard and mouse, simulating a trigger device that signals the cloud when conditions are met.

E.11 Cloud Infrastructure Requirements (from C.7, C.8, C.9, C.10, C.11)

The cloud here is the controlling point between the trigger devices and activation devices. The whole system relies on the cloud to make the flow of the escape room. The cloud is responsible for sending and receiving commands and controls the process. The cloud start receiving and collecting necessary data when the escape room begins.

One example of cloud server that can be used to send, receive and collect data is Microsoft Azure. The cloud should have enough space to store usage data or some other necessary data, if required.

E.11.1 Setup

Build an Azure function for setup of devices. During setup, each device sends the setup command to the cloud and the cloud responds with that device's configuration. Each trigger contains an initial configuration in the cloud on how to sense and send data to the cloud.

E.11.2 Heartbeat

Build an Azure function for logging the heartbeat of devices. Periodically, every device in the system will send a heartbeat. See E.1.4.

E.11.3 Trigger

Build an Azure function for receiving trigger signals and sending activation commands. Trigger devices call this Azure function and the function sends commands to activation devices based on the flow configuration of the escape room. See E.11.

E.11.4 Complete

Build an Azure function for completing the escape room. The function commands all end-devices to stop sensing.

E.12 Flow Configuration Requirements (from C.10)

Used within the Trigger Azure Function (E.10.3), the flow configuration determines the commands for activation devices after the cloud receives a trigger signal. The flow configuration is a JSON file where each trigger device has a JSON object within the file. The activation devices and their commands are configured for each trigger device.

E.12.1 Cloud-based Flow Configuration

A JSON configuration file lives within the cloud and determines the commands for activation devices upon receiving a trigger signal.

5. Implementation Style Sheet

The implementation style sheet is intended to maintain uniformity throughout the codebase for this project. The following rules will govern our implementation:

- Each program file must have a comment header including the following at a minimum:
    - Title
    - Description
    - Lead Developer
- Arduino Sketch File Naming:
    - ExampleName###_main.ino
    - The optional digits (###) would be used to distinguish between multiple end-devices of the same kind. For instance, two end-devices could have LEDs attached.
- Python code should be written according to PEP8 standards.
- C++
    - Class Names: PascalCase
    - General Variable Names: camelCase
    - Activation Device Variable Names: Type#_activation
    - Trigger Device Variable Names: Type#_trigger
        - Type identifies the type of device such as LED or Temp.
        - # is the ID of the subdevice (not the endnode)

6. Test Plan

The testing procedure will follow in three phases: unit testing, integration testing, and system testing. Unit and integration testing will verify that the engineering requirements are met. System testing will confirm that the customer requirements are met.

Note that we are not using test-driven development for this project. Thus, we have not yet determined all our tests yet. We will develop and perform more tests as system components are completed.

6.1 Unit Testing

Each system component, including trigger devices, activation devices, end-devices, the gateway, and the cloud will be tested individually to verify they function correctly in isolation. We first list the completed unit tests and then the engineering requirements left to unit test.

Round 1 of Completed Unit Tests

Component (Requirement)	**Test**	**Result**	**Comments**	**Personnel, Date**
Abstract node class (E.1.2)	Serialize initial state data	pass	JSON generated successfully	Sam 11/12/2022
Abstract activation device class (E.4.1)	Set state of device through class	pass	SetState function pointer allows state to be stringified and parsed	Sam 11/12/2022
Abstract activation device class (E.4.2)	Read state of device through class	pass	GetState function pointer works too	Sam 11/12/2022
Abstract activation device class (E.1.7, E.4.1)	Error handling of bad set state	pass	Fails gracefully, stringifies error code, puts it into JSON, queues it for next transfer	Sam 11/12/2022
Abstract activation device class (E.1.7, E.4.2)	Error handling of bad read state	pass	Fails gracefully, stringifies error code and puts it into JSON correctly, queues it for next transfer	Sam 11/12/2022
Abstract trigger device class (E.3.2)	Read state of device through class	pass	Gets the state of (temperature) device and stringifies it	Sam 11/12/2022
Abstract trigger device class (E.1.7, E.3.2)	Error handling of bad read state	pass	Fails gracefully, stringifies error code and puts it into JSON correctly, queues it for next transfer	Sam 11/12/2022
Abstract node class (E.1.2)	Deserialize JSON string	pass	Data extracted successfully	Sam 11/13/2022

Abstract node class (E.1.5)	Use JSON to set particular activation device's state	pass	Sets desired activation device state and no other device	Sam 11/13/2022
Abstract node class (E.1.3)	Ignore JSON not addressed to node	pass	If node ID in JSON does not equal self node ID, ignore	Sam 11/13/2022
Abstract node class (E.1.7)	Ignore repeated packets	pass	If packet ID is not > than prev packet ID, generated warning and ignored it	Sam 11/13/2022
Abstract node class (E.1.2)	JSON string requests activation/trigger device state	pass	Queued a request to send device state, send it thereafter	Sam 11/13/2022
Abstract node class (E.1.3)	Ignore requests to set/get state of nonexistent activation/trigger device state	pass	When ID and type of device from JSON do not match any devices on the node, does nothing	Sam 11/13/2022
Abstract node class (E.1.7)	Node ID too far into future (indicating there was a missed packet)	pass	Generates an error code, but still processes JSON and executes any requests	Sam 11/13/2022
Abstract node class (E.1.7)	Handling of improperly formatted JSON strings	pass	Generates error code, does nothing	Sam 11/13/2022
Abstract node class heartbeat (E.1.4)	Send heartbeat if a TX hasn't occurred after a timeout	pass		Sam 11/13/2022

Round 2 of Completed Unit Tests

Component (Requirement)	**Test**	**Result**	**Comments**	**Personnel, Date**
Light Sensor (E.5.1)	Verify in serial monitor that light sensing level changes when shining a flashlight on picture with sensor behind.	pass	In tests, the sensor readings were around 5 without flashlight and >50 with flashlight.	Nazib 12/3/22
Light Sensor (E.5.2)	Verify the light sensor does not get triggered just by changing the room light condition	pass	The threshold for the flashlight was set to high so normal light condition does not trigger the light sensor	Nazib 12/3/22
Light Sensor (E.5.3)	Verify the light sensor is getting enough light even after the	pass	The internal condition of the housing was kept dark and the front side was white. So, the housing let enough light to go	Nazib 12/3/22

	housing/installing inside the frame		inside and the light sensor could also trigger the flash light.	
Switched Outlet (E.6.1)	Verify the power turns on when receiving message.	pass		Sam 11/26/2022
Push Button (E.7.1)	Verify in serial monitor that a book on a shelf presses button and unpresses when removed.	pass	Used the SAMD21 internal pull-up resistor and in the input pin, instead of input, input-pullup was used to reflect this.	Rodoshi 12/3/2022
Raspberry Pi GUI Activated (E.10.2)	Verify the GUI is displayed when receiving the DisplayGUI command.	Pass	Integrated with the Azure IoT Central Command.	Marcus 12/8/2022
Raspberry Pi GUI Code Submitted (E.10.3)	Verify the Submit button activates	Pass	Integrated with the Trigger Azure Function.	Marcus 12/8/2022
Switched outlet (E.6)	Safety test: check for exposed hot wires, correct wiring	pass	No visible problems. Ohmmeter shows outlet cover is electrically connected to ground pin on plug.	Sam 11/26/2022
Switched outlet (E.6)	Outlet tests: connected outlet tester to each outlet in turn. Switched on, monitored for open hot, open neutral, reversed hot and neutral, open ground, etc.	pass	All four outlets are wired correctly and can be switched on and off via GPIO pins	Sam 11/26/2022
Switched outlet (E.6)	Sent JSON string over LoRa to turn on each outlet	pass	Response time approx 0.5 seconds	Sam 11/26/2022
Switched outlet (E.6)	Sent JSON string over LoRa to turn off each outlet	pass		Sam 11/26/2022
Step trigger (E.9)	Tested resistance when not stepped on	pass	Ohmmeter overflow	Sam 12/8/2022
Step trigger (E.9)	Tested resistance when stepped on	fail	Holes in substrate too small to allow parallel sheets of aluminum to contact each other when stepped on	Sam 12/8/2022
Step trigger (E.9)	Tested resistance when stepped on	pass	Increased hole diameter	Sam 12/8/2022

Step trigger (E.9)	Tested debouncing of step trigger connected to SAMD21 board	pass	Noise debounced from trigger, one trigger per step and release	Sam 12/8/2022
Step trigger (E.9)	Integration of trigger drivers with generic end-node class test	pass	Sends state of step trigger when stepped on and released	Sam 12/8/2022
Watchdog Timer on End Nodes (E1.8)	Tested watchdog timer without kicking it	pass	System resets every 8 seconds	Sam 12/15/2022
Message Forwarding (E.2.3)	Print data received from LoRa over USB serial port	Pass	With this approach, receiving data from the node and passing it to the computer is checked.	Sam 12/15/2022
Message Forwarding (E.2.3)	Send data received from USB over LoRa network	Pass	Command received from USB and forwarded to device.	Sam 12/15/2022
Message Forwarding (E.2.3)	Use sample python code to send and receive data from serial.	Pass	Using Sam's python code, we were able to pass data to and from the Gateway using USB.	Sam 12/15/2022

Note that we are far behind in terms of testing. The focus of the last two weeks of the semester will be on integrating the entire system and testing it.

Engineering Requirements To Be Unit Tested

The following engineering requirements have not been unit tested yet. In most cases, this is because they have been completed. As soon as a requirement is completed, unit tests will be developed and performed.

- E.2 Gateway Requirements (partially unit tested)
  - E.2.1 Gateway Network Setup
  - E.2.2 WIFI Connected
  - E.2.4 Gateway Heartbeat
- E.5 Light Sensor (partially unit tested)
  - E.5.1 Signal at Light Threshold
- E.7 Push Button (partially unit tested)
  - E.7.1 Signal on Pushdown/Release
- E.8 LED/Candle
  - E.8.2 Set State
- E.10 Raspberry Pi (partially unit tested)
  - E.10.2 Graphical User Interface Activated
  - E.10.3 User Input Signaling Cloud

6.2 Integration Testing

Due to the nature of the system, some engineering requirements can only be tested by combining multiple system components. The following engineering requirements were tested with integration tests:

- E.1.1 Network Membership
- E.2.3 Message Forwarding
- E.3.1 Cloud-based Trigger Configuration
- E.8.1 LED Blink Configuration
- E.10.1 Cloud-based Application Configuration
- E.11 Cloud Infrastructure Requirements:
    - E.11.1 Setup
    - E.11.2 Heartbeat
    - E.11.3 Trigger
    - E.11.4 Complete
- E.12.1 Cloud-based Flow Configuration

Completed Integration Tests

Component (Requirement)	**Test**	**Result**	**Comments**	**Personnel, Date**
Network Membership (E.1.1)	Verify that each end-device join the LoRaWAN network upon setup	Pass		Sam 12/14/2022
Message Forwarding (E.2.3)	Verify if gateway device forward messages to cloud from end devices and vice-versa	Fail	Gateway does not connect to cloud. Only to a local python script with sample flow configuration.	Sam 12/14/2022
Cloud-based Trigger Configuration (E.3.1)	Test signals from triggers received and commands sent from cloud with correct message structure.	Pass		Marcus 12/8/2022
LED Blink Configuration (E.8.1)	Hard coded the turn on string command on the code manually after that I sent command using gateway to turn on and off the LED.	Pass		Sep 12/14/2022

Cloud-based Application Configuration (E.10.1)	Verify that the GUI is updated by configuration sent from the cloud.	Pass	The GUI's header, instructions, submit button text, and passcode are set by cloud.	Marcus 12/8/2022
Cloud Infrastructure Requirements (E.11)	Verify that an Azure Function deployed to the cloud can receive triggers and send commands.	Pass	Sending triggers to cloud from either the GUI application or a separate source both work correctly. Commands are sent.	Marcus 12/8/2022
Cloud-based Flow Configuration (E.12.1)	Verify that the flow configuration sends correct commands with each input trigger	Pass	Each input trigger {LightTrigger, ButtonTrigger, StepTrigger, PasscodeTrigger} sends the correct commands.	Marcus 12/8/2022
Step trigger (E.9) and end-node abstract class (E.1.2)	Multiple rapid steps	fail	Tries to send all step events, even if they happen while a prior step event is being transmitted	Sam 12/14/2022
Step trigger (E.9) and end-node abstract class (E.1.2)	Multiple rapid steps	pass	Prevented any further attempts to transmit if the end-node device is already transmitting, allowed again after end of transmission	Sam 12/14/2022
LoRa-USB Forwarding Gateway (E.2.1 and E.2.3)	Packet transfer from LoRa to USB and from USB to LoRa	fail	The original code for this did not work. So, it was scrapped and re-worked on the last day	Sam 12/14/2022
LoRa-USB Forwarding Gateway (E.2.1 and E.2.3)	Packet transfer from LoRa to USB	pass	Sam wrote a new SAMD21 program for doing this	Sam 12/14/2022
LoRa-USB Forwarding Gateway (E.2.1 and E.2.3)	Packet transfer from USB to LoRa	fail	Transmitted current packet and all previous packets on each reception of a USB serial packet	Sam 12/14/2022
LoRa-USB Forwarding Gateway (E.2.1 and E.2.3)	Packet transfer from USB to LoRa	pass	Gateway working well	Sam 12/14/2022

6.3 System Testing

System testing will be performed on the complete escape room after connecting all the devices to the cloud and beginning a play-through of a simple escape room. All unit tests and integration tests must pass before system testing. The focus of the system tests will be verifying all the customer

requirements. For each customer requirement, we list question(s) that must be answered while performing our system tests.

- C.1 Light Sensor Device
  - Does a flashlight shining on the light sensor trigger a signal to the cloud?
- C.2 Switched Outlet
  - Does the outlet's power turn on when the cloud command is sent?
- C.3 Push Button Device
  - Does lifting the book from on top of the push button send a signal to the cloud?
- C.4 LEDs/Candles
  - Does the LED turn on when the cloud command is sent?
- C.5 Step Device
  - Does stepping on the device send a signal to the cloud?
- C.6 Raspberry Pi
  - Does entering an input via keyboard/mouse send a signal to the cloud?
  - Does the display change when a cloud command is sent?
- C.7 Setup Configuration
  - Do devices request and receive initial configuration during setup?
- C.8 Heartbeat
  - Is heartbeat data received by the cloud?
- C.9 Data Log
  - Is usage data for the system viewable on the cloud?
- C.10 Flow Configuration
  - Do the appropriate activations occur after each kind of trigger?
- C.11 Cloud Triggered Activation
  - Are signals from triggers received and commands sent from cloud?

The final test was to be an Acceptance Test performed by subject matter expert, Jacquelyn Schranz.

Completed System Tests

Component (Requirement)	**Test**	**Result**	**Comments**	**Personnel, Date**
Light Sensor Device (C.1)	Test whether shining a flashlight on the light sensor trigger a signal to the cloud	Pass	Due to the change in the architecture, the triggering command was sent to the local gateway instead of the cloud	Nazib 12/14/2022
Switched Outlet (C.2)	Test whether the outlet's power turn on when the cloud command is sent	Pass		Sam 12/8/2022
Push Button Device (C.3)	Test whether lifting the book from on top of the push button	Pass	Due to cloud integration problem, the push button was tested to successfully send	Rodoshi 12/14/2022

	send a signal to the cloud		signal based on trigger to the local gateway.	
LEDs/Candles (C.4)	Test whether LED turn on when the cloud command is sent	Pass		Sep
Step Device (C.5)	Test whether stepping on the device send a signal to the cloud	Pass		Sam 12/8/2022
Raspberry Pi (C.6)	Test whether entering an input via keyboard/mouse send a signal to the cloud	Pass		Marcus 12/8/2022
Raspberry Pi (C.6)	Test whether display change when a cloud command is sent	Pass		Marcus 12/8/2022
Setup Configuration (C.7)	Test devices request and receive initial configuration during setup	Fail	Not implemented	Marcus 12/8/2022
Heartbeat (C.8)	Test heartbeat data received by the cloud	Pass	Passes only for Raspberry Pi device. The telemetry for other devices was not completed.	Marcus 12/8/2022
Data log (C.9)	Test whether usage data for the system viewable on the cloud	Fail	Not implemented. Telemetry is sent to cloud, but no visualization or cloud processing was implemented.	Marcus 12/8/2022
Flow Configuration (C.10)	Test appropriate activations occur after each kind of trigger	Pass	Flow configuration sends the correct commands based on the input trigger.	Marcus 12/8/2022
Cloud Triggered Activation (C.11)	Test signals from triggers received and commands sent from cloud	Pass	Posting a HTTP request to the cloud function sends commands to Raspberry Pi. Cloud only works with Raspberry Pi device.	Marcus 12/8/2022
Local Python Flow Controller (E.12)	Test command to turn on switched outlet	pass	Full control of switched outlet	Sam 12/14/2022
Local Python Flow Controller (E.12)	Test command to turn on LED	pass		Sam 12/14/2022

Local Python Flow Controller (E.12)	Test receive data from gateway	pass		Sam 12/14/2022
Local Python Flow Controller (E.12)	Test flow of triggers to activators	fail	LED not receiving commands consistently from Python controller	Sam 12/14/2022
Local Python Flow Controller (E.12)	Test flow of triggers to activators	pass	Sent repeat packets. If extra packets were received, they were ignored. This solved the problem of not receiving some of the packets that controlled the activation devices	Sam 12/14/2022
Entire system with local python flow controller	Escape room procedure	pass	The demo video shows the escape room working successfully! Note that the cloud was not implemented with the entire system.	Sam 12/14/2022

7. Schedule and Staffing Plan

7.1 Schedule

The implementation of this project will come in three phases: Planning, Development, and Testing. We will execute the plan iteratively rather than sequentially, with the understanding that most of the planning will occur first and most of the testing will be last. In the table below, we break down our weekly tasks for completing the project based on the engineering requirements and test plan.

As of December 5, we are behind on our schedule significantly. Development of each system component in isolation is complete. However, the system is not fully integrated. The cloud works with the Raspberry Pi GUI but the LoRaWAN network is not yet connected to the Raspberry Pi. Furthermore, we have not fully unit tested and not started integration or system testing.

Week	**Tasks**
As of December 16 *What we've completed so far*	Delegated development tasks. Completed abstract base classes (E1, E3, and E4). Unit test abstract base classes – all test passed using test devices. Designed flow configuration (E11) for a simple escape room. Complete E2. Complete E9 and E10. Complete enchanted objects (E5, E6, E7, and E8). Prepare demo presentation and finalize paper. Thoroughly document the final state of the implementation. Unit test cloud infrastructure (E10) and raspberry pi device (E9). Perform integration testing for the cloud infrastructure (E10) and flow configuration (E11) using one or more raspberry pi devices (E9). System test: test all engineering and customer requirements. Fine tune system and fix bugs. Unit test enchanted objects. Perform integration testing for the enchanted objects (E1, E3, E4, E5, E6, E7, E8) with gateway (E2).

	Test the gateway with a python script.
Incomplete Tasks	Fully implement and test the gateway with the cloud. Complete System Integration with Cloud, Gateway and LoRaWAN network. Finalize all testing, including acceptance testing.

7.2 Staffing Plan

The team is structured as a product owner and development team. While all team members will collaborate on all aspects of the project, each team member has individual responsibilities.

Product Owner

- Marcus Gubanyi
- Product Owner Responsibilities:
    - Determine the direction of the project, in collaboration with the entire team.
    - Lead writing efforts for documenting the project.
    - Communicate with subject matter experts and business representatives.
    - Contribute to the implementation of the project with the Development Team.

Development Team

- Rehenuma Tasnim Rodoshi, Rezoan Ahmed Nazib, Samuel Murray, Sepehr Tabrizchi
- Development Team Responsibilities:
    - Implement, document, test, and deliver the IoT system.
        - Device Code
        - Cloud Code
    - Design the framework for the system, writing abstract classes: Trigger, Activator, and End-Device.
    - Build each enchanted object.
    - Collaborate with product owner and each other on individual responsibilities.
- Individual responsibilities are assigned below based on who will take the lead role in developing specific engineering requirements.
    - Marcus Gubanyi:
        - *E.9 Raspberry Pi Requirements (E.9.1 - E.9.3)*
        - *E.10 Cloud Infrastructure Requirements (E.10.1 - E.10.4)*
        - *E.11 Flow Configuration (E.11.1)*
    - Rehenuma Tasnim Rodoshi:
        - *E.7 Push Button Requirements (E.7.1)*
        - *E.10 Cloud Infrastructure Requirements (E.10.1 - E.10.4)*
    - Rezoan Ahmed Nazib:
        - *E.5 Light Sensor Requirements (E.5.1)*

        - *E.10 Cloud Infrastructure Requirements (E.10.1 - E.10.4)*
    - Samuel Murray:
        - *E.1 Generic End-Device Requirements (E.1.1 - E.1.8)*
        - *E.3 Generic Trigger Device Requirements (E.3.1 - E.3.3)*
        - *E.4 Generic Activator Device Requirements (E.4.1 - E.4.2)*
        - *E.6 Switched Outlet Requirements (E.6.1)*
        - *E.8 Step Device Requirements (E.8.1)*
    - Sepehr Tabrizchi:
        - *E.2 Gateway Requirements (E.2.1 - E.2.4)*
        - *E.8 LED/Candle Requirements (E.8.1 - E.8.2)*

8. Architecture and System Design

The system will be composed of many trigger devices and many activation devices all contained on end-devices that communicate with a LoRaWAN gateway to connect to a cloud infrastructure. In this section, we put the pieces together by providing high-level architecture diagrams and describing how the system is connected with its message dictionary and system design.

8.1 Architecture Diagrams

The four main architectural components of the system are:

- **LoRaWAN End-Devices** contain trigger and activation devices connecting to the cloud through a gateway on the LoRaWAN network.
- A **LoRaWAN Gateway** forwards all LoRaWAN messages to the cloud by sending them serially through USB to a computer (Raspberry Pi or Laptop) that is cloud-connected.
    - Note that the Gateway was intended to be WiFi connected, but we were unable to receive cloud-to-device commands due to limitations with the libraries. In an effort to implement a complete system, we adjusted our architecture.
- A **GUI Application** serves as a trigger and activation device in the system, but also connects the cloud and LoRaWAN devices in our system.
- The **Cloud** controls all interactions with devices. For every trigger, the cloud determines which activations receive commands.

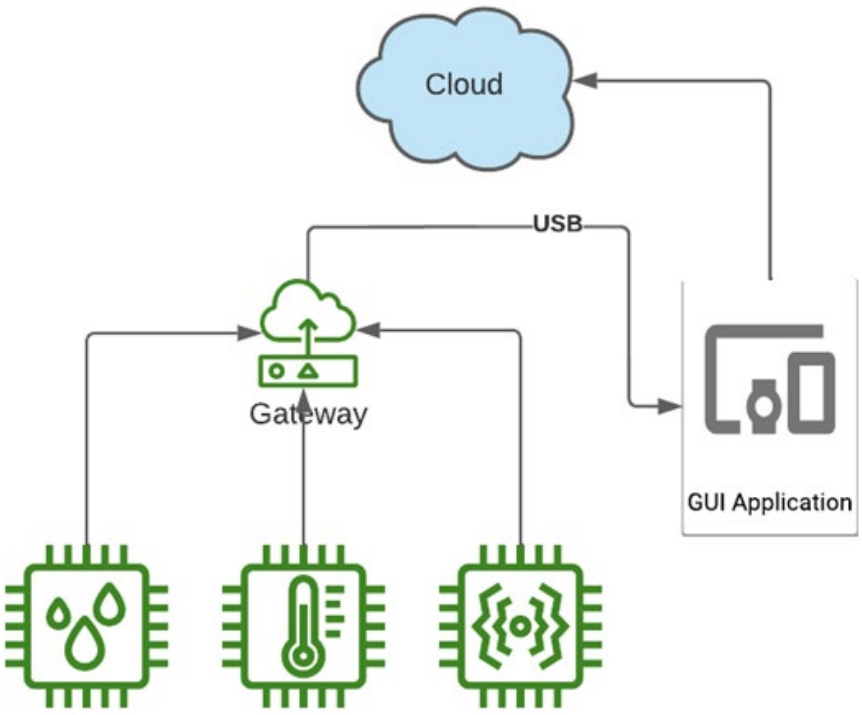

Figure 8.1.1: The system architecture implemented is shown above.

Other System Architectures

In the initial project planning, we designed an architecture with a WiFi-connected Gateway that directly forwarded LoRaWAN messages to the cloud. While attempting to implement this architecture we ran into issues. The libraries for the LoRaWAN network are not compatible with the libraries for connecting the Gateway to the cloud. Specifically, we were unable to receive cloud commands using the libraries suggested by the TA. Thus, we switched to our implemented architecture shown above because it was the easiest solution with the given time constraints.

The implemented architecture is not ideal. One disadvantage is that the GUI Application is now required when not every escape room will want it. In future iterations of the system, we will push WiFi end-devices instead of LoRaWAN. The ideal board is the Raspberry Pi Pico W due to its WiFi and Micropython capabilities.

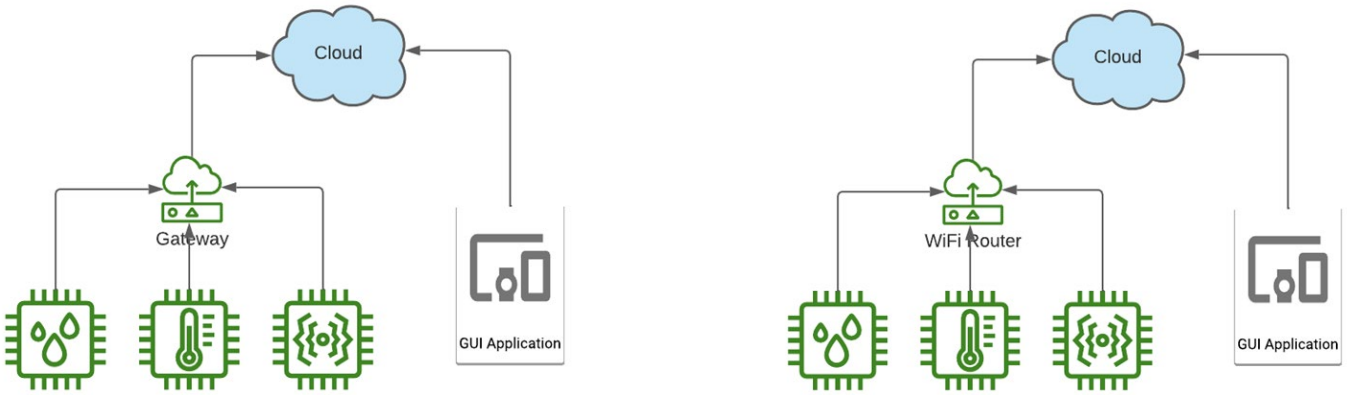

Figure 8.1.2: The originally planned system architecture (left) and future system architecture (right) are shown above.

8.2 Architecture – Message Dictionary

All messages in this project will use a JSON string format. Messages are sent either from the server to an end node, or from an end node to the server. The following table shows the complete list of JSON dictionary keys at the top level of the dictionary hierarchy, along with the value types and formats:

Key	Type	Required?	Description
“SNID”	int	always	The source node’s unique ID. The server’s ID is always 1.
“TNID”	int	always	The target node’s unique ID. The server’s ID is always 1.
“PID”	int	always	The packet ID. Each end node keeps its own tally of packets transmitted between it and the server. Every TX or RX packet increments the packet ID by +1. Packets with a packet ID less than or equal to the packet ID tally are ignored as repeats.

			Packets with a packet ID greater than the tally generate an error/warning that is sent to the server for logging.
“UT”	int	always	The uptime of the source node in milliseconds. This helps with synchronization and reset detection.
“CMD”	string	Optional, server to end node only	A custom command sent from the server to the end node. Commands include “START”, indicating that the escape room game has begun, and “STOP”, ending the escape room game.
“AL”	list	optional	The activation device list. The server can control any activation device on a particular end node by including it in this list as another JSON dictionary within the list. Likewise, the end node can report the state of an activation device to the server by including it in this list. Not all available activation devices need to appear in the list: only those that require a state change or are requested for update. The format of the activation device JSON dictionary is listed in a table below.
“TL”	list	optional	The trigger device list. The server can request to receive the state of any trigger device on a particular end node by including it in this list as another JSON dictionary within the list. Likewise, the end node can report the state of a trigger device to the server by including it in this list. Not all available trigger devices need to appear in the list: only those that are requested for an update or that have a new state to report. The format of the trigger device JSON dictionary is listed in a table below.
“ER”	string	Optional, end node to server only	The end nodes can report general errors with this optional string, such as watchdog timer faults or packet ID faults. If this key is not present in the JSON dictionary, it is assumed that no error has occurred since the last transmission. If the end device has an error, it is required to report it immediately to the server.

The top level of the JSON dictionary can include the activation device list (“AL”) key if desired, either from server to end node or end node to server. Each item in the activation device list corresponds to exactly one activation device on the end node. The end node can include an activation device in its transmission to the server to report the current state of the activation device. The server can include an activation device in its transmission to an end device to either: set the state of the activation device, set up the activation device configuration, or request an update of the activation device state. The following table shows the complete list of JSON dictionary keys within each activation list item:

Key	**Type**	**Required?**	**Description**
“ID”	int	always	The activation device ID. Starts at 1 and increments by 1 for every activation device on the end node.
“T”	string	always	The type for the activation device. Every type of activation device, such as an LED or switched outlet, has a unique type string. The type string allows the server to know how to control the activation device

“C”	string	Optional, server to end device only	The configuration string. The server may set an optional configuration for the activation device that is dependent on the type of activation device. Not all activation devices require a configuration.
“S”	string	optional	The state of the activation device. The server can control the state of the activation device by including this key in its transmission. Or, it can request an update of the state of the activation device by including that activation device in the list without including this key. On the other hand, the end node can report the current state of the activation device by including this key in its transmission.
“ER”	string	Optional, end node to server only	The activation device can report errors to the server with this string, such as if it failed to make a state change or if a state change string was not parsed correctly. If the activation device has an error, it is required to report it immediately to the server.

The top level of the JSON dictionary can include the trigger device list (“TL”) key if desired, either from server to end node or end node to server. Each item in the trigger device list corresponds to exactly one trigger device on the end node. The end node can include a trigger device in its transmission to the server to report the current state of the trigger device. The server can include a trigger device in its transmission to an end device to either set up the trigger device configuration or request an update of the trigger device state. The following table shows the complete list of JSON dictionary keys within each trigger list item:

Key	**Type**	**Required?**	**Description**
“ID”	int	always	The trigger device ID. Starts at 1 and increments by 1 for every trigger device on the end node.
“T”	string	always	The type for the trigger device. Every type of trigger device, such as a button or light sensor, has a unique type string. The type string allows the server to know how to configure and process the information from the trigger device.
“C”	string	Optional, server to end device only	The configuration string. The server may set an optional configuration for the trigger device that is dependent on the type of trigger device. Not all trigger devices require a configuration.
“S”	string	Optional, end device to server only	The state of the trigger device. The end node can report the current state of the trigger device by including this key in its transmission. The server can request an update of the trigger device state by sending its ID in the trigger device list, but leaving out the state key.
“ER”	string	Optional, end node to server only	The trigger device can report errors to the server with this string, such as if it failed to read its state. If the trigger device has an error, it is required to report it immediately to the server.

The following example depicts a JSON string from the server to end node #2 that sets the state of an activation device #2 and requests an update of the state of trigger device #4:
{“SNID”: 1, “TNID”: 2, “PID”: 10, “UT”: 23756, “AL”: [{“ID”: 2, “T”: “LED”, “S”: “1”}], “TL”: [{“ID”: 4, “T”: “BUT”}]}

The following example shows a JSON string from the end node #2 to the server that reports the state of trigger device #4 and reports a packet error:
{“SNID”: 2, “TNID”: 1, “PID”: 11, “UT”: 24620, “TL”: [{“ID”: 4, “T”: “BUT”, “S”: “0”}], “ER”: “Missed PIDs 7 to 9”}

8.3 System Design

The system components can be categorized into one of the following: Generic Classes, Device Drivers, Cloud Functions, and Other Components.

Generic Classes

The generic classes serve to establish consistency and reduce redundancy in our codebase. Every device driver uses these generic classes to handle communication, error logging, and more.

Device Drivers

Device drivers handle the specific details of activation and trigger devices. A device driver is directly connected to one or more sensors or peripherals that serve as Enchanted Objects.

Cloud Functions

We use the Function as a Service (FaaS) model for Azure Functions. Each device in our system is connected to the cloud and works with the cloud by calling Azure Functions or receiving commands.

Other Components

The Gateway and Raspberry Pi do not fit into the above categories, but each serve their own specific purpose in the system.

9 Development Progress Report

9.1 Software Development Plan

Our approach to developing this group project was to first build a framework for the structure of our system (the base classes built by Sam) and then delegate specific implementations of enchanted objects amongst the team. In section 7.2, we outline how we delegated specific engineering requirements. In this section will further specify the program files that are in development and who is taking the lead role on each document.

Below, we list the actual documents to be developed with team members who are taking the lead in parentheses. We also provide a brief description of the minimum requirements for each document.

Classes

- EscapeRoomEndNode.hpp/.cpp (Sam)
    - Implements E.1 Generic End-Device Requirements
- EscapeRoomTriggerDevice (Sam)
    - Implements E.3 Generic Trigger Device Requirements
- EscapeRoomActiviationDevice (Sam)
    - Implements E.4 Generic Activator Device Requirements

SAMD21 Device Drivers

- EXAMPLE_main.ino (Sam)
    - Example use of generic classes. Not part of final system.

- OUTLET100_main.ino (Sam)
    - Implements E.6 Switched Outlet
- LED200_main.ino (Sep)
    - Implements E.8 LED/Candle
- BUTTON300_main.ino (Rodoshi)
    - Implements E.7 Push Button
- LIGHT400_main.ino (Nazib)
    - Implements E.5 Light Sensor
- STEP500_main.ino (Sam)
    - Implements E.9 Step Device

Gateway

- GATEWAY_main.ino (Sep)
    - Implements E.2 Gateway
    - Did not finish
- GatewaySamMurray.ino (Sam)
    - Implements part of E.2 Gateway
    - Interfaces between the LoRa network and the USB serial interface, but does not implement the portion that interfaces between USB and the Azure cloud

Raspberry Pi

- RASP_main.py (Marcus)
    - Implements E.10 Raspberry Pi

Cloud Azure Functions

- The following Azure Functions collectively implement the cloud (E.11 and E.12)
    - Setup*
    - Trigger (Marcus)
    - Heartbeat**
    - Complete*
- Note the Azure Functions send commands to devices based on the IoT Central configuration.

*These are lower priority and were not be completed during this semester's project.

**This is not yet fully implemented. Given our recent architectural changes, we partially implemented the Heartbeat functionality using IoT Central Device Telemetry.

9.2 Individual Final Progress Report with Bug List

For each team member, we provide a final progress report on their contributions to the development of the system. The engineering requirement sections delegated to the individual are listed.

Marcus Gubanyi

- *E.9 Raspberry Pi Requirements (E.9.1 - E.9.3)*

- *E.10 Cloud Infrastructure Requirements (E.10.1 - E.10.4)*
- *E.11 Flow Configuration (E.11.1)*

E.9, E.10, and E.11 are complete except that the cloud is not connected to the LoRaWAN network through the gateway.

In addition to leading the writing efforts, Marcus is working on the Raspberry Pi application, the cloud infrastructure, and the flow configuration.

Completed the Raspberry Pi GUI Application, except for the USB serial connection with the Gateway and forwarding the triggers/commands between the LoRaWAN network and the Cloud.

Tested Raspberry Pi GUI Application with Cloud.

Set up and configured an IoT Central device template and device for sending commands to the GUI application.

Implemented the Trigger Azure Function invoking device commands.

Tested the Azure Function and the Cloud commands.

Designed a flow configuration for a simple escape room using our project's components.

Deployed the Azure Function to the cloud. Can be called anywhere with an HTTP request.

Complete and fully tested the subsystem of the Raspberry Pi GUI Application and the Cloud Infrastructure.

Bugs:

1. *The Raspberry Pi GUI Application is tasked with calling Azure Functions when a trigger occurs. The same application receives the commands from the cloud. The application waits for the Azure Function to respond, and the Azure Function waits for a response from the command. This stalls the system. The bug was initially fixed by intentionally timing-out the Azure Function call because the application doesn't need an HTTP response. Then, we implemented a better fix by posting the Azure Function HTTP trigger in a separate thread.*

Rehenuma Tasnim Rodoshi

- *E.7 Push Button Requirements (E.7.1)*
- *E.10 Cloud Infrastructure Requirements (E.10.1 - E.10.4)*

Designed prototype of a bookshelf with a hidden push button. The button is connected to the SAMD21 board (using ground pin and pin D3) with wires. It can be pressed by keeping book on the shelf and can be un-pressed as well by lifting the book.

The push button is tested to check whether it gives the correct result to the serial monitor for pressing/un-pressing.

The push button is integrated with the gateway code to send signal when the books are lift from the shelf to trigger the next step. Also, the button sends signal whenever a step change occurs.

Bugs:

1. *The push button was giving garbage value initially when not pressing due to not using resistor. However, SAMD21 pro rf board has an internal pull-up resistor that was used instead of the INPUT button, then the issue was resolved.*

Rezoan Ahmed Nazib

- *E.5 Light Sensor Requirements (E.5.1)*
- *E.10 Cloud Infrastructure Requirements (E.10.1 - E.10.4)*

Designed an embedded photo frame that will carry the light sensor device inside it, so that when light is flashed into the photo frame, the device will be triggered. The Adafruit BH1750 light sensor is used in this regard.

The light sensor is connected to the SAMD21 pro rf board using QWIIC port. Then the sensor is attached to the inside of the photo frame.

The Adafruit BH1750 light sensor uses an I2C serial communication protocol, and two pins are necessary for that communication which as SDA and SLC. However, the layout of the pins on the SAMD21 pro rf board is different and they exploit Sparkfun's proprietary QWIIC interfacing port. So, QWIIC cables were used to connect the sensor with the board.

The light sensor device was tested both in a dark room and an illuminated room to check whether it gives accurate intensity value. Also, the embedded photo frame was checked by directing a flashlight towards it.

The final light sensor device inside the photo frame is integrated with the gateway code and it triggers the next step when a flashlight is shined into the photo frame.

According to the design, we planned to implement a two-way communication with the cloud. However, I was only able to send data from the gateway/device to the cloud and could not figure out how to send back data from the cloud to the device.

Bugs: None to report.

Samuel Murray

- *E.1 Generic End-Device Requirements (E.1.1 - E.1.8)*
- *E.3 Generic Trigger Device Requirements (E.3.1 - E.3.3)*
- *E.4 Generic Activator Device Requirements (E.4.1 - E.4.2)*
- *E.6 Switched Outlet Requirements (E.6.1)*
- *E.8 Step Device Requirements (E.8.1)*

Designed generic end device, trigger, and activator classes that are easy to populate with simple driver functions for the desired triggers/activators on a particular end node.

Created the switched outlet and step trigger main code files, complete with the generic classes, device drivers, and LMIC library working together to form a working system. These provided examples for the others to easily get their triggers and activators streamed into the whole network.

Designed 3D model of a plastic chassis to hold the switched outlet relay PCB and the Sparkfun SAMD21 board. The model is designed to hold the relay board safely and prevent electric shocks to people or objects. The model has been printed using a 3D printer.

Assembled the switched outlet module.

Performed safety testing on the switched outlet to ensure correct wiring and inspect for exposed mains wire.

Tested switched outlet's switching capability using basic test software running on SAMD21 board.

Wrote the C++ code for the switched outlet. This is a merger of the LMIC code from lab 4 and the generic classes I developed for end nodes. It also includes the driver functions to operate the relay board and a heartbeat mechanism.

Tested the switched outlet's message receiving and processing capability using another SAMD21 board emulating the cloud's transmissions.

Wrote a USB-LoRa gateway bridge to interface between the LoRa network and a USB serial port attached to a computer or Raspberry Pi.

Wrote a local Python script to interface with the USB-LoRa gateway and perform the flow control for the escape room. This is the controller used in the final demo video we showed in our presentation.

Bugs: WDT causes millis() to run about 10 times slower, for some reason. I tried using different generic clock MUXes to use for the WDT, but they all resulted in the same problem. I checked the register configs over many times, and at this point I suspect I may have a faulty SAMD21 chip...

Sepehr Tabrizchi

- *E.2 Gateway Requirements (E.2.1 - E.2.4)*
- *E.8 LED/Candle Requirements (E.8.1 - E.8.2)*

Completed LED/Candle.

Working on completing the gateway with the new architecture: Serial Message to LoRaWAN and LoRaWAN to Serial Message.

Bugs:

1. *ESP-1ch-Gateway-v5.0 library has a problem to making connections from azure to LoRa. Therefore we try the Azure SDK for C Arduino library. This library normally is not compatible with our board therefore we have to change the board name to ESP32 Wrover Module. In this case, the connection is established but the LMIC library does not work with it. so at the end, we consider the rasberypi board as the gateway*