Team Components

Requirements

Requirement Shells (a.k.a. Snow Cards)

Functional Requirements

F1: The LED strip shall turn red (as a default) when the audio corresponds to anger.

- Description: When mood lighting is enabled for the first time by a user we should provide an initial configuration of the feature so that users can get an idea of how the feature works.
- Rationale:Red is a color that people commonly correlate to anger so red correlating to anger will make a good default to the Mood Lighting feature.
- Originator: Andrew Wojciechowski
- Fit Criterion: When mood lighting is enabled for the first time via the Sisyphus mobile app one quadrant of the grid shall show as red with the emotion anger.
- Customer Satisfaction: 7
- Customer Dissatisfaction: 5
- Priority: Medium
- Dependencies: None
- History: Last modified 2/8/2021

F2: The LED strip shall turn blue (as a default) when the audio corresponds to sadness.

- Description: When mood lighting is enabled for the first time by a user we should provide an initial configuration of the feature so that users can get an idea of how the feature works.
- Rationale:Blue is a color that people commonly correlate to sadness so blue correlating to sadness will make a good default to the Mood Lighting feature.
- Originator: Andrew Wojciechowski
- Fit Criterion: When mood lighting is enabled for the first time via the Sisyphus mobile app one quadrant of the grid shall show as blue with the emotion sadness.
- Customer Satisfaction: 7
- Customer Dissatisfaction: 5
- Priority: Medium
- Dependencies: None
- History: Last modified 2/8/2021

F3: The LED strip shall turn green (as a default) when the audio corresponds to calmness.

- Description: When mood lighting is enabled for the first time by a user we should provide an initial configuration of the feature so that users can get an idea of how the feature works.
- Rationale:Green is a color that people commonly correlate to sadness so green correlating to sadness will make a good default to the Mood Lighting feature.
- Originator: Andrew Wojciechowski
- Fit Criterion: When mood lighting is enabled for the first time via the Sisyphus mobile app one quadrant of the grid shall show as green with the emotion calmness.
- Customer Satisfaction: 7
- Customer Dissatisfaction: 5
- Priority: Medium
- Dependencies: None
- History: Last modified 2/8/2021

F4: The LED strip shall turn yellow (as a default) when the audio corresponds to happiness.

- Description: When mood lighting is enabled for the first time by a user we should provide an initial configuration of the feature so that users can get an idea of how the feature works.
- Rationale:Yellow is a color that people commonly correlate to happiness so yellow correlating to happiness will make a good default to the Mood Lighting feature.
- Originator: Andrew Wojciechowski
- Fit Criterion: When mood lighting is enabled for the first time via the Sisyphus mobile app one quadrant of the grid shall show as yellow with the emotion happiness.
- Customer Satisfaction: 7
- Customer Dissatisfaction: 5
- Priority: Medium
- Dependencies: None
- History: Last modified 2/8/2021

F5: The mood lighting mode shall be enabled via the Sisyphus mobile app.

- Description: An option needs to be available for the user to enable the mood lighting feature.
- Rationale:All options for the Sisyphus table are controlled via the Sisyphus mobile app so we should put the option to enable mood lighting where all of the other options for the table are. Also, since there are other light patterns users shall be able to enable/disable the feature on demand in order to select other light patterns.
- Originator: Andrew Wojciechowski
- Fit Criterion: When a user navigates to the settings page in the Sisyphus mobile app an option to enable mood lighting shall be available to the user as a checkbox.
- Customer Satisfaction: 8
- Customer Dissatisfaction: 3
- Priority: High
- Dependencies: None

- History: Last modified 2/8/2021

F6: The table shall collect audio samples when the mood lighting mode is enabled.

- Description: When mood lighting is enabled samples shall be collected in order to determine what color the LEDs on the table should be turned to.
- Rationale: The only option that requires audio input is mood lighting so audio samples should only be collected when mood lighting is enabled.
- Originator: Andrew Wojciechowski
- Fit Criterion: When mood lighting is enabled the table shall collect 5 second samples to use for the mood lighting feature.
- Customer Satisfaction: 8
- Customer Dissatisfaction: 10
- Priority: High
- Dependencies: None
- History: Last modified 2/8/2021

F7: The color shall be changed through already existing externally facing APIs

- Description: The table has APIs for setting the colors already and the mood lighting feature should take advantage of those existing APIs.
- Rationale:It will be easier to use the existing APIs to change the color instead of writing our own APIs.
- Originator: Andrew Wojciechowski
- Fit Criterion: When the mood lighting algorithm has calculated a color to set on the table. The feature shall use the existing light APIs on the table to set the primary color of the lights on the table.
- Customer Satisfaction: 3
- Customer Dissatisfaction: 3
- Priority: Low
- Dependencies: None
- History: Last modified 2/8/2021

F8: The user shall be able to select custom colors that map to different moods

- Description: Customization shall be available in order for users to change which colors coordinate to which moods.
- Rationale: People may correlate moods to color differently so a feature shall be provided in order to select custom color to correlate to different moods.
- Originator: Andrew Wojciechowski
- Fit Criterion: When mood lighting is enabled in the Sisyphus mobile app the color axises shall be displayed and the colors of each of the quadrants shall be customizable in the Sisyphus mobile app.
- Customer Satisfaction: 8
- Customer Dissatisfaction: 5

- Priority: High
- Dependencies: None
- History: Last modified 2/8/2021

F8: The user shall be able to enable and observe beat detection

- Description: The user can enable beat detection via the app and the lights will respond accordingly
- Rationale: People may want the table to respond to ambient audio in different ways according to more concrete metrics (beats per minute vs. mood) and this mode does not affect color but rather brightness
- Originator: R. George Burkhardt
- Fit Criterion: When beat detection is enabled in the Sisyphus mobile app the lights shall brighten and dim at the same rate as the beats per minute (BPM) of the audio being played.
- Customer Satisfaction: 8
- Customer Dissatisfaction: 2
- Priority: Low
- Dependencies: None
- History: Last modified 5/13/2021
- **NOT MET**

Nonfunctional Requirements

NF1: The audio system shall be installable within 20 minutes, by a user who has minimal experience with technology and should require similar technical skills to that necessary for installing the Sisyphus LED Kit.

- Description: The modification necessary to add audio recording and processing to the table should be minimal so that Sisyphus industries can market this addition as an upgrade kit to their users
- Rationale: Keeping install time below 20 minutes means that the user can use the new functionality quickly and does not require more of their time then necessary (since we assume most Sisyphus table owners have minimal technical experience)
- Originator: R. George Burkhardt
- Fit Criterion: The installation of the audio module (i.e. microphone, Pi processor) should take no more than 20 minutes. The upgrade kit shall be of a similar technical level to the LED install kit.
- Customer Satisfaction: 8
- Customer Dissatisfaction: 10
- Priority: High
- Dependencies: N/A
- History: Last Modified 2/9/2021

NF2: The system shall alert the user that any and all audio recordings are not saved.

- Description: A disclaimer should be available to the user saying than no audio will be recorded and stored on external servers if any web services are used
- Rationale: In keeping with ethical software practice, the system should ensure the user is aware of how the audio is used.
- Originator: R. George Burkhardt
- Fit Criterion: A disclaimer is immediately available to the user upon installing the audio upgrade kit
- Customer Satisfaction: 7
- Customer Dissatisfaction: 3
- Priority: High
- Dependencies: F5
- History: Last Modified 2/9/2021
- **NOT MET**

NF3: Any new additions to the UI will abide by the current UX design patterns and standards
- Description: Any new UI components (e.g. dropdowns, icons, etc.) should be styled in similar ways and any interactions should match how the app currently operates (e.g. checkboxes for toggling options)
- Rationale: The app should provide a cohesive user experience between old features and new features. Changes in styles and patterns can be jarring to users, thus it should be avoided or kept to a minimum
- Originator: R. George Burkhardt
- Fit Criterion: New UI elements will be styled with the same colors, fonts, and icon packs that are currently in the app. New UI elements will be kept to a minimum.
- Customer Satisfaction: 5
- Customer Dissatisfaction: 5
- Priority: Medium
- Dependencies: F6, F8
- History: Last Modified 2/9/2021

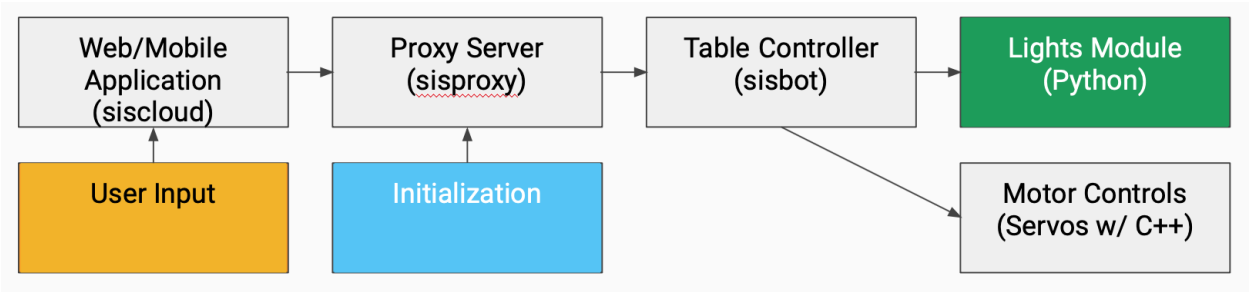
NF4: The audio feature shall not negatively impact any existing functionality
- Description: Any changes made should be done in such a way that they do not regress the system
- Rationale: Any changes will need to exist side by side tables that do not have audio/lighting functionality thus any changes must be atomic as to not ruin the product experience for other users
- Originator: R. George Burkhardt
- Fit Criterion: The table will be able to function normally with or without audio input
- Customer Satisfaction: 8
- Customer Dissatisfaction: 10
- Priority: High
- Dependencies: N/A
- History: Last Modified 2/9/2021

The project did not succeed in meeting all requirements, however, a large headway was made in attempting to complete those requirements. If this team (or any other) were to continue the project - those requirements would be top priority and the wealth of information and experience generated by this team in their attempt would serve well as a project basis.

Software Architecture and Modeling

Original Sisyphus Table Architecture

The Sisyphus table consisted originally of 3 distinct parts - the motor control, the lights control, and the server that orchestrates all operations and communicates with the client app which provides user controls. These parts map to real world elements of the table and serve as a consistent separation of concerns. For the beginning of our project the team was tasked with exploring the lighting system and generating new and interesting interactions. This required knowledge of how the system and it's APIs work but also required understanding of how commands are sent from the server and processed by the lighting module. For example, to control the lights a command is sent from the client app to the table's server which will then gather data from the appropriate locations (data stores for color preference, current ball position queried from motor control), package it, and send it over a local file server connection to the lighting module. Planned work for incorporating APIs to determine lighting behavior must understand, and extend this client-server architecture in order to produce new functionality.



Implemented Architecture: High Level

The team has decided that the project goal is to integrate a new feature into the table whereby the table will take an audio stream and use machine learning (ML) libraries and a trained model to map the songs mood and energy into a color which will be displayed via the table's LED light strip.

Because the table is powered by a Raspberry Pi and Arduino duo and through a series of tests, the team has determined that it is not feasible to run both the audio stream and ML model on the table while running the already existing siscloud, sisproxy, and sisbot servers. It is a strict requirement that the table does not functionally regress with the addition of this "listening" feature. Therefore, the team has investigated and is currently settled on using a server based

architecture to both accept, stream, process, and apply the audio stream and its corresponding RGB output.

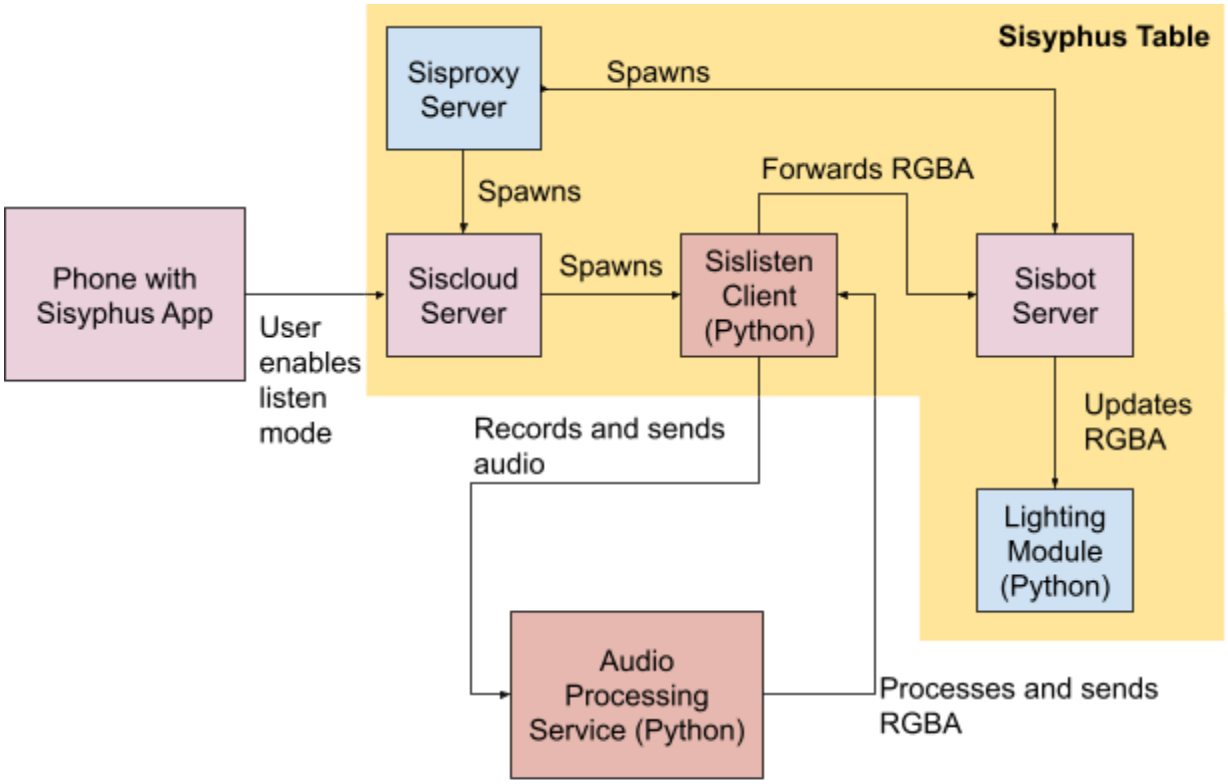
The team chose a server architecture primarily for 3 reasons. First, the table is already set up with a server-based architecture with different application domains passing messages and commands via HTTP and WebSocket calls. Second, running a ML model and processing audio requires fast and reliable hardware. The low RAM amount and slow processor of the Raspberry Pi makes it less suited to this application. Servers, either local or in the cloud, are able to leverage large computing power at a lower cost. Thus, keeping different activities (i.e. recording and processing) abstracted from each other allows the team to tailor hardware to the specific activity's requirements. Third and last, a server architecture allows for a great deal of abstraction and extension in both the interface of accepting and processing audio as well as the location of the server's main processing unit. A server may live in the cloud or on a local network (assuming a user has a powerful enough device). A server can service many customers at once and allow for incremental upgrades (changes to the API) without requiring massive deprecation or massive upgrade cycles.

With these considerations in mind, the team has set up a solution that allows for a client script and server script (currently running in Google cloud) to record, stream, and process audio on the Pi. The mapping of audio data to a mood coordinate pair (representing valence and energy for a given piece of audio) is then sent as a reply to the original caller's Sisyphus table, which in turn matches colors to this mood and manipulates the lights to reflect this change via existing endpoints in sisbot.

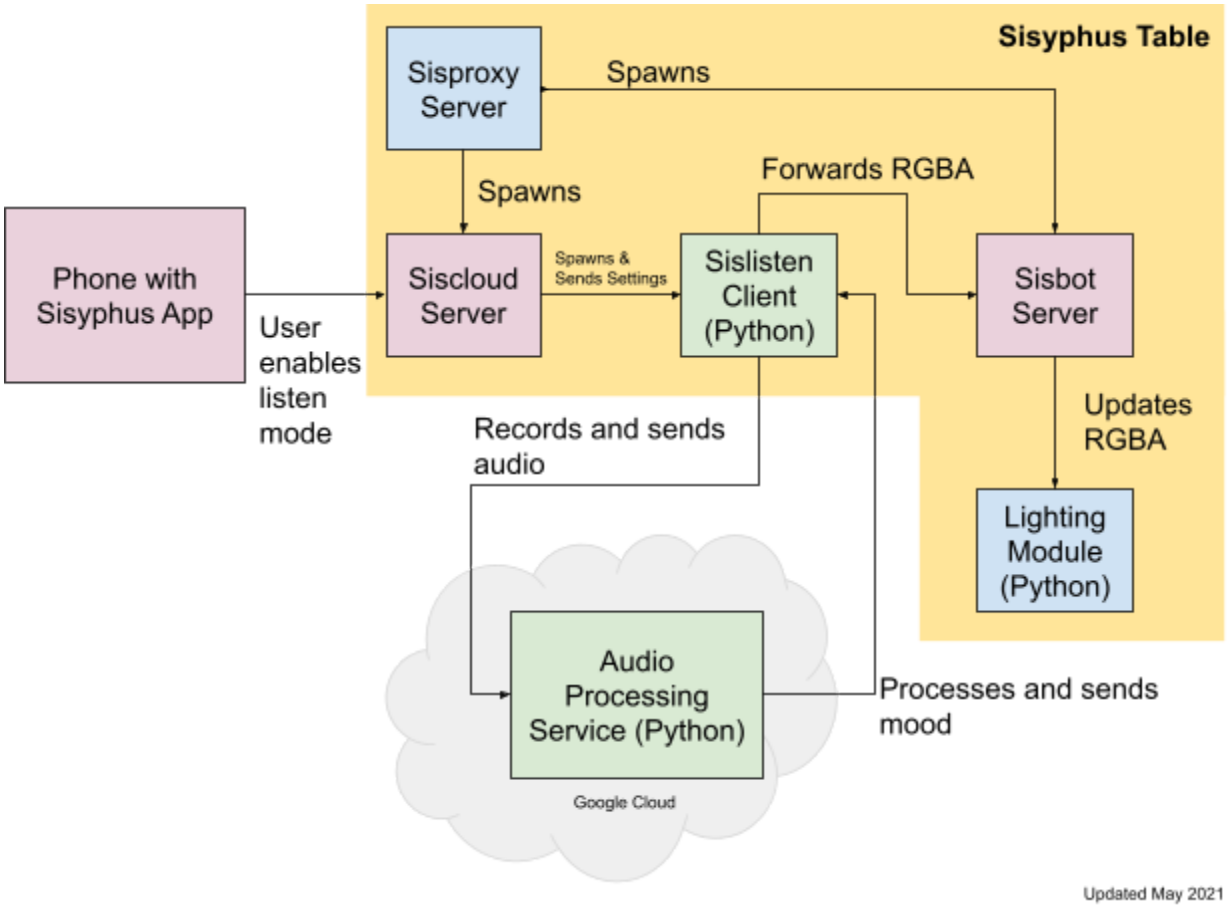
However, there are some drawbacks to this server architecture model that the team is aware of. First is the issue of latency. Streaming large amounts of data (at a max 80kb for 1 second clips) is not a cheap procedure and typically yields a latency of just over 1 second which is an important issue in terms of real-time audio processing. Were the team to continue with this project, different protocols that allow for more streamlined audio (e.g. WebRTC) rather than sending everything in one large HTTP request would be explored.

Ultimately, the team is confident in the constructed architecture in terms of quality attributes, and believes the current implementation, if security measures related to audio-recording controls placed on-platform are added prior to deployment, will safely function without problem in the production environment.

Below is the original systems diagram showing the interactions between existing modules (from the diagram above) and new modules, marked in dark red.



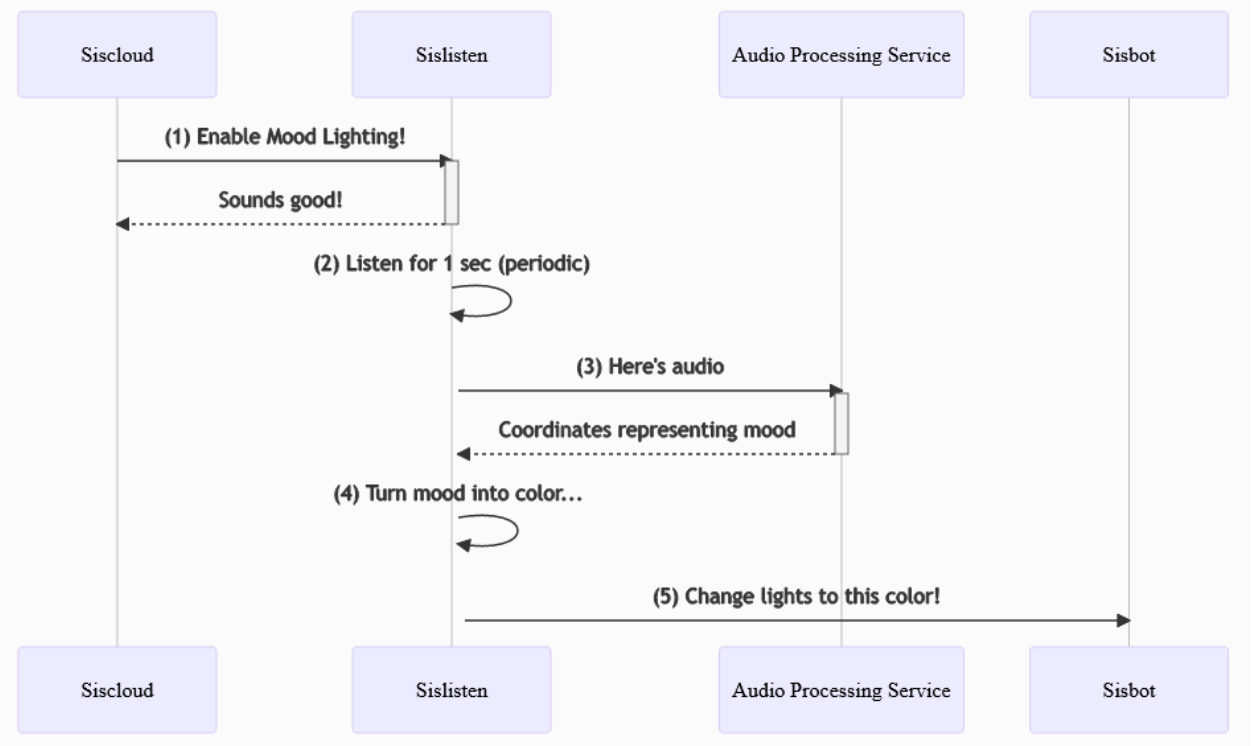
The architecture shown diagram was changed slightly in implementation, and an updated version is shown below:



This updated version incorporates two key changes : first, the siscloud server is responsible for not just enabling/disabling mood lighting, but also communicates user settings to the new sislisten application. “User Settings” are settings that refer to a user’s preferred color for each of seven basic moods--disgust, anger, alertness, happiness, calmness, relaxation, and neutrality. This new module is discussed in further depth below. The second key change is that the Audio Processing Service, whose final deployment location was unknown at the time of diagramming previously, has been updated to reflect its current home, google cloud.

Implemented Architecture: Sislisten

The largest addition to the existing codebase by the team was undoubtedly sislisten. Sislisten is pictured in the upper right middle of the diagram directly above, and serves as an audio recorder, a mood-to-color translator, and a color-to-lights communicator. To facilitate these jobs in near-real-time, the team made use of a periodic task to record audio, and the thread pool pattern to spawn off tasks asynchronously for processing responses from the APS (Audio Processing Service, also called the “ML” service since the Machine Learning model lives on this server). In practice, this looks something like:



Step 1 is handled by a web server built with Flask. Step (2) is handled by a periodic task, and steps 3-5 are handled by a large thread pool, ensuring that audio responses from the Audio Processing Service are serviced outside of the main thread responsible for listening to audio.

Design

The system currently does not use any patterns and this has resulted in monolithic files and copy-pasted files with duplicated behavior and code. If given enough time, it would be helpful to refactor this to be more modular. However, that is unlikely to be achievable in the time given.

One such example is the lighting module. Each lighting pattern has its own file with two functions (an init and update). The init behavior is nearly identical across files while the update contains the business logic for executing the different lighting patterns. This module can be refactored to use a strategy pattern (context uses common interface, multiple implementations).

In keeping with agile principles, the team does not plan on working on refactors outright but will rather incorporate re-writes and patterning as code is modified with other work. With new functionality, patterns and other design strategies will be considered in how to best implement the work.

With the additions of the sislisten module and mood lighting A.I. service the team introduced a thread pool and client-server design pattern respectively. The sislisten module is composed of a recorder, communicator, and translator which all run asynchronously from each other by passing and communicating audio data and coordinate mappings via thread-locked queues. The goal of using this pattern is to to decouple different operations from each other and to enable

coordinated asynchronous execution of those operations due to reliance on an external web service. The A.I. service uses a client-server pattern since processing audio data on the platform (Raspberry Pi) is not feasible as to not degrade performance of other functions (as noted in NF4). With two platforms now in use, a client-server pattern is perfect for sending data, guaranteeing interface contracts, extensibility (for adding new services in the future), and decoupling the recording functionality from the transformation functionality. Finally, a client-server architecture allows for variability in location and availability requirements that may arise in the future.

Testing

For testing the mood lighting server, if there are changes to the machine learning algorithm itself we went through and manually tested the changes to that algorithm to make sure that the algorithm is doing what we expect it to. Also, for the new sislisten server we created a suite of integration tests for that server. Those included tests to make sure that the server can receive a valence/energy coordinate properly from the AI server, tests to make sure it can translate those coordinates properly to a color, and we had tests which verified that we can set a user's mood color settings properly on the server. We used the Python library PyTest to write those tests and we used the Python library assertpy for making assertions in those tests. The sislisten automated tests were run in a GitLab CI pipeline which ran on every push to the sislisten repository. Also, for any change we made to our new modules we made sure to manually test our changes. For any change to sislisten we made sure to manually test our changes on the target table platform to make sure there were not any platform specific issues with sislisten.

Security and Privacy

Security and Privacy of Existing Code

The security and privacy of existing code is an area of concern to the team. Production devices are sent into production with port 22 (SSH) enabled, and the same password, which is brute-forcible in about 1 year according to empirical estimates. The system does use TLS to communicate with the Sisyphus web server, however, so the system is not entirely insecure. Given more time, the team would have liked to make an audit of the existing security of the system.

However, given that the focus of this project is to provide additional functionality in the area of mood lighting, the remainder of this section will concern itself only with the security of added functionality.

Security and Privacy of New Functionality

These new areas of implementation consist primarily of:

1. Adding a new web API that will listen to requests from a client and respond with a color relating to the audio data given.
2. Creating (a) client (s) able to capture audio and contact the API, then take the API's response and give the response (presumably a color) to the table's *sisbot* API in order to change the color of the lights.

Threat Model

Spoofing

Incident Example:
In this case, an unauthorized client contacting the web API would be the worst-case scenario.

Mitigation:
The client and server interactions will utilize API keys to ensure that each client in contact with the server is authenticated. There is a risk that an illegitimate source steals an API from a client. Thus, if we are to ensure that API keys are secure, our clients must store them securely. This can be accomplished by ensuring that API keys are only accessible to root users and are encrypted on-device; furthermore, API keys (encrypted or unencrypted) must not be used in code that undergoes compilation; raw inspection of an executable may reveal this key.

Tampering

Incident Example:
A bad actor could gain access to the device hosting our server and corrupt the model onboard the server, which would degrade the quality of our service.

Mitigation:
Ensure the hardware hosting the API has proper access control, and is physically in a secure location. The advantage of our system in event of a tampering attack is that the API will have no persistent storage attached to it, and so there is little information to be corrupted in the first place from a tampering attack on the server side.

Repudiation

Incident:
Repudiation deals with the ability of a user to make an illegitimate claim that cannot be disproved--for instance, claiming that a transaction was never completed at a store. Without a receipt or record of sale, the business at which the transaction was completed cannot disprove the claim.

In the case of our system, repudiation could look like a customer complaining that the API responsible for handing mood lighting is down, because their table's lights aren't changing when they have their microphone. In reality the issue might span several causes: a user's microphone could be broken, the client could be failing to collect valid audio, the client could be failing to

stream that valid audio, the *sislisten* API could be actually down, the client could be receiving a valid response from the *sislisten* API but failing to send the color to the *sisbot* server running on the table, or the request could be sent but the *sisbot* server could be down.

Mitigation:
Keep clear logging in all aspects of our pipeline. Moreover, ensure logs are protected by either sudo access only, or secured behind server accounts so that in the event of a system incident or an intrusion, log files are not able to easily be removed from the system.

Information Disclosure

Incident:
A user has mood-lighting enabled on their sisyphus table. A client is collecting audio and sending it to the server. An eavesdropping breach occurs where an unauthorized actor is able to capture audio data being transmitted between the client and the server. The unauthorized actor is able to listen to this data, and potentially use it in a variety of ill-fated future exploits: blackmail, posting it online, revealing company IP (if table is in a business or home of an employee for a business), social engineering (learning names and inside conversation of a household, etc.).

Mitigation:
Repel eavesdropping attacks by using public-private key infrastructure. Ensure the private key is kept secure and is 1) password protected and 2) only visible to a sudo/admin user.

Denial of Service

Incident:
A DoS attack is aimed at our *sislisten* API.

Mitigation:
Use rate-limiting for clients connecting to the API. In the case of repeated rate-limit violations, blacklist the client and refuse connections.

Elevation of Privilege (& Privacy Concerns)

Incident:
An unauthorized actor has gained terminal access to the client and added themselves as a sudo user--perhaps they stole troubleshooting credentials from Sisyphus, or used another kind of social engineering to gain access. The end result is the same: an unauthorized user has sudo access. At this point, a user with sudo access can delete logs, potentially access the API key for contacting *sislisten*, utilize the microphone to listen to a customer, and most likely access the API for *sisbot* as well.

Mitigation:

Once an attacker has breached the client (assuming not a mobile phone) as a sudo-er (or administrator) there is little that can be done to protect the integrity of the client. Ideas to slow down the hacker include:

- Creating resource locks on the microphone to prevent the attacker from using it to listen while the mood-lighting client process is running.
    - The attacker could kill the mood-lighting server to release the lock
- Make the client-process auto-restarting such that the hacker has a hard time permanently disabling it. Have the client process lock down the resource even when not actively listening.
    - The attacker would most likely delete the service entirely and need to reboot.
- Utilize DHCP such that the hacker must repeatedly network-scan in order to retrieve the IP for continued access to the device.
    - If a hacker found the IP once, they will likely find it again. This will just make it more irritating for them to do so.
- Regularly rotate passwords such that the hacker cannot re-access the device with old credentials. Regularly remove unwanted accounts.
    - In order to get around this obstacle, a hacker would need to have a replicable way of obtaining root passwords from Sisyphus (hopefully difficult)
- Have a watchdog service to send alerts to the *sislisten* and *sisbot* API's when the client microphone is enabled and the mood-lighting feature is not enabled. Have the alerts email the Sisyphus support team.
    - The hacker could disable this, but it would likely take them time to determine that the events were being broadcast, by which time a broadcast would have been pushed to the other servers already, and the email to Sisyphus sent.
- Include logging for all access to the microphone.
    - A hacker could destroy the logs, of course. Failing to do so, however, if the hacker is in a rush or cannot find the logf iles, and would provide information to the Sisyphus team and customer about how the hacker used the microphone.
- Turn on a visible light on the client when listening to audio.
    - Depending on the driver used, a hacker could disable this also, but it might alert the user to an issue before the hacker has time to disable it.
- Create a physical switch to disable/cut power to the microphone
    - This wouldn't be required for users, but it would with absolute certainty prevent a hacker from using the microphone
    - At worst, the user could simply disconnect the microphone from the table, but this is 0/10 on usability.

Security Summary and Future Directives

For each of the security threats mentioned above, appropriate mitigation measures were recommended. In summary, these include:

- Use of TLS to encrypt audio streams transmitted between *sislisten* and the APS (Audio Processing Server, or the "AI" service).
- Use API keys to communicate between the client and *sislisten*.

- Use TLS to communicate between the client and *sislisten*, *even if over a local network.*
- Have clear logging on both the client and server.
- Construct a watchdog for the microphone to protect clients from being listened to in the event of a client breach.
- Explore use of rate-limiting on the server.

Security State at Project Conclusion

Despite the team's best efforts to enforce security in all of their activities, of the mitigations identified in the threat model, only the first has been applied as of 05/12/2021. **Implementing the rest of the recommendations is considered to be a prerequisite for deployment, and the team would urge Sisyphus industries to consider these options prior to deployment of the final system.**

Tools

Code Storage & Continuous Integration

Gitlab

- Issue Tracking - we utilized the Gitlab native issue tracker to assign tasks to specific people, track the time we were spending on each issue, and commenting on knowledge acquisitions to create a detailed log for anyone else who viewed the issue
- Milestones - as a scrum team we leveraged the Gitlab milestones to act as sprints, assigning issues to be completed, tracking our burndown, and grooming the backlog to ensure the maximum priority tasks are always getting completed
- Wiki - we utilized the builtin wiki in Gitlab to keep track of our external documentation: from ceremonies to brainstorming to meeting logs
- CI - we created a project to run our test set against all the existing projects on a push

Docker

- Used by the gitlab-runner service to provide a repeatable, lightweight environment for running tests. Our specific application incorporates the python-test image provided by a gitlab template.

Pytest

- Pytest is the official python-sanctioned framework for constructing unit tests in python. The team made extensive use of this for testing the sislisten.

Google Cloud

- Used as the deployment platform for the APS (Audio Processing Service). This was determined to be the best option after a series of cloud provider spikes evaluated for security, latency and cost.

SSH Clients

- Since our target platform was linux, on the raspberry pi, our senior design team spent considerable time using SSH clients to communicate with our target hardware. The two most popular were MobaXTerm and Putty.

Hardware

- Our hardware for this project is somewhat set except we made a slight adjustment by adding a USB hub to the table so that we could add a USB microphone to the table. However, we also had to overcome a number of problems in order to develop locally. For developing code controlling the lights, we each had to purchase the relevant hardware, and then adapt the code so it would work locally. We also needed to interface with the existing hardware, so we set up a live-stream on twitch.tv and got ssh capabilities exposed on a VPN.

Software

- Pylint & ESLint - in order to improve the quality of the code, the team introduced linters to the projects. These will hopefully ensure that the code we are adding to the project is clear, readable, and maintainable.
- pyAudioAnalysis - Python library for extracting features from audio data.
- Flask - a Python web framework which can be used for Python servers. We used this for the new sislisten module as well as for the mood lighting server.

Experimentation and Prototyping

While this project is fairly limited where new technology is concerned, we are interested in improving some out of date software packages included in the existing application. To do this, we are creating prototypes of the existing code with the newer packages to gauge the difficulty of upgrading the codebase.

Spikes

1. Choosing Audio Setup

Related Code : https://gitlab.com/msoe.edu/sdl/sd21/sisyphus/lights-audio-input

Wiki: https://gitlab.com/groups/msoe.edu/sdl/sd21/sisyphus/-/wikis/Knowledge-Acquisitions/Pis-and-Lights-and-Sound-Oh-My!

With this prototype the team wanted to test the optimal audio setup and explore different low level and high level audio interface libraries. One of the first things done in this prototype was to explore how audio may be recorded on a Raspberry Pi via the Raspbian Linux distribution. The team discovered that both OSS (deprecated) and ALSA are available on the Raspberry Pi. There are two Python libraries that wrap these interfaces, ossaudiodev (a native Python module) and pyalsa. Ultimately, it was decided that these libraries were too “low-level” and would require a lot of development time to produce useful code. Next, the team evaluated higher level C/C++ libraries like PortAudio and PulseAudio which allow for cross platform development and the use of “higher level” Python libraries like pyaudio and sounddevice. PusleAudio was found to be too complex and overkill for the requirements of the new feature but PortAudio and pyaudio were found to be appropriate in terms of API footprint and feature set (see above for additional details). Finally, the team tested different audio hardware devices - primarily USB microphones and USB sound cards with 3.5mm microphones. Via a series of tests with passing audio to the machine learning model, it was decided that USB microphones were the better choice since they yielded better audio quality on average at a cost that wasn’t too expensive (versus the wide range of audio cards and microphones that would yield poor quality audio/data on the lower end side). So, for the purposes of developing the feature, the team decided on using USB microphones with PortAudio and pyaudio to record and process audio.

2. Exploring ML on Pi

Related Code : https://gitlab.com/msoe.edu/sdl/sd21/sisyphus/sound-output-to-color-spike
Related Code : https://gitlab.com/msoe.edu/sdl/sd21/sisyphus/sisbot-lights-from-audio-input
Related Code : https://gitlab.com/msoe.edu/sdl/sd21/sisyphus/mood-lighting-performance

With this prototype the team wanted to learn if we could run the machine learning algorithm to convert audio data into a color on a Raspberry PI. One of the first things we did with this prototype was to investigate running a machine learning algorithm to convert audio data from a wav file to a color and then updating a light strip. We found that this approach was very feasible but this was causing the Raspberry PI to overheat. We then converted the prototype to get the audio data from a microphone and make a web request to the lab table which was also very feasible but it still caused the Raspberry PI to overheat. We did a CPU analysis of this algorithm on the PI and we found that it was impeding the CPU of the Raspberry PI. We then got this down to about 60% once we removed the library OpenCV from the algorithm but we wanted to get that down further. Finally, one of the last things we investigated having the PI only listen to the audio data and stream that to a Python server running elsewhere which had a very minimal impact on the Raspberry PI. So, even though running the ML on a Raspberry PI was feasible we decided that running the ML elsewhere was a better option in order to not impede the CPU on the table.

3. Exploring ML on Phone

Related Code : https://gitlab.com/msoe.edu/sdl/sd21/sisyphus/phone-ml-demo
Related Wiki:
https://gitlab.com/groups/msoe.edu/sdl/sd21/sisyphus/-/wikis/Knowledge-Acquisitions/Mobile-as-a-Artificial-Intelligence-Computation-Platform

With this prototype the team wanted to learn if we could run a machine learning algorithm to convert audio data into a color on a phone. One of the first things that we looked into if it was possible to leave the existing code in Python but this was not feasible since there are very few mobile app libraries out there that support Python and there were none that support installing pip dependencies. The next thing we tried is extracting all of the audio features on the phone and streaming them to a Python server to do the classification. This was not feasible as well since there's a lack of libraries for feature extraction in Java which was the language we used. Also, it was very difficult to get the features similar to what pyAudioAnalysis was expecting and linear algebra in Java was more difficult than Python. This technically is possible and this would have not impeded the CPU on the table's Raspberry PI but given the difficult feasibility we determined that the risk was too high for this option compared to the alternatives such as having the ML completely be on a web server.

4. Exploring ML on Server (ML as a Service)

Related Code : https://gitlab.com/msoe.edu/sdl/sd21/sisyphus/mood-lighting-server

With this prototype, the team wanted to test if the machine learning audio processing could be moved from a local environment to an external server without degrading performance. The team was able to split existing spikes into a client and server script. The server script was uploaded to Heroku while the client script remained on a microphone equipped Pi. After sorting out VPN issues (due to remote development policies in lieu of COVID-19) the team was able to take audio from a Pi, send it to the server, receive a RGBA value, and then forward that to the team's table to update it's color. The latency sat at about 2 second which isn't optimal for "real-time" audio processing but there are still many optimizations available (such as a local server rather than one in the cloud) and the goal for the spike was met, proving that a server architecture is feasible.

5. Mood Lighting & Alternative Algorithms

Related Wiki:
https://gitlab.com/groups/msoe.edu/sdl/sd21/sisyphus/-/wikis/Mood-Lighting-Feature-Extraction

- Attempted to investigate an algorithm for correlating music to color, with the goal of delivering a feasible algorithm to turn an audio into a color.
- Addressed writing a home-grown solution
- Addressed writing a solution that does not utilize any "machine learning" techniques but rather raw audio feature extraction / fourier transformation.

- Ultimately decided that the risk and cost of maintaining our own algorithm was quite high: the development team lacks skills and time to acquire sufficient skills to author an algorithm effectively, especially without a readily available dataset.
- Using a non-machine learning technique was found to be unreliable due to difficulty quantizing the relation between common audio features and a mood output.

6. Cloud Investigations

Related wiki:
https://gitlab.com/groups/msoe.edu/sdl/sd21/sisyphus/-/wikis/Architecture/Deployment

Using the results from previous investigations that determined that cloud hosting of the server would be viable, the team worked to investigate some viable services to host the server. Of the different options tested, Amazon Web Services turned out to have complex pricing, and no security unless a domain was also purchased. Azure cloud turned out to be extremely unreliable. Heroku was easy and free, but had issues with the server going to sleep and slow performance. Google Cloud ended up being the final choice because it provided reliable hosting while not costing any money, and was capable of running the server with security.

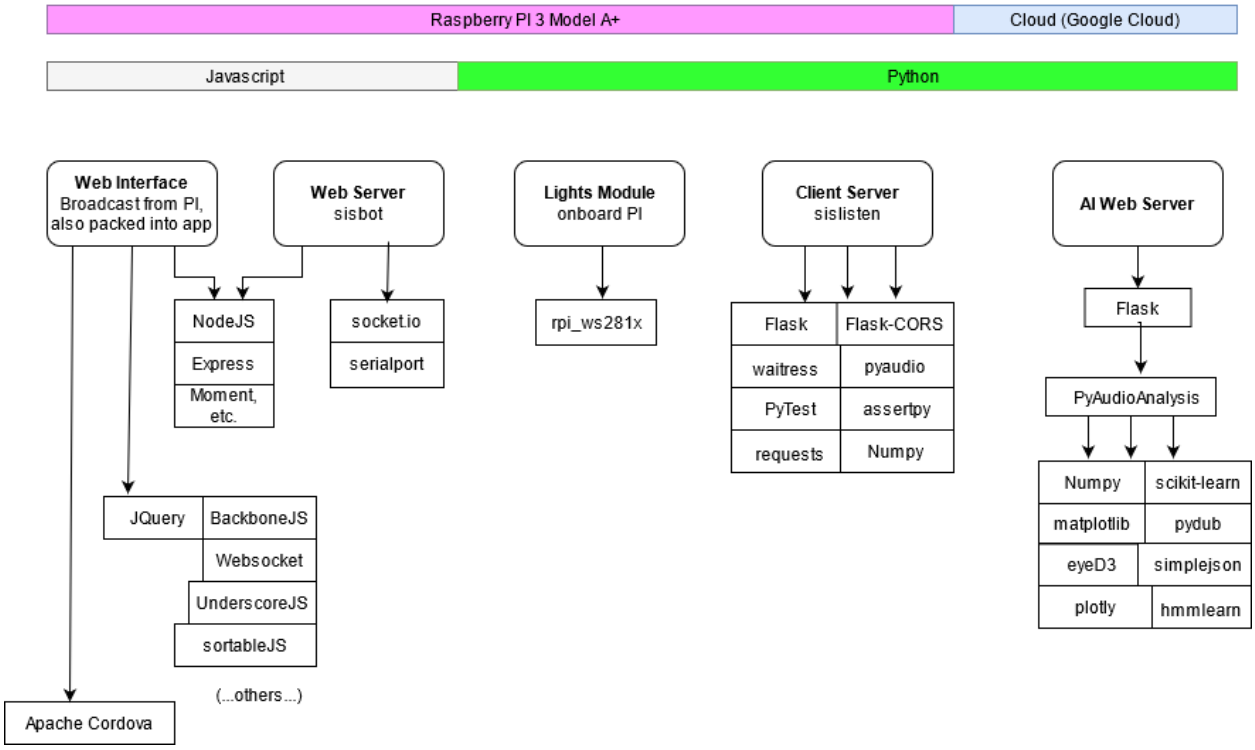
7. Upgrading Python

During the development of our machine learning algorithm, the team ran into an issue with a necessary library only supporting python3.7 while the table was running an older version of Raspian that only supported python3.4. At first, we tried simply installing the binary, but this failed due to the OS being out of date. We then tried to install Python3.7 from source, but ran into issues with it not being able to compile a C++ dependency. Manually compiling the dependency seems to work, but multiple other binaries turn out to be corrupted. Ultimately, the whole project had to be ported over to support a newer version of the Raspbian OS, and the entire table was updated.

The final image for the OS is located here:
https://drive.google.com/file/d/1VZGxCC5UIQvTIJFN0PRCLqooJrSmV0H0/view?usp=sharing

Third Party Components

The following diagram shows the primary third-party components in use in our project, both existing that those that we are adding in order to facilitate mood lighting. This diagram serves as an organizational guide for where the different libraries are used, and which are dependencies of each other. It is not comprehensive, but rather covers those most important.



Software Third-Party Components

Web Interface

This is the component of the codebase that covers handling user input from the app (when a user changes a track, toggles the lights on/off, etc.). It contains a minimal server backend to deliver the JS to the client, and then all requests from there are directed to the **Web Server** (sisbot). The web interface makes use of a number of 3rd party libraries. The most important on the frontend are as follows:

- JQuery/Underscore provide convenience functions for manipulating the DOM
- BackboneJS library serves as a framework to manage views, events, data, etc.
- SortableJS provides sorting functionality for collections by drag and drop

Additionally, there are number of JS libraries used on the backend (NodeJS server) to facilitate delivering pages and server functionality:

- Express allows running JS as a server
- Apache Cordova allows packaging the application (in its entirety) as an app for a mobile platform (both Android and iOS).
- Moment, one of a number of convenience libraries, allows for readable formatting of dates

Web Server (sisbot)

This is the component of the codebase that handles requests from the app (on a user's phone or in the web browser) and translates these requests into hardware-level changes; e.g., changing the lights, speeding up and down the ball, etc. It makes use of

- Express, to run JS as a server
- Socket.io, as a server-client real-time bidirectional and event-based communication between the client and server
- Serial-port allows for contacting the motors on the arduino running the step motors for the ball.
- Moment, one of a number of convenience libraries, allows for readable formatting of dates

Lights Module

The lights module uses Python, and specifically uses rpi_ws821x, a library written in python, go and C++ to handle interfacing over GPIO with the Adafruit Neopixels.

Audio Processing Service / AI Web Service

The new AI web server being added into our architecture is written in Python, and utilizes the following:

- Flask, a python package that allows for running Python as a web server
- PyAudioAnalysis, a library that allows for extracting audio features from raw data
- PyAudioAnalysis depends on a variety of 3rd-party libraries, primarily numpy (for linear algebra manipulation of audio data matrices) and scikit-learn (for classifying audio and applying machine learning models).

Client (sislisten)

The new client web server responsible for recording audio and communicating with the AI web server is written in Python, and utilizes the following:

- Flask, a python package that allows for running Python as a web server
- Flask-CORS: A Flask extension used for allowing Cross Origin requests to the web server
- Waitress: A Python library that can serve Flask applications as a production quality WSGI server
- PyAudio: A Python library used for recording audio from a microphone
- PyTest: A Python library used for writing automated tests for the server
- AssertPy: A Python library used for making assertions in our automated tests
- requests: A Python library used for making web requests
- numpy: A Python library used for linear algebra manipulation

Hardware Third-Party Components

Existing

- A Raspberry Pi 3 Model A+, used to run the Web Interface, *sisbot* Web Server and Lights Module.
- A RGBW LED lightstrip

Additions

- The cloud, which will in some capacity, be used to run the *mood-lighting-server* Web Server and provide audio processing for a client. This implementation used Google Cloud services.
- A USB microphone. This device was chosen due to its availability, cost, and quality as needed by the team for remote development and for the A.I. service.
- A USB hub in order to extend the Raspberry Pi 3 Model A+'s baseline capabilities

Documentation

The team currently manages a wiki (powered by Gollum) that exists alongside the fork of the codebase in GitLab. This structured wiki includes weekly status reports submitted to our advisor, sprint plans that document promised work, and sprint reports and retrospectives that analyze completed work and the team's process in order to reduce development risk and ensure incremental and sustainable progress is made during the course of the project. Additionally, the team uses GitLab's project management features to record PBIs, tasks, and time investments. This structure changed slightly towards the end of the project, as in order to increase sprint velocity, the team transitioned from written documentation of work to weekly verbal debriefs.

Status reports consist of time investments, descriptions of work completed that week per person, team findings, team successes, risk updates, and questions to our advisors. Sprint plans consist of a sprint goal and planned PBIs (with justification). Sprint plans are updated at the end of every sprint with the completed PBIs and their associated points in order to calculate a team velocity. Sprint retrospectives detail the team's process and progress. Typically these documents include different action items the team wants to improve on as well as how the team progressed on past action items.

All group based ABET-type requirements (such as technology reports, and outcomes) are documented in the wiki in order to maintain a single source of truth for artifacts relating to the project.

Documentation relating to spikes and the codebase are either recorded in the wiki or in each spike's associated README. The team has not made any major changes to existing code requiring documentation to exist in code.

Lastly, the team manages a shared Google Drive folder which stores long-format documents and sprint/term presentations for easier collaboration and editing. All finalized drafts are transferred to the project's wiki.

Managing Risks

There are many risks associated with our project, both in terms of development continuation and completion. Risks whose threat to the project is highest and most likely are listed below:

Product Owner

The Product Owner will lose interest in the project or be otherwise indisposed to provide useful feedback.

- Managing Risk**:** Even without proper feedback from the Product Owner, our team can continue to innovate new ways to interact with the table. In addition to our primary Product Owner, the team and team's advisor can together serve as a proxy product owner team to decide what areas of investigation and development are worth pursuing.
    - Update, 02-18-2021: Bruce has been cooperating with us well this far. He has attended most of our sprint reviews, but has missed some due to personal reasons. He has shown interest in the direction that our project is going.
    - Update, 05-13-2021: Bruce has stayed involved with the team for the entire duration of the project. He was happy to see the progress that we've made, and asked for the code so that he may use it for his future projects.

Legacy Code

Development on the product will become difficult or impossible due to the deprecated nature of Python 2.7, the outdated rpi_ws281x lighting library, the use of Node 8, and the fading support of the Cordova cross-platform web and mobile framework.

- Managing Risk**:** Look into upgrading libraries that can be upgraded. In particular, perform incremental upgrades as code areas are worked on and investigated.
    - Update, 02-18-2021: We have not upgraded the Python version nor the Node version, and these old versions haven't caused us major problems yet.
    - Update, 05-13-2021: We ended up needing to upgrade the Python version due to several dependencies requiring a new Python version. This was a massive hassle. An upgraded version of Raspbian needed to be installed on the Pi in order to have the new Python version run. The legacy code in siscloud also gave many problems. It took several months to get something as simple as a checkbox working correctly.

Development Environments

Local development environments do not mirror the production environment precisely, since developers don't have their own tables. Developers are developing locally with different Raspberry Pi versions and their own versions of Neopixel strips that are not identical to the strips used on a production table.

- Managing Risk**:** Developers can interface consistently about issues with their setups to ensure that when a developer faces an issue with their setup the issue is not replicated across setups. Once a person has their development environment completed, they can distribute an image from their SD card for a certain platform (e.g., the Raspberry Pi 4). This should help with consistency across setups. For those using Raspberry Pi 3's, a production image from a production raspberry pi can be used.
    - Update, 02-18-2021: Each developer has their own Pi running the Sisyphus code. This has worked mostly well, though one developer broke the Pi's SD card and needed to rebuild the environment again. Other than that, the team's development environments have been working well.
    - Update, 05-13-2021: Individual development environments ended up causing some problems. One team member runs their code on a Mac, which led to some OS specific issues needing to be resolved to make the code work. Another team member ran into severe issues with their computer and had to install a new operating system.

There is only one production environment. This production environment is located in a common lab that all developers have access to at MSOE (DH329); however, this lab has the potential of not remaining open if the institution must revert to complete-online instruction due to the pandemic.

- Managing Risk**:** A webcam has been placed on the production environment to enable remote development. SInce the team is able to exert some level of control over the production environment locally, in-person access does not need to be frequent. In the case of a total closure of the labs (no personnel permitted inside the building), the table can be migrated to a team member's house, or accommodations with MSOE to allow 1 team member into the building at an agreed upon interval of time can be arranged.
    - Update, 02-18-2021: The stream is still used for development. It has been helpful, as each developer is able to make changes to the table and see the effects real time without being in the lab.
    - Update, 05-13-2021: The stream has worked well. Along with the webcam, a microphone and speaker were set up so that music could be played on the table remotely.

Both local and production environments have proved to be quite brittle. Local setups behave strangely on occasion (interacting oddly due to apartment wiring issues) and the production environment has had issues with the LED lights shorting out.

- Managing Risk**:** The production table as well as the miniature table owned by Dr. Taylor had their lights replaced by Sisyphus Industries after the lights burned out. Our team has resolved to be more careful about plugging/unplugging the table to prevent future burnouts.
    - Update, 02-18-2021: The lights on the table are flickering once again, even though the team has been more careful with it. The team is not sure what is causing the flickering. Reducing the brightness seems to help, and the table will be turned off during the break to reduce further damage.
    - Update, 05-13-2021: The lights on the table were replaced again. The team is still not sure why the lights kept shorting out, and think that there may be a wiring issue with the table. Bruce has also said that their LED supplier has had issues in the past, and our faulty lights may be broken because of this.

In order to capture sound, USB microphones need to be introduced into the environment. The team will need to incorporate these microphones into their local environments, and the actual table as well.

- Managing Risk: In order to minimize errors, the team will all order the same microphone. That way, teammates can help each other troubleshoot errors that come up with the microphone, and there will not be any differences between how sound is captured.
- Update, 05-13-2021: The team did not end up following through with this, though team members using different microphones did not seem to be an issue.

Standards Used

- PEP 8 - code standard for Python code. Implemented with pylint.
- ES6 JS - code standard for Ecmascript code and Ecmascript features. Ensured with eslint.
- ISO 8601 - Date and time standard. Used whenever dates are present.
- GitLab CI - Pipeline manager. Runs with each commit.
- Git - Industry standard Version Control System
- Scrum - Industry standard Agile implementation