Smart Mirror

System Design Descriptions

Version 1.3

12/12/2017

Version History

Version #	Implemented By	Revision Date	Reason
1.0	Austin Nantkes	11/5/17	Initial Document
1.1	Austin Nantkes	12/7/17	Adding Sequence Diagrams
1.2	Austin Nantkes	12/12/17	Adding Class Diagram
1.3	Zena Abulhab	12/12/17	Final Touches

Table of Contents

List of Figures

1. Introduction
    1.1. Purpose

    This project aims to provide a comprehensive interactive mirror application for users' convenience. The mirror will recognize a user and display useful information for that user's viewing.

    1.2. Scope

    The software will recognize a user from their face using a camera and classifying system. The software will display a user's emails and upcoming events. It will give live weather reports and the time. An alarm will be added to be used by a voice recognition system to set and stop an alarm. Relevant news stories to a user will be pulled from a news website to be displayed. After a short period of time the screen will go into standby mode where a user face will awake it.

    1.3. Context

    This software is a Raspberry Pi based application, combined with a display. Since we utilize Raspberry Pi and a display which we already own, funding should not be a problem. Furthermore, development and maintenance does not cost money either.

    1.4. Summary

    This document will go over various aspects of this project including the design concerns, software architectures, and the dependencies of each component.

2. Identified Stakeholders and their Design Concerns
    2.1. Customer

Design Concerns

In this case, the customer base will be limited to a small number of people. But the customers will still have two primary concerns:

The image classification system must be reliable enough to differentiate a specific user's face consistently from other users', that is, the false positive rate should be small enough that it is negligible according to initial customer and developer standards to be established at a later date.

2.2. Project Team Member

Design Concerns

If one feature cannot be developed in a timely manner or at all how it impacts the rest of the system. Some features depend on others.

3. Design Viewpoints

3.1. Dependency Viewpoint:

Specific Design Concerns: The system will rely on multiple hardware and software components communicating at times simultaneously and at times sequentially. This means that our design will necessitate at least one-way dependency between most of the components in the system. These inevitable dependencies also mean that if there are any requirements changes over the development stage, there needs to be a clear definition of the components that depend on others and each other.

Design elements:

Vision Subsystem:

Recognition algorithm component:

Classification module

Post output for use with NodeJS framework

Display Subsystem

Google API component

Email module

Calendar module

Weather and time component

Dependencies attributes:

The primary dependency between the two subsystems will require the Display subsystem to listen for an applicable JSON message from the Vision subsystem. This means that the Vision subsystem must be able to successfully classify an image with a user's features. Then, the Vision subsystem will send a message to the Display subsystem with the detected user's key. The Display subsystem will then call on the required APIs to deal with displaying all of that user's information.

There exist less obvious dependencies in the architecture of the project that also require attention. These are important to point out because of the problems that could stem from failure in the following sets of components:

Hardware dependencies: The entire system is dependent on the successful functioning of the camera through which the user is recognized.

Software dependencies: A successful display is dependent primarily in being able to access a user's information through the APIs for google, weather, and time. If user authentication fails, or if one of these components cannot successfully access the required information, then the whole system fails and the error needs to be handled.

High Level Diagram of Dependency View (Not UML but should still be useful for visualization)

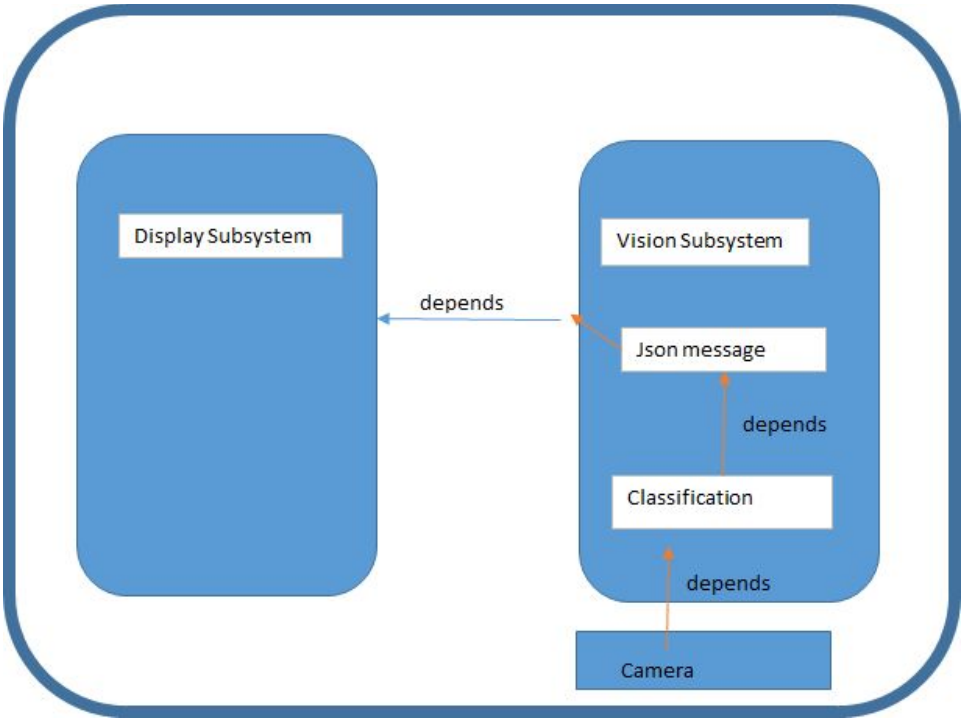

Figure 1. High Level Diagram of Dependency View.

3.2. Interface Viewpoint

Interface attribute: To trigger the mirror, the user enters the camera's view. To trigger personalized information (emails and calendar), the user's face is placed close to the camera so that it fills up the entire view. To break off the personalized information being displayed, the user's face moves away from the camera and when the face is far away enough, a threshold is reached where the personalized information disappears. If the user is no longer seen, the display times out and goes back to standby. If the user's face is not recognized, the screen displays a quick message that no known user is detected.

3.3. Interaction Viewpoint

Use Case 05 Diagrams:

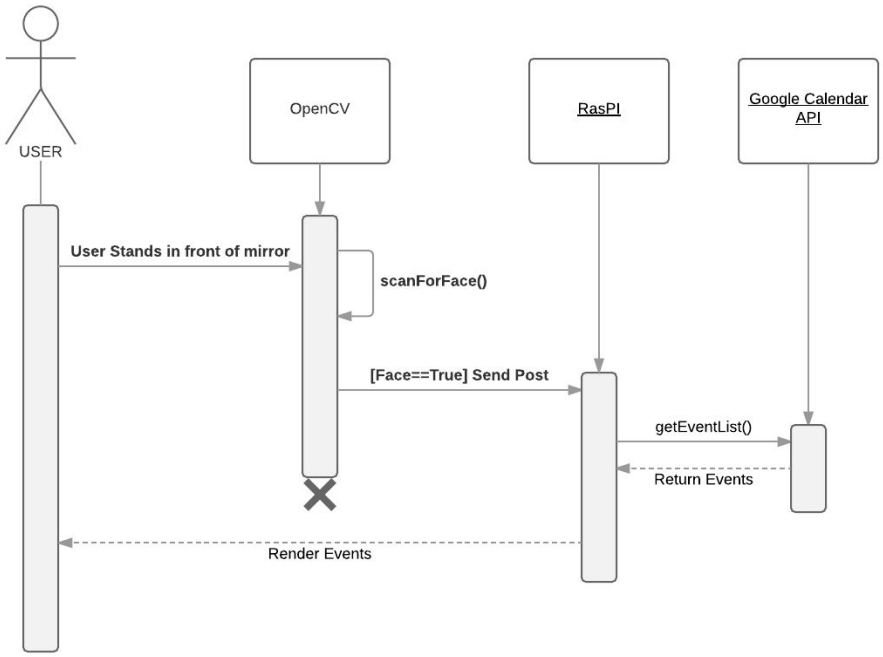

Figure 2. Sequence Diagram for UC05: Displaying the day's events

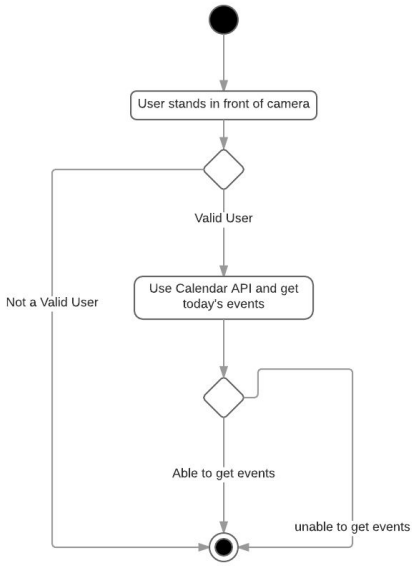

Figure 3. Activity diagram for UC05: displaying the day's events

Use Case 06 Diagrams:

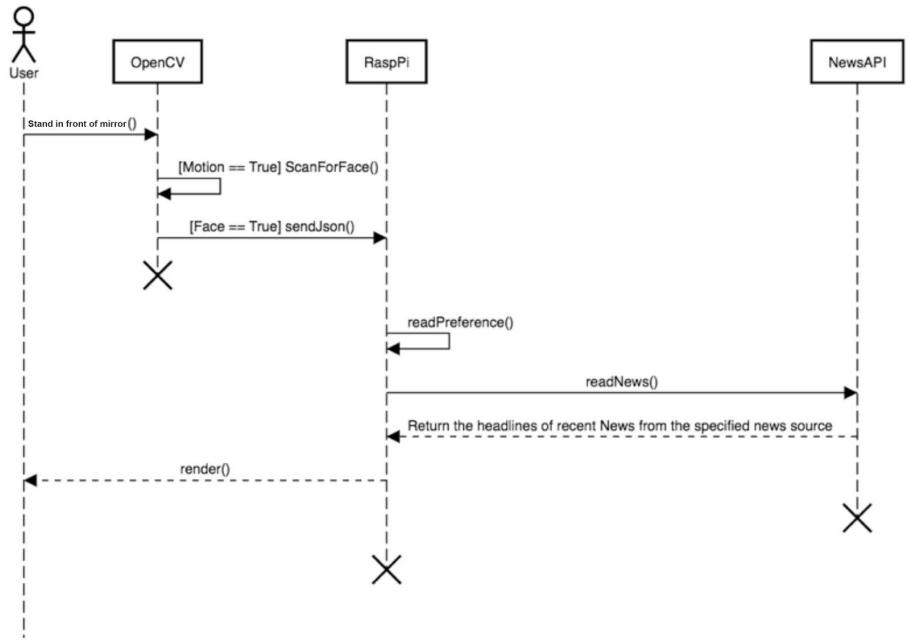

Figure 4. Sequence Diagram for UC06: Displaying the day's news

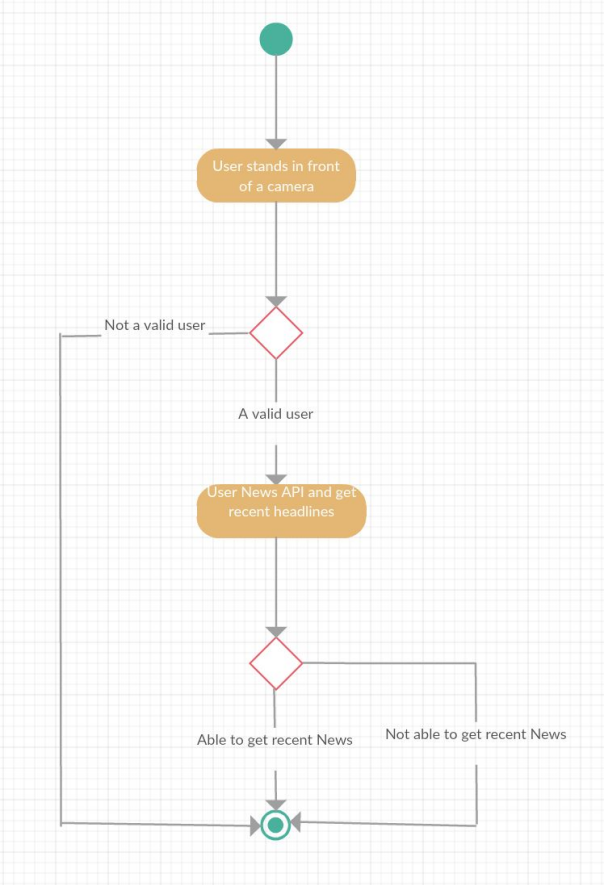

Figure 5. Activity Diagram for UC06: Displaying the day's news

Use Case 04 Diagrams:

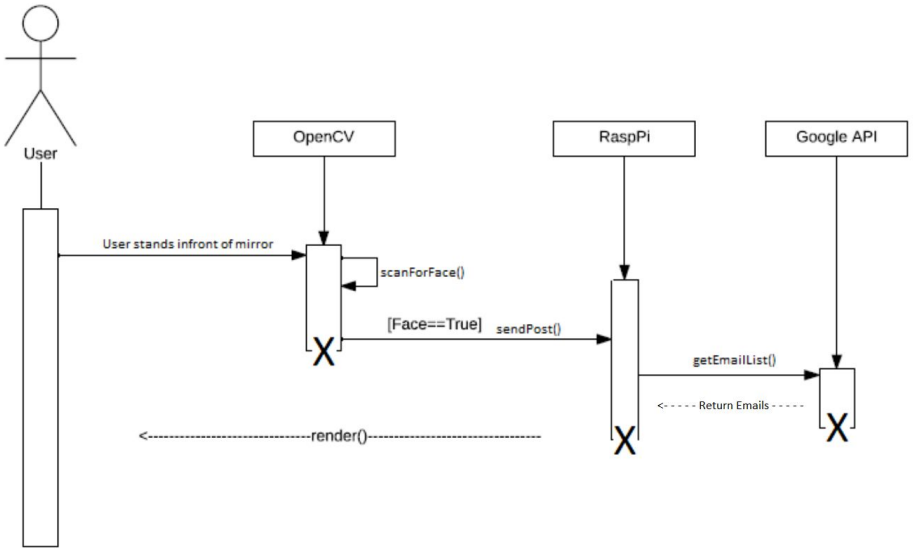

Figure 6. Sequence Diagram for UC 04: Displaying the user's emails

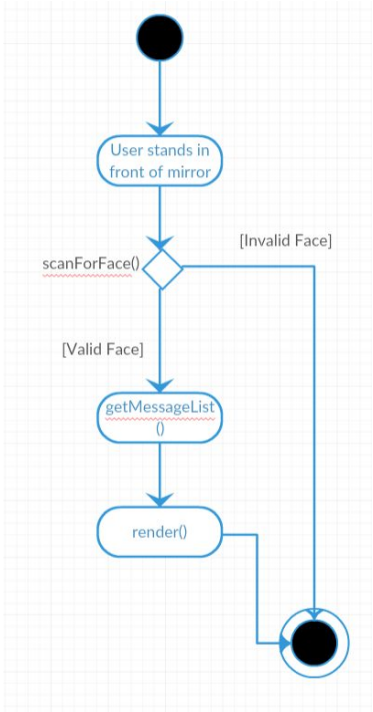

Figure 7. Activity Diagram for UC 04: Displaying the user's emails

Use Case 06 Diagrams:

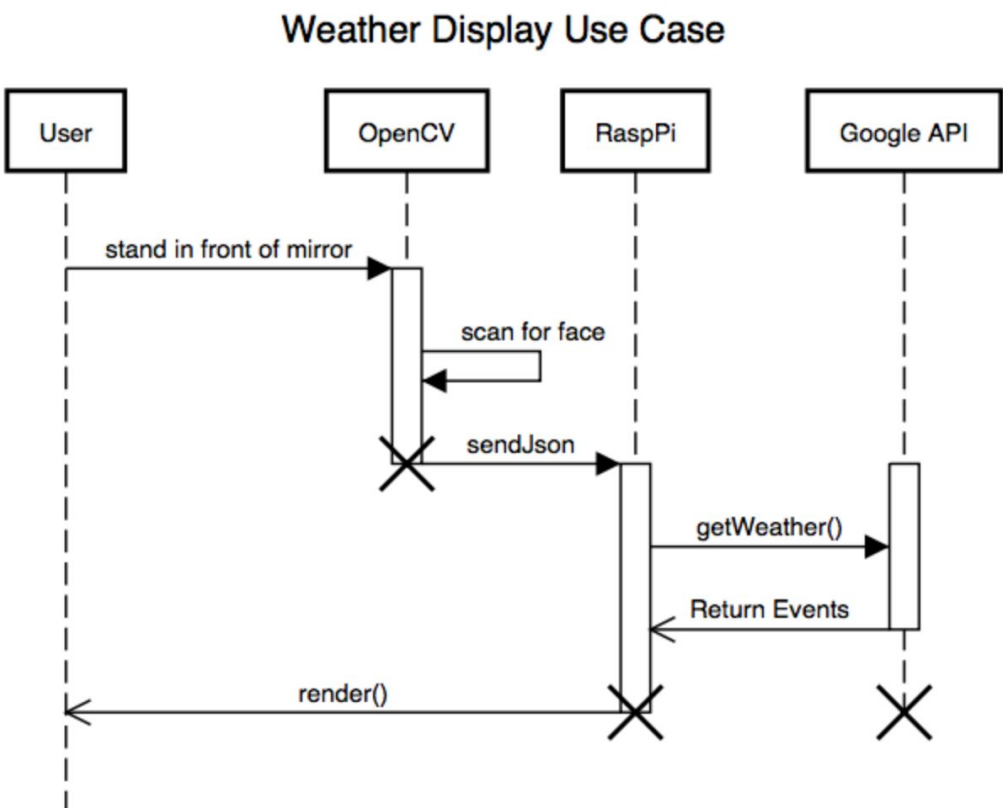

Figure 8. Sequence Diagram for UC 01: Displaying the weather

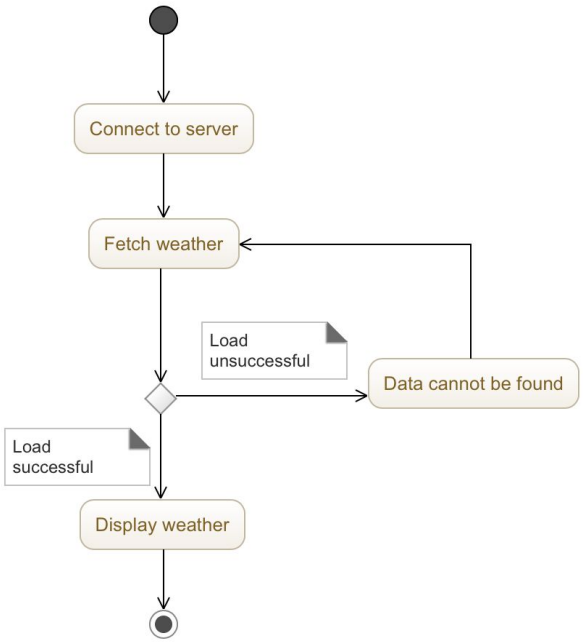

Figure 9. Activity Diagram for UC 01: Displaying the weather

3.4. State Dynamics Viewpoint

Design Concerns: The primary concern would be how quickly and accurately a state can transition from idle state to active state. To make this state transition robust, the face recognition system has to keep taking images and feeding them into the classification program every few seconds or so.

Design Elements:

Design Entities:

Idle state: On this state, the facial recognition system keeps taking images and runs a classification program at every certain interval of time. In terms of the interactions with a user, it displays the default features such as calendar and news

Face detection event: this occurs when the facial recognition system detects a face.

Active state: once face detection event happens, state changed from idle state to this state. On this state, it shows the person's latest emails and calendar.

Face lost event: this occurs when the facial detection system loses a human face. Once this happens, the state changed from active state to movement state

Movement state: On this state, the system waits to detect a human face again for a certain amount of time. Once it does not sense a human face for a certain amount of time, it changes the state from movement state to idle state.

Design Relationships:

State Diagram:

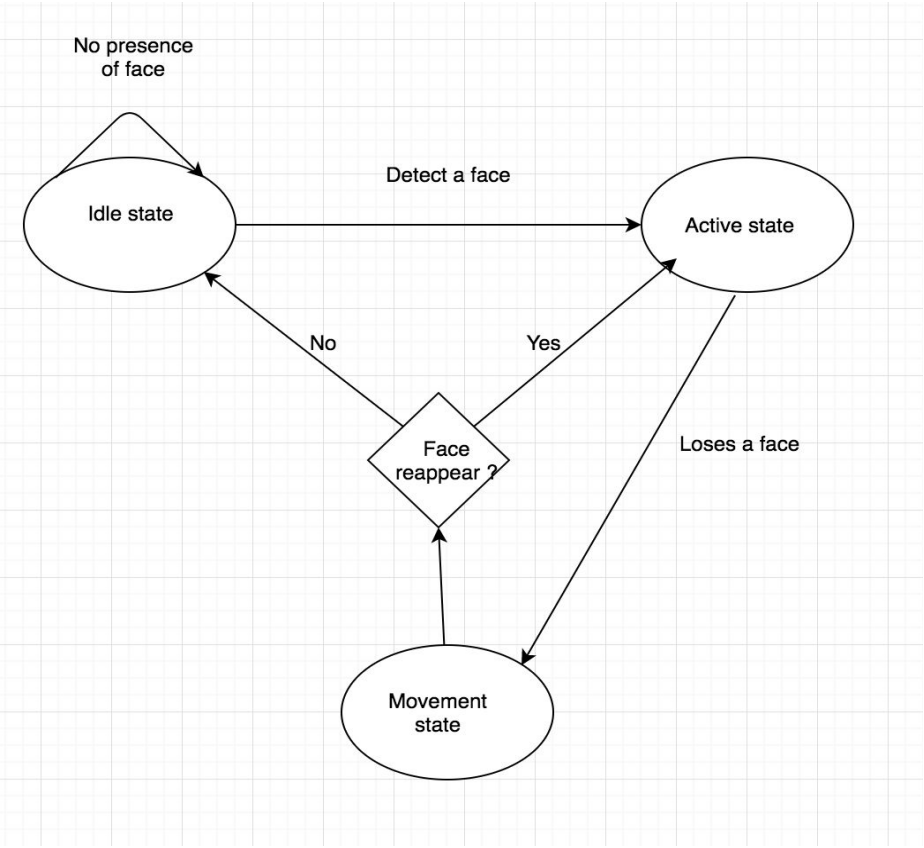

Fig 10. State Diagram

Design Attributes

This project infinitely changes its state from idle to active state based on the presence of a human face. As is shown in Figure 2, there are only two events in these states: detect a face and lose a face. The state transition is fairly simple, so the complexity of the state changes is unlikely to be a problem.

Design Constraints

One of the constraints would be the execution time of taking images and running facial detection program using them. Since the state change depends of the output of facial recognition system, its execution is likely to be the biggest constraint in the state transition.

3.5. Algorithm Viewpoint:

The current project does not actually require the team to consider time or space complexity of algorithms due to the relatively small amount of data processing that the Vision Component will undertake. Another thing that we are considering is that due to the time constraints we have, and the amount of things our team is doing as both engineers and students, planning the algorithm portion will have to be a more or less concurrent process with the actual development of the system. This means that this document will be revised to reflect the algorithm viewpoint that the project reflects after some development.

3.6. Logic Viewpoint:

Class Diagram for the whole system. NodeJS is an objected oriented language but not a class based language. These classes below are representations of what we think the classes would be in another language.

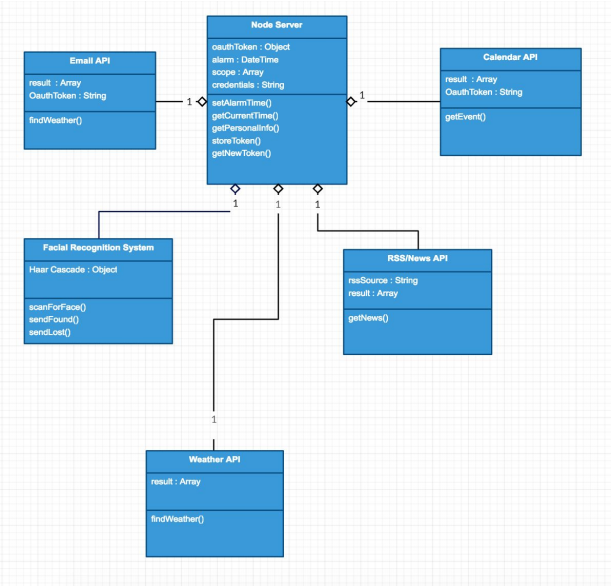

Figure 11. Class Diagram

4. Design Rationale

4.1. Architecture Rationale

We are using a client-server architecture because we need to connect to APIs that require server connections, such as the Google API. These enable the client to

connect to the local server.

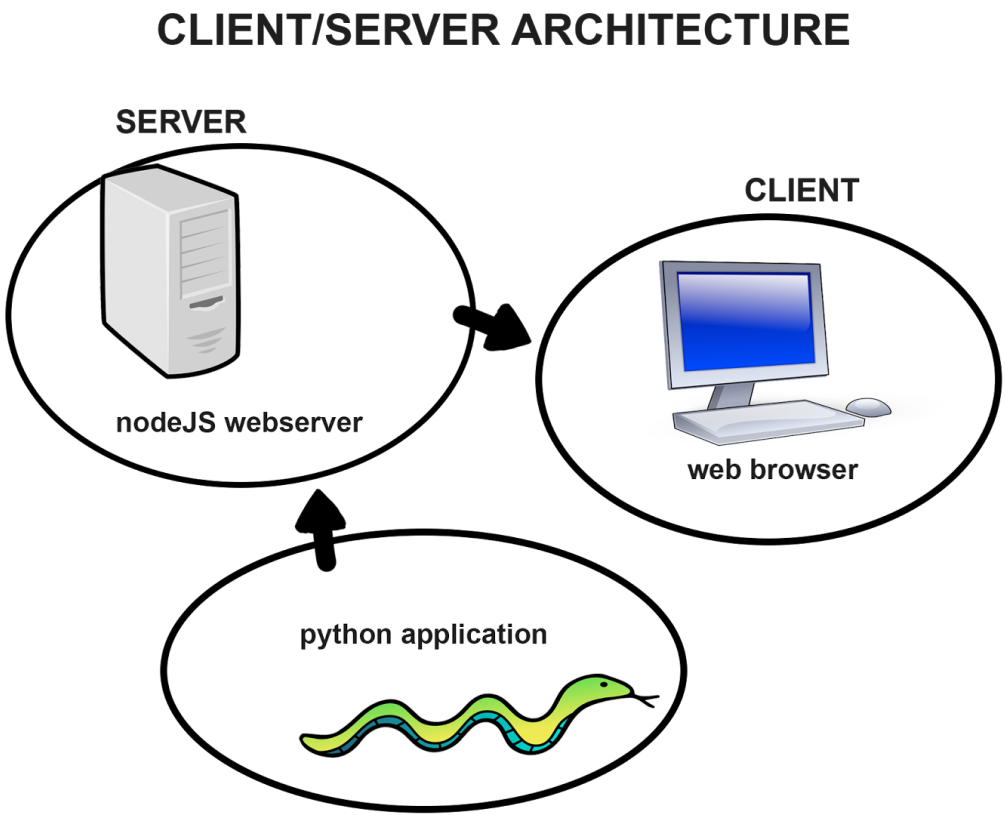

Figure 12. System Architecture Diagram. We used a Client/Server Architecture.

4.2. Hardware Rationale

We made the choice to use a mirror because this is something that almost everyone uses in the morning, which coincides with checking the weather, emails, and news, as included in the system.

4.3. Features Rationale

The features displayed on the mirror were chosen based on individual opinions on what is important to get information about at the start and throughout a day. The facial recognition aspect was deemed the most convenient way to allow for each individual user in a room to get their own email and calendar event information. We considered having a touchscreen, sound, voice recognition, and new face registration capabilities but decided with time constraints in mind that it would be better to save them for a later update. We decided to design the system this way

based on past experiences with using and working with facial recognition. We thought it would be best to have the system use visual and movement-based triggers rather than touch, voice, buttons, etc. because we could then reduce the amount of dependencies that allow the system to function.