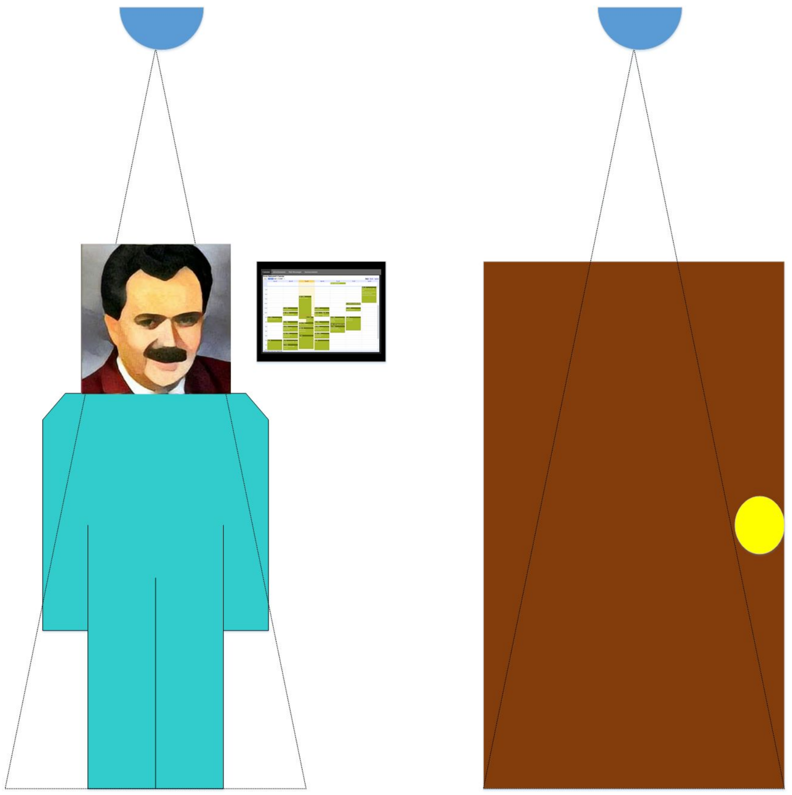
Internet of Things Smart Calendar Final Report

Advisor: Dr. Aleksander Malinowski

Cole Lindeman, Jason Morris



Abstract

Many university professors like to post their schedule and office hours right outside their office door so that students know when and where they can find them. Unfortunately, sudden changes to a professor’s agenda can make it hard to keep that schedule up to date. The Internet of Things Smart Calendar seeks to offer an easy solution. The Internet of Things Smart Calendar is a device that interfaces with Google Calendar and keeps an updated display of the professor’s Google Calendar. It provides the service of displaying a professor’s office hours, advertisements, and announcements to communicate relevant information that the professor would like the students to know about.

Creating an approach to this solution involved many hours of planning and preparation. Research was done on relevant projects that were similar to this one. It was decided that the project would be powered by a Raspberry Pi and a touch screen would be used as the display. The Raspberry Pi interfaces with motion sensors and makes decisions as to what should be displayed.

This solution utilizes new and up to date technology giving our professor an easier way to manage their schedule and provide their students with desired information. Producing it into a small, stable package allows the option for the project to be possibly duplicated for other teachers too.

Table of Contents

I. Introduction

The scope of this project is an Internet of Things Smart Calendar that organizes a collection of input connections and uses the information to display data as well as send messages.

The Smart Calendar displays the professor's calendar as well as advertisements and announcements on an interactive monitor outside his room. Sensors are used to recognize when students are nearby for tracking purposes and to control the display; the Smart Calendar is also able to display ads idly for passersby as well as display the calendar for people stopping at the Smart Calendar. There is also a feature to optionally leave text or email messages using the Calendar which is sent directly to the professor.

The Smart Calendar display is placed directly outside the professor's office while the main controller device is in the ceiling. Additionally, motion sensors are placed on the ceiling.

II. Review of Literature and Prior Work

Similar projects involving a smart calendar have been created. One person has created a project that displays their Google Calendar on one half of the screen, and the local weather on the other half [1]. It automatically shuts down at night and has buttons on the side to toggle the calendar between monthly, weekly, and daily views of the calendar. This particular project is powered by a Raspberry Pi. However, it does not have motion sensors or touch screen interactivity.

Another similar product that has been created is called the DAKboard [2]. The DAKboard is a customizable wall display that displays relevant information to the user's life, such as pictures, weather, and upcoming calendar events. This display also lacks motion sensors and a touch screen.

III. Subsystem Level Function Requirements

The Raspberry Pi 3 interfaces with all sensors and devices. It also communicates with the Internet using its onboard ethernet port. It has a power supply connected to the power inlet from a 120V conventional power line. The Pi rests inside a protective case in the ceiling.

The wires coming in/out from the sensors are attached to the Raspberry Pi's digital I/O pins. Power is siphoned out of the Pi's Vcc and Ground pins for the sensors' usage using more wires. The sensors are placed in necessary positions to be able to track required information. The wires reach the Pi from these positions.

The monitor for the display is mounted on the wall underneath the Raspberry Pi. The specific monitor ordered uses an HDMI cable, a USB cable, and its own USB power supply. The USB and HDMI cables reach the Pi and the power supply sources its own power from a 120V conventional power line.

An ethernet cable comes from Dr. Malinowski's office where it is connected to a router; the ethernet cable goes through a hole in the concrete wall in order to reach the Raspberry Pi. This cable will allow the Pi to access the Internet.

A. Inside the Pi

The Raspberry Pi has multiple functions to make meaningful information out of sensors, control devices, store data, and use the Internet. These functions include:

- GUI Process

This process handles the graphics going to the in/out connection with the touch display monitor connected via USB and HDMI. Some of the data displayed in the graphics (Doctor Malinowski's Google Calendar, class advertisements, announcements.) will be grabbed from the Internet directly or from a different process. The GUI Process runs automatically upon booting.

There is also interactivity for people to use the GUI to control the information being displayed using the monitor's touch controls; Users are able to choose what they want to display. When not being interacted with, this process defaults to displaying advertisements to passers-by; it switches to displaying the calendar for people who stop at the calendar when it is idly displaying ads. This process also handles the message interface which allows users to leave a message using software keyboard input typed via the touch screen. The messages are then stored in memory to be sent.

- Update Process

Automatic updates are made every ten minutes. This way, anytime the professor wants to update anything (changes in announcements, advertisements, etc.), they can update all of their new changes and the device can download them over the Internet.

- Tracking Process

In order for the board to make sense of all of the sensor data, it uses a process to read pin information and make decisions. The process gathers information from the motion sensors by reading the board's specific drivers for the digital I/O pins that the sensors are connected to. It then forms coherent, human-readable tracking data and sends it out to memory for storage.

The Tracking Process also tells the GUI Process when someone stops in front of the Smart Calendar and tells the Startup Process to turn on the monitor if needed. After 15 minutes of inactivity, this process tells the Startup Process to turn off the monitor.

- Memory

A portion of temporary read/write memory is allocated on the Raspberry Pi 3's main memory for temporary data. Among this data are temporary messages used for communication between processes and the tracking data. The tracking data is moved out and sent to the Internet at the end of the day.

- Startup Process

One process is not continually running, but is called whenever necessary for transitional periods. This process is divided into four sections; each section has its own instruction set for what needs to happen.

The monitor-off section is called to cut the power to the monitor by sending a serial signal to a relay connected through USB and shuts off the Pi's HDMI port. When activity is noticed again, the monitor-on section turns the monitor back on by using the opposite steps as the monitor-off section.

The start section is called to set up everything necessary for most of the other processes. In the temporary memory, it creates a message file for the GUI and Tracking process to communicate and it creates a tracking file for the Tracking process to dump information into. This section is also in charge of enabling the digital I/O pins for the Tracking Process.

Additionally, the Raspberry Pi restarts every day at 4AM as programmed from its internal clock scheduler. Before the restart is performed, the stop section is ran. The stop section formats the tracking file and sends it off to the Internet. This section also disables the digital I/O pins. When done, the device is safe to restart at any moment.

- Boot Process

Whenever the Pi boots up, the Boot process is ran by the internal clock scheduler. The boot process details a short list of instructions to be performed before letting the user interact with the display. This process calls the Startup process to shut off the monitor, calls the Startup process again to start, waits a minute, and finally calls the Startup process one last time to turn the monitor back on.

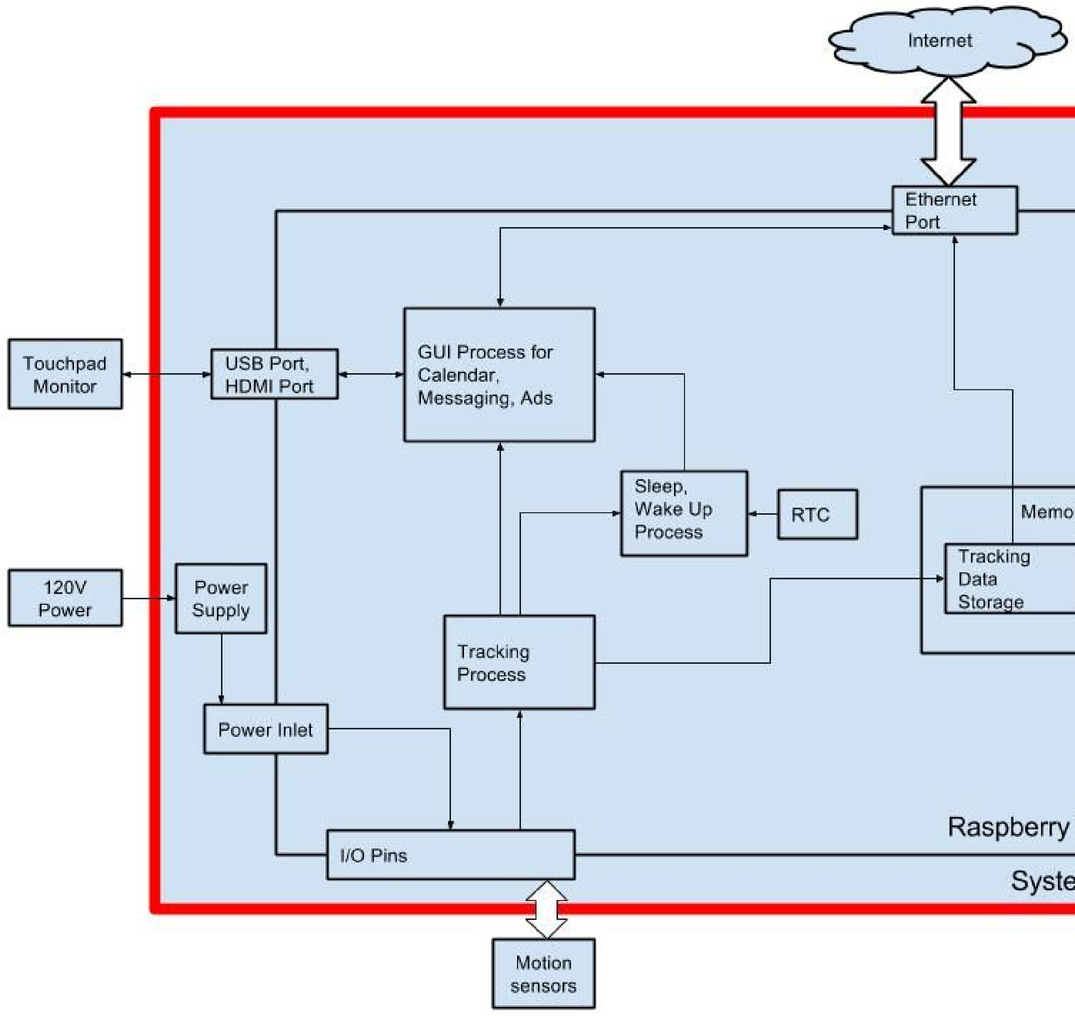


Figure 1. Subsystem Level Diagram
A diagram detailing the connections between subsystems and I/O.

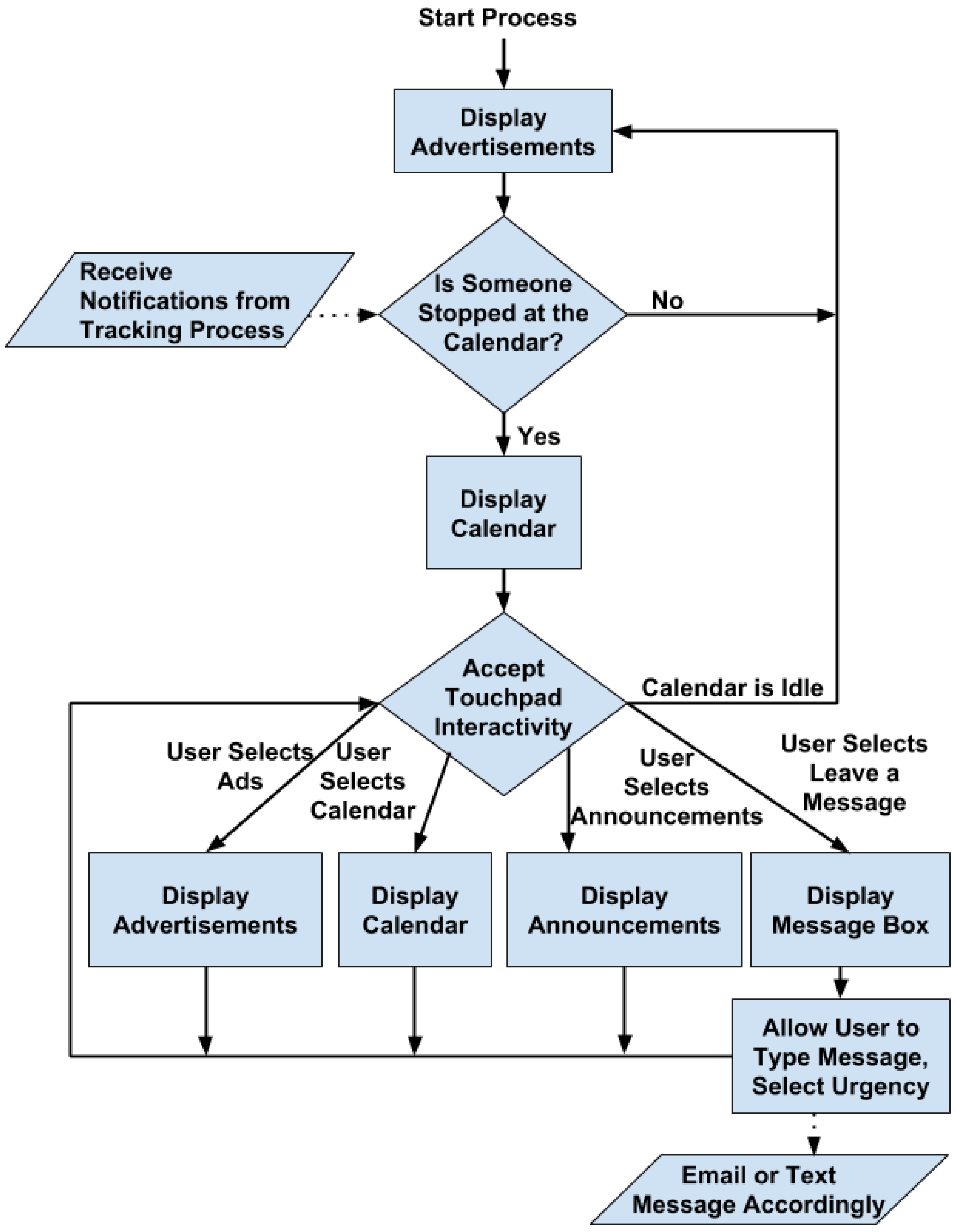

Start Process
Display Advertisements
Receive Notifications from Tracking Process
Is Someone Stopped at the Calendar?
No
Yes
Display Calendar
Accept Touchpad Interactivity
Calendar is Idle
User Selects Ads
User Selects Calendar
User Selects Announcements
User Selects Leave a Message
Display Advertisements
Display Calendar
Display Announcements
Display Message Box
Allow User to Type Message, Select Urgency
Email or Text Message Accordingly

Figure 2. Gui Process Block Diagram
A block diagram detailing the design for the process that controls what is displayed. The process also handles messaging.

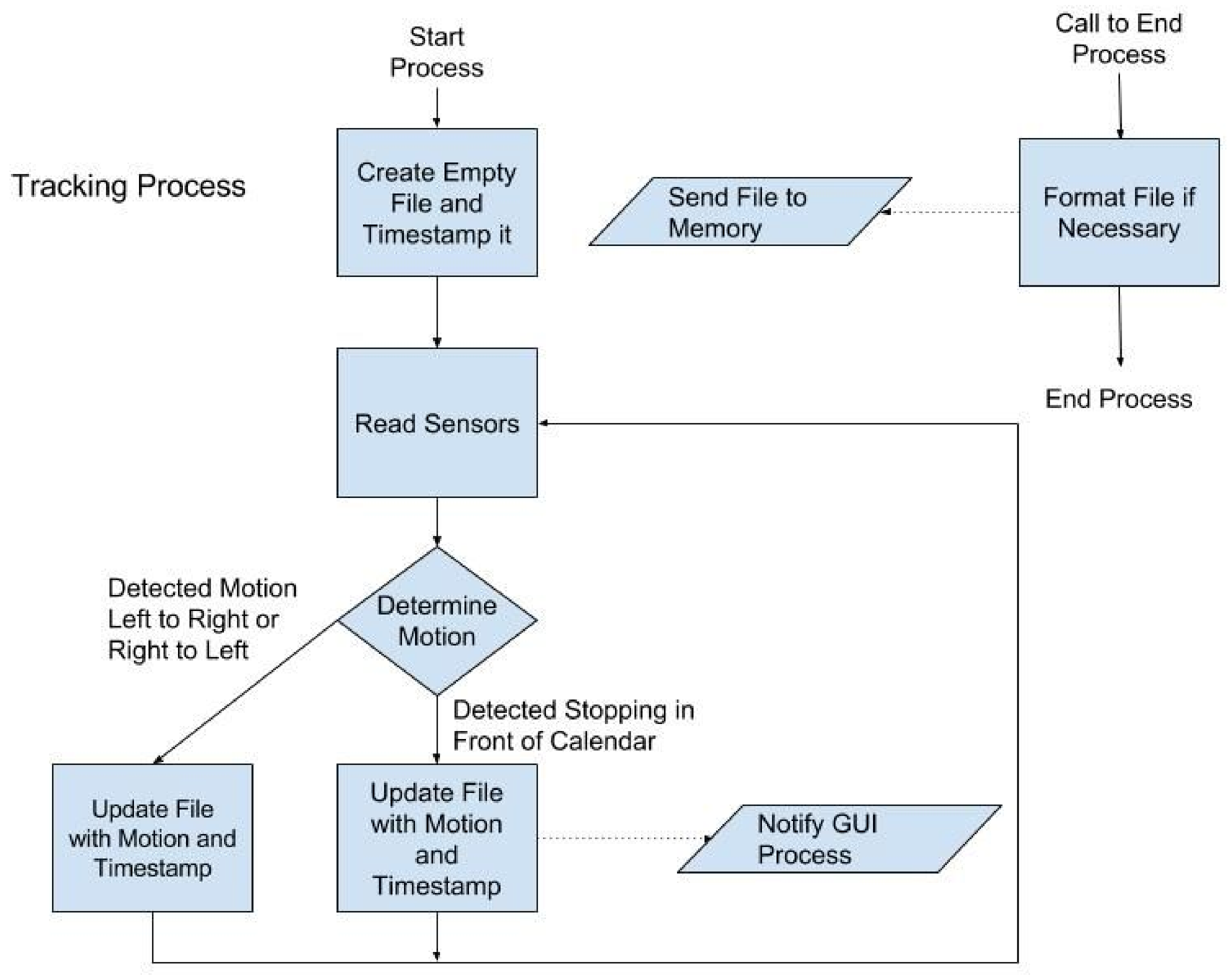

Tracking Process
Start Process
Create Empty File and Timestamp it
Read Sensors
Determine Motion
Detected Motion Left to Right or Right to Left
Detected Stopping in Front of Calendar
Update File with Motion and Timestamp
Update File with Motion and Timestamp
Notify GUI Process
Call to End Process
Format File if Necessary
Send File to Memory
End Process

Figure 3. Tracking Process Block Diagram
A block diagram detailing the design for the process that reads the infrared sensors and sends notifications to other processes. The process also creates tracking data and uploads the data to an online database when it is told to end.

B. Modes of Operation

In order to conserve power, the Smart Calendar will also enter modes of operation where the system sleeps.

- Off

The device is completely powered off. This happens briefly while restarting or under abnormal circumstance if the device is manually unpowered or if the power goes out. From this mode, the device needs to receive power again to turn back on if the power was cut; this requires fixing whatever happened. Either after resetting or once it receives power, it turns back on and enters Startup.

- Startup (transition)

The GUI is loaded in kiosk mode and the display is briefly off and the sensors are turned on; this transition is for loading features out of being Off. The Pi automatically enters Startup from Off when booting. When this transition is finished, the device is in the Powered On mode.

- Powered On

The device is actively retrieving all inputs, displaying functionality on the monitor, and deciding if to send alerts. This mode is for implementing the Smart Calendar features. If the device finds that nobody has been around for over 15 minutes, it will enter the Lie Down transition.

- Lie Down (transition)

This transition is exclusively for shutting down the display. When it finishes, the device enters Low Power mode.

- Low Power

The device is in a power saving mode with the display is off while the sensors stay on; this mode is to save power for when nobody is around anyway. The Raspberry Pi cannot shut off completely because it cannot wake itself up. When someone is sensed, the device enters Wakeup.

- Wakeup (transition)

This transition is exclusively for turning on the display. When it finishes, the device enters Powered On mode.

- Shutting Down (transition)

The device sends stored tracking data to the Internet, and all inputs and outputs are shut down. This transition is for stopping all features and enabling the system to be safely restarted before it is actually restarted. The Pi automatically enters Shutting Down based on time of day at 4 AM.

IV. Subsystem Level Design and Implementation

See the ***Appendix*** for source code.

- Operating System

Ubuntu MATE is used for the Raspberry Pi’s operating system. It is installed on the Raspberry Pi’s SD card alongside the hard drive. Ubuntu MATE is a lightweight version of the Ubuntu operating system for small devices. The version used for this project is specifically tailored for Raspberry Pis.

- GUI Subsystem

The GUI was created using HTML, CSS, JavaScript, and PHP. The GUI displays a section for calendar, messages, advertisements, and announcements. Each section has its own corresponding HTML file and web page.

Most of the styling for the GUI was done using CSS. A black and grayish theme was chosen that has the tabs at the top for the user to navigate between the different pages.

The HTML web pages are displayed using Mozilla Firefox. The homepage is set to the Calendar page; whenever it is started, the Google Calendar is displayed. There are a couple of extensions installed into Firefox. The main extension that is constantly running is R-kiosk; this extension enables kiosk mode and vastly limits the unnecessary features of Firefox for this project.

Firefox starts on bootup and displays the calendar in kiosk mode. This was done by adding Firefox to the “startup applications preferences”. The application does not start immediately as it is slow to load. While loading, the device shows the desktop, but the monitor is disabled and inoperable, so no user can actually see the display on when it is not running Firefox.

For the Calendar page, the professor’s Google Calendar was embedded into the HTML web page (index.html). The external links for the calendar were removed using the AdBlock Plus extension so that users cannot break the system.

The advertisements page (Ads.html) alternates between displaying different advertisements (saved as pictures) every five seconds. The picture changing functionality was done using JavaScript (SlideFunctionality.js). Whenever this web page is idle for longer than 90 seconds, it actively reads a message file being updated by the tracking script using AJAX. Whenever the web page is told someone stopped at the Calendar, it will redirect to the Calendar page. Additionally, a Python script (images_Grab.py) was created that grabs any of the advertisement picture names in the images folder and throws it into the advertisement HTML page while replacing the old images.

The Calendar, Announcements, and Messenger pages use idle redirecting; this functionality was created using JavaScript (idleDirect.js). It directs any page to the Advertisements page after being idle for a period of 90 seconds.

A PHP script (send_Message.php) was used for sending email and text messages to the professor. The HTML for the messenger page (Mali_messenger.html) allows the user to type out their message on the touchscreen. The FxKeyboard extension allows the user to type out their message on the touchscreen.This message is then sent to the

PHP script using HTTP post. The PHP script then sends this message as an email to either the professor's email address or phone number address.

- Web Server Subsystem

An Apache2 web server was installed on the Raspberry Pi. The web server starts on boot and locally hosts the HTML web pages. The web server is necessary for PHP and AJAX functionality.

- Python Background Script Subsystems

Python scripts were used for most every feature running behind the scenes. From startup to shutdown, there is at least one script running.

On booting up, an initialization script is ran. This script runs the start and monitor control sections of the startup script to start all features. This script uses the monitor control to turn off the monitor while the Ubuntu MATE Desktop would normally be shown for a minute. This full minute is the maximum time it takes for the GUI Subsystem to load from boot. No user should be able to mess with the system during this one minute long loading time.

The startup script is divided into four different sections. The first two sections are for starting and stopping the other background scripts; the other two sections are for turning the monitor on and off.

The starting section is additionally responsible for creating temporary files inside of the system's allocated temporary memory; these files are the timestamp for tracking and the message file for AJAX communication. Finally, the starting section enables the pins that the motion sensors are connected to by turning them on and setting them to input.

The stopping section stops the other background scripts by finding the process and sending it the kill signal. Once that is done, it disables the motion sensor pins by turning them off. Lastly, it finalizes the temporary tracking file and sends it to the Internet.

The monitor control sections are able to control the monitor by both disconnecting the monitor's power and disabling the HDMI port. Disabling the HDMI port saves power for the Pi and can be done with two simple commands, but doing this does not turn off the touchscreen monitor as its backlight stays on. This makes it necessary to disconnect the monitor's power so that it does not waste energy and/or melt. This is why the monitor control sections are able to communicate with a USB relay connected to one of the Raspberry Pi's USB ports. This relay cuts in between the monitor and its power supply and is able to turn it on and off. In order for the Python script to communicate with the relay, it establishes a serial connection using the pySerial package and the Pi's USB tty channel and sends a signal. The two different signals sent are for either opening or closing the relay thereby powering or depowering the monitor.

The tracking script is a continually running finite state machine design Python background script that is constantly reading the GPIO pins that the sensors are attached to. It is in charge of looking for any activity noticed by the motion sensors. Any activity it sees is determined by the two motion detectors mounted on the ceiling; one sensor is above the display and the other is above the professor's door. This script can determine which area the activity started and ended and can determine if the activity involved someone physically stopping at the door and/or display. All of this information is formatted into the temporary timestamp text file as human-readable text. If any activity has happened within the last 90 seconds, this script will communicate with the GUI Subsystem by writing to the temporary message file used for communication. If no activity has happened within the last 15 minutes and the monitor is off, the script will use the startup script to turn off the monitor; this script will also turn on the monitor if the opposite case is true.

- Update Subsystem

We are using Git for updates. To do this, a command was added to the crontab on the Raspberry Pi that does an automatic git pull from the Calendar's master branch once every ten minutes. This way, anytime the professor wants to update anything (changes in announcements, advertisements, etc.), they can just push all of their new changes to the master branch.

V. Parts List

- Waveshare 10.1 inch 1280x800 IPS LCD Capacitive Touchscreen with case
    - $118.99
- Raspberry Pi 3 with power supply, case and heatsinks
    - $51.94
- Sandisk 32GB microSDHC card with normal SD card adapter
    - $10.59
- Emy passive infrared motion sensor detector modules
    - $5.49
- Ethernet*, HDMI, USB and digital I/O cables*
    - $14.89
- SMAKN LCUS-1 type USB relay module*
    - $0.00

Total cost: $201.90

*Provided in house and is not included in price.

VI. Major Changes from Prior Models

In the initial project bid, there were a couple of features that did not make it into the final product; this included two major features. One was a keyless room entry where the Smart Calendar would control the door lock and let the professor in the office. The other feature was GPS tracking where the Smart Calendar would know the professor's GPS coordinates and would be able to make decisions based on that information.

The door lock functionality was an idea that was scrapped in the early planning phase of the project. The University did not allow tinkering with the office door. This feature was not worked on at all.

The professor tracking functionality was talked about early on, but never came to fruition due to technical and time constraints. There were some general ideas floating around. One idea was having an application on the professor's phone that would constantly send his GPS coordinates to the Smart Calendar. Looking for the professor's MAC address on the network was also discussed. Lastly, adding magnetic sensors to the professor's door to see if it was open or not was discussed; these sensors would allow the system to assume whether or not the professor was in his office. These ideas were not tested due to running out of time. The door sensors were ordered but never used. Not having this functionality disallowed further work into a special alert. This alert would tell the professor if a student is waiting at the office door during office hours if the professor is not in the room.

One last feature that was in the initial bid was the feature for ads to follow passersby. There were plans to do this and was time allotted in the schedule, but it was never implemented due to technical constraints and the need to move along with the project.

It was planned to use Google Chrome as the web browser. However, Chrome did not have any extensions needed. Because Firefox had the extensions needed, the browser was switched to Firefox.

During the early stages of the project, it was not realized that a web server was required to implement some functionality. Initially, it was planned to host a site on a local port for the GUI features in order to embed the Google Calendar, but this embedding worked without a web server as the browser could directly use the calendar HTML file just fine allowing the project to continue without needing a web server.

Originally, it was planned for the Python background scripts to directly control the browser GUI Process. This initially worked when testing on virtual machines, but did not work upon trying to implement the control system on the Raspberry Pi; the source of this problem was that the controller used was only for 64-bit operating systems. The only other available option found was to control the browser was with Javascript.

The unfortunate part about Javascript is that it is restricted to run in a very tight, secluded area and cannot communicate directly with any process other than the HTML referencing it. There is one exception to this rule: Javascript can make XML requests if and only if the request is for data originating from the same website it is currently running from. The combination of using Javascript and XML requests is called AJAX. These requests do not work if the browser it is run on is referencing an HTML file on the hard drive instead of a website with the same files. A quick solution was using the SimpleHTTPServer Python module to create a script and throwing everything onto that local port server.

Around the same time, work was being done on PHP scripting to email and text messages from the system. This required a web server because PHP scripts will not work without one. It was decided to install the LAMP software package as most of the software inside of it was needed. The LAMP package for Ubuntu (as well as Ubuntu MATE) installs Apache2, MySQL and PHP. MySQL is not required for this project, but the other two softwares are required. Apache2 is a web server hosting software that is customizable and lightweight. It was used to host the GUI on a local port server.

Although both web servers worked, it was decided to stick with Apache2 and not use the SimpleHTTPServer as the Apache2 software was much more powerful and useful.

VII. Schedule

Table 1
Planned Schedule for Completion and Actual Schedule

		Planned Schedule		**Actual Schedule**	
Week		**Jason's work**	**Cole's work**	**Jason's work**	**Cole's work**
1/15/17	1/21/17	**Spring Semester begins** Write Python code to host HTTP web server for AJAX to communicate with		Setup Raspberry Pi Setup local Apache2 web server	Researched temporary file storage systems and AJAX
1/22/17	1/28/17	Write XML code using AJAX to direct browser	Continue writing Python code to communicate with AJAX	Continued setup of local web server Troubleshooted message sending with PHP Script	Implemented temporary file storage system Implemented AJAX and Python web server

1/29/17	2/4/17	Write HTML code to direct browser back to ads when idle for long enough	Setup Raspberry Pi Setup monitor for Pi	Replaced Chrome browser with Firefox	
				Continued fixing email message sending	Researched commands for monitor control
2/5/17	2/11/17	Write JavaScript for ads that “follow” passersby	Figure out reading, writing, and permissions for I/O pins Connect sensors to Pi	Cleaned up messaging Got Firefox running in Kiosk mode on bootup Wrote script to grab ad files and put them into ads page	Added and configured Firefox extensions
2/12/17	2/18/17		Write Python script to poll I/O pins Write Python script to enable and disable I/O pins	Wrote JavaScript to redirect page to calendar if idle Added command that does frequent Git updates Rewrote image grabbing script Added announcements page	Wrote JavaScript to whitelist allowed pages
2/19/17	2/25/17	Find method to upload text files Write script to use method to upload tracking text file	Write Python script to track movement with IR sensors	Blocked external links on google calendar	Tested motion sensors and GPIO pins
2/26/17	3/4/17		Write Python script to compile movement information into a text file	Talked to lab directors about mounting Smart Calendar	
3/5/17	3/11/17	Write Python script to send commands to Ajax using movement information	Write Python script to communicate with door lock	Split and stripped power cable to measure devices	Started work on background script drafts
				Measured all power ratings and gave them to lab directors	
3/12/17	3/18/17	**Spring Break**			
3/19/17	3/25/17	Test Internet communication	Write script for sleep/wakeup process	Started work on website	Refined background scripts

3/26/17	4/1/17	Test mount setup for project	Worked on web site and poster.	Further refined background scripts
4/2/17	4/8/17	Mount project	Finished project poster	
			Continued work on website	Added USB relay control
4/9/17	4/15/17	Spare time in case of changes	Finished project poster and went to Student Expo	
4/16/17	4/22/17	Spare time in case of changes	Worked on final report and presentation	

VIII. Future Directions

If any additional work is put into the Smart Calendar, it can include adding any of the previously discussed ideas in the initial project bid.

Many other addon features can still be implemented. The first one likely to be implemented would be a better interface for updating the calendar. Many professors do not know what Git is. So telling them to change files then commit and push the changes to the master branch might not be easy for them.

One solution would be to create an application that gives the professor an easier way to update the calendar. The application could then commit and push the changes based on what the professor wants. This would be especially useful for updating the announcements and advertisements page. For the announcements page, the professor could type in the heading and text for each announcement. They would also have the option to delete announcements. A script would then take this text and throw it in to the Announcements.html page. These changes would then be committed and pushed. The advertisements page would have similar functionality that would allow professors to upload or delete images.

Another way to solve the Git issue is to use Google Drive. Additional software would need to be installed in order to use this. This method would be functionally the same as using Git, but with a system professors would be more comfortable with.

There was also talk on adding new tabs to the Smart Calendar for added interactivity. These features could be local weather information, local weather alerts, or a simple game.

Additionally, there was interest in making a companion phone application to the Smart Calendar. The application would have many of the same features of the Smart Calendar, but for a mobile platform.

VIII. Applicable Standards and Patents

A. Standards to Consider

Since this project involves electrical wires, a large concern is safety. The project will be installed on a wall in a building where many people walk by and possibly interact with the device. OSHA has a myriad of safety standards for electronic equipment and electrical wiring. These standards must absolutely be adhered to as installing the device in the building could be breaching building code and would have to be taken down. As an example, OSHA standard 1910.305(f)(1) demands any “conductors used for general wiring shall be insulated” which means that anything used has to be jacketed or covered somehow so no conductors are exposed [3].

It would also be a good plan to follow one or more sets of cyber-security standards as the device will connect to the Internet and could potentially have a multitude of possible security breaches. If the device or any database that the device uses is comprised, unwanted advertisements and announcements could appear on the Smart Calendar.

Some helpful tools applicable to solving functionality are JavaScript, HTML and CSS which all fall under W3C standards [4]. Any web applications should be designed and tested with the wealth of standards, specifications and guidelines provided by this platform. These standards are intended to improve quality whenever applicable and are also free to view.

B. Patents for Similar Ideas

There are some patents for features on electronic calendars that could possibly be used to implement required functionality. In order to avoid possible patent troubles, the features should be implemented using different methods.

One method that cannot be used is using email to update the Smart Calendar. This method is patented by Xerox and is very specific on patenting the idea of updating an electronic calendar with information in an email message [5]. Luckily, there are plenty of other options on how to update the Smart Calendar. Avoiding email updates altogether could be possible using a direct update link or by having a website host retrievable updates.

There are no current plans to obtain a patent for any of the other methods used for the functionality. The project could possibly later be packaged as a product as well as reproduced and redistributed; in order to allow this to be possible, no patents can be violated.

IX. References

Citations

[1] Kmccb (2016, Apr. 7). *Raspberry Pi Framed Informational Display - Google Calendar, Weather, and More..* [Online]. Available: http://imgur.com/gallery/z94Vr

[2] M. Archambault. *DAKboard Is a Customizable Wall Display for Photos, Calendar Events, and Weather* [Online]. Available: http://petapixel.com/2015/08/19/dakboard-is-a-Customizable-wall-display-for-photos-calendar-events-and-weather/

[3] (2007, Feb. 14). *Wiring Methods, Components, and Equipment for General Use* [Online]. Available: https://www.osha.gov/pls/oshaweb/owadisp.show_document?p_table=STANDARDS&p_id=9882

[4] *Standards - W3C* [Online]. Available: https://www.w3.org/standards/

[5] JL. Meunier, C. Hagage, S. Castellani, D. Proux, E. Cheminot, F. Segond, “System and method for updating an electronic calendar*”* U.S. Patent 9 436 649, Sept., 6, 2016.

Additional References

[6] D. J. Barrett, *Linux Pocket Guide,* 2nd ed. Sebastopol, CA: O'Reilly, 2004.

[7] *PHP 5 Tutorial* [Online]. Available: http://www.w3schools.com/php/default.asp

[8] (2016). *Python 2.7.12 Documentation* [Online]. Available: https://docs.Python.org/2.7/

[9] *Linux Documentation* [Online]. Available: https://linux.die.net/

[10] *Ajax jQuery API Documentation* [Online]. Available: http://api.jquery.com/jquery.ajax/

Appendix: Software Source Code

1. Graphical User Interface

index.html

```
<!DOCTYPE html>
<html>

<head>
    <title>Smart Calendar</title>
    <link rel="stylesheet" type="text/css" href="./CSS/Style.css">
</head>

<body>
    <!--tabs at the top-->
    <ul class="toptabs">
        <li id="active"><a>Calendar</a></li>
        <li><a href="Ads.html">Advertisements</a></li>
        <li><a href="Mali_Messenger.html">Mali Messenger</a></li>
        <li><a href="Announcements.html">Announcements</a></li>
    </ul>

    <!--all page content-->
    <div class="content">
        <iframe
src="https://calendar.google.com/calendar/embed?title=Doctor%20Malinowski%27s%2
0Calendar&amp;showPrint=0&amp;showCalendars=0&amp;mode=WEEK&amp;height=600&amp;
wkst=1&amp;hl=en&amp;bgcolor=%23FFFFFF&amp;src=olekmali%40fsmail.bradley.edu&am
p;color=%235F6B02&amp;src=en.usa%23holiday%40group.v.calendar.google.com&amp;co
lor=%23AB8B00&amp;ctz=America%2FChicago"
            style="border-width:0" width="1240" height="700" frameborder="0"
scrolling="no"></iframe>
    </div>

    <script src="./JavaScript/idleDirect.js"></script>
</body>

</html>
```

Ads.html

```
<!DOCTYPE html>
<html>

<head>
    <title>Advertisements</title>
    <link rel="stylesheet" type="text/css" href="./CSS/Style.css">
</head>

<body>
    <!--tabs at the top-->
    <ul class="toptabs">
        <li><a href="index.html">Calendar</a></li>
        <li id=active><a>Advertisements</a></li>
        <li><a href="Mali_Messenger.html">Mali Messenger</a></li>
        <li><a href="Announcements.html">Announcements</a></li>
    </ul>

    <!--all page content-->
    <div class="content">
        <!--images-->
                   <img class="mySlides" src="./images/ad2.png"
style="width:96%">
                   <img class="mySlides" src="./images/ad1.png"
style="width:96%">
                   <img class="mySlides" src="./images/ad3.png"
style="width:96%">
            <!--images-->
    </div>

    <script src="./JavaScript/SlideFunctionality.js"></script>
    <script src="./JavaScript/activeDirect.js"></script>

</body>

</html>
```

Announcements.html

```
<!DOCTYPE html>
<html>

<head>
      <title>Announcements</title>
      <link rel="stylesheet" type="text/css" href="./CSS/Style.css">
</head>

<body>

      <!--tabs at the top-->
      <ul class="toptabs">
      <li><a href="index.html">Calendar</a></li>
      <li><a href="Ads.html">Advertisements</a></li>
      <li><a href="Mali_Messenger.html">Mali Messenger</a></li>
      <li id="active"><a>Announcements</a></li>
      </ul>

      <!--all page content-->
      <div class="content">
            <h1>Today's Announcements</h1>
            <p>Today we will have a meeting about the Smart Calendar</p>
      </div>

      <script src="./JavaScript/idleDirect.js"></script>
</body>

</html>
```

Mali_Messenger.html

```
<!DOCTYPE html>
<html>

<head>
      <title>Mali Messenger</title>
      <link rel="stylesheet" type="text/css" href="./CSS/Style.css">
</head>

<body>

      <!--tabs at the top-->
      <ul class="toptabs">
      <li><a href="index.html">Calendar</a></li>
      <li><a href="Ads.html">Advertisements</a></li>
      <li id="active"><a>Mali Messenger</a></li>
      <li><a href="Announcements.html">Announcements</a></li>
      </ul>

      <!--all page content-->
      <div class="content">
      <form action = "send_Message.php" method = "POST">
            <p>Message: </p>
            <p><textarea rows="9" cols="67" name="message"></textarea></p>
            <p>Message type:&nbsp;<input type="radio" name="messageType"
value="Regular" checked />&nbsp;Regular (email)&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
            <input type="radio" name="messageType" value="Urgent"/>&nbsp;Urgent
(text)
            <input type = "submit" name = "send button" value = "SEND"></p>
      </div>

      <script src="./JavaScript/idleDirect.js"></script>
</body>

</html>
```

Style.css

```
body {
      background-color: hsl(0, 0%, 14%);
      color: white;
      margin:0;
      padding:0;
      overflow: visible;
}

html {
      margin:0;
      padding:0;
}


/* top TABS STUFF */
ul.toptabs {
      position: fixed;
      list-style-type: none;
      margin: 0;
      padding: 0;
      border: 1px solid #8c8c8c;
      background-color: #666666;
      width: 100%;
}

/* Float the list items side by side */
ul.toptabs li {float: left;}

/* Style the links inside the list items */
ul.toptabs li a {
      display: inline-block;
      color: white;
      text-align: center;
      padding: 14px 16px;
      text-decoration: none;
      transition: 0.3s;
      font-size: 17px;
}

/* Change background color of links on hover */
ul.toptabs li a:hover {background-color: #ddd;}
```

```
#active {
      background-color: #333333;
}

/*Content stuff*/
div.content {
      display: block;
      position: absolute;
      left: 20px;
      top: 55px;
      z-index: -1;
      overflow: auto;
      color: hsl(0, 0%, 88%);
      font-size: 30px;
}

/*h2 content style*/
div.content h2 {
      font-size: 40px;
}

/*h3 content style*/
div.content h3 {
      padding-top: 50px;
      font-size: 35px;
      line-height: 10%;
}

/*paragraph content style*/
div.content textarea {
      background-color: #8c8c8c;
      font-size: 30px;
}

/*radio button content style*/
div.content input[type='radio'] {
      transform: scale(2);
}

/*submit button content style*/
div.content input[type='submit'] {
      width: 100px;
      height: 40px;
      float: right;
}


/*SlideShow styling below*/
```

```
.mySlides {
position: fixed;
top: 8%;
left: 2%;
right: 2%;

  display:none;
-webkit-animation:fade 2s;
-moz-animation:fade 2s;
animation:fade 2s;
}

@-webkit-keyframes fade {
      0% {
            opacity:0.1;
      }
      100% {
            opacity:1;
      }
}

@-moz-keyframes fade {
      0% {
            opacity:0.1;
      }
      100% {
            opacity:1;
      }
}

@keyframes fade {
      0% {
            opacity:0.1;
      }
      100% {
            opacity:1;
      }
}


/*removing "+Google Calendar" button*/
div.subscribe-image{
      display: none;
}
```

2. Message sending

send_Message.php

```
<html>
<head>
      <title>Mali Messenger</title>
      <link rel="stylesheet" type="text/css" href="./CSS/Style.css">
</head>

<body>


      <!--tabs at the top-->

      <ul class="toptabs">
      <li><a href="index.html">Calendar</a></li>
      <li><a href="Ads.html">Advertisements</a></li>
      <li id="active"><a>Mali Messenger</a></li>
      <li><a href="Announcements.html">Announcements</a></li>
      </ul>

      <!--all page content. has left margins for the side tabs-->
      <div class="content">

<?PHP
require "/home/jason/vendor/phpmailer/phpmailer/PHPMailerAutoload.php";
$email="jmmorris2@mail.bradley.edu";
$phone="3097121597@vtext.com";
$mail = new PHPMailer(); // create a new object
$mail->IsSMTP(); // enable SMTP
$mail->SMTPAuth = true; // authentication enabled
$mail->SMTPSecure = 'ssl'; // secure transfer enabled REQUIRED for Gmail
$mail->Host = "smtp.gmail.com";
$mail->Port = 465; // or 587
$mail->IsHTML(true);
$mail->Username = "malismartcalendar@gmail.com";
$mail->Password = "iotcalendar";
$mail->SetFrom("example@gmail.com");
$mail->Body = $_POST['message'];
$messageType=$_POST['messageType'];
if ($messageType == "Urgent"){
$mail->AddAddress($phone);
}
else
```

```
{
$mail->AddAddress($email);
$mail->Subject = "Calendar Message";
}
 if(!$mail->Send()) {
      echo "Mailer Error: " . $mail->ErrorInfo;
 } else {
      echo "Message has been sent";
 }
?>

</div>

      <script src="./JavaScript/idleDirect.js"></script>

</body>
</html>
```

3. Web Page Functionality

SlideFunctionality.js

```
var myIndex = 0;
carousel();
function carousel() {
      var i;
      var x = document.getElementsByClassName("mySlides");
      for (i = 0; i < x.length; i++) {
      x[i].style.display = "none";
      }
      myIndex++;
      if (myIndex > x.length) { myIndex = 1 }
      x[myIndex - 1].style.display = "block";
      setTimeout(carousel, 5000); // Change image every 2 seconds
}
```

activeDirect.js

```
var IDLE_TIMEOUT = 10; //seconds
var _idleSecondsCounter = 0;

document.onclick = function()
{
	_idleSecondsCounter = 0;
};

document.onmousemove = function() {
	_idleSecondsCounter = 0;
};

document.onkeypress = function()
{
	_idleSecondsCounter = 0;
};
window.setInterval(CheckIdleTime, 1000);

function getText()
{
	var textrequest = new XMLHttpRequest();
	textrequest.open("GET", "temporary/message.txt", false);
	textrequest.send(null);
	//window.alert(textrequest.status);

	if (textrequest.readyState === 4 && textrequest.status === 200)
	{
	var type = textrequest.getResponseHeader("Content-Type");
	if (type.indexOf("text") !== 1)
	{
		var textfile = " ";
		textfile = textrequest.responseText;

		if(textfile.includes("b")) window.location.href = "index.html";
		//else window.alert(textfile);
	}
	}
}

function CheckIdleTime()
{
	_idleSecondsCounter++;

	if (_idleSecondsCounter >= IDLE_TIMEOUT)
	{
    getText();
       }
}
```

idleDirect.js

```
var IDLE_TIMEOUT = 90; //seconds
var _idleSecondsCounter = 0;
document.onclick = function() {
      _idleSecondsCounter = 0;
};
document.onmousemove = function() {
      _idleSecondsCounter = 0;
};
document.onkeypress = function() {
      _idleSecondsCounter = 0;
};
window.setInterval(CheckIdleTime, 1000);

function CheckIdleTime() {
      _idleSecondsCounter++;

      if (_idleSecondsCounter >= IDLE_TIMEOUT) {
      window.location.replace("Ads.html");
      }
}
```

4. Advertisement Image Grabbing Functionality

images_Grab.py

```
# grab all picture file names in images folder
# files_grabbed = array of image file names
import glob, os
types = ('*.jpg', '*.png', '*.gif')
files_grabbed = []
for files in types:
      files_grabbed.extend(glob.glob(files))

# get all text in the Ads.html file and put in a string
# called htmlString
os.chdir("..")
with open('Ads.html', 'r') as myfile:
      htmlString=myfile.read()

# split the string in between the '<!--images-->'
picString = htmlString.split("<!--images-->")
```

```
# what goes before and after file names in Ads.html
preFName = '<img class="mySlides" src="./images/'
postFName = '" style="width:96%">'

# go through and chang newText to have the new files
newText = ''
for x in range(0, len(files_grabbed)):
      newText += '\n\t\t\t' + preFName + files_grabbed[x] + postFName
newText += '\n\t\t'

# replace the old text with the new for the new images
htmlString = htmlString.replace(picString[1], newText)

# write the htmlString into the Ads.html file
writeFile = open("Ads.html", "w")
writeFile.write(htmlString)
```

5. Background System Functionality

init.py

```
import os, sys, time

os.system("python /var/www/Smart-Calendar/Background/startup.py --monitor-off")
#dont let anyone see desktop
os.system("python /var/www/Smart-Calendar/Background/startup.py --start")
#startup services

time.sleep(60) #wait in seconds

os.system("python /var/www/Smart-Calendar/Background/startup.py --monitor-on")
#browser should have loaded by now
```

pins.py

```
import os, time, pwd, time

do_monitor = 1;

def timestamp(first, llast, rlast, door_on, cal_on): #timestamp generator
    stamp = 'Detected activity from '

    if(first == 0): #entry point
       stamp = stamp + 'left to '
    elif(first == 1):
       stamp = stamp + 'right to '
    else:
       stamp = stamp + 'IDK to '

    if(llast == 1 and rlast == 0): #exit point
       stamp = stamp + 'left at '
    elif(llast == 0 and rlast == 1):
       stamp = stamp + 'right at '
    else:
       stamp = stamp + 'IDK at '

    stamp = stamp + time.asctime() + '.' #time

    if(door_on == 1 and cal_on == 1): #extra info
       stamp = stamp + ' They stopped at the door and calendar.'
    elif(door_on == 1):
       stamp = stamp + ' They stopped at the door.'
    elif(cal_on == 1):
       stamp = stamp + ' They stopped at the calendar.'

    stamp = stamp + '\n'
    file = open('/var/www/Smart-Calendar/HTML/temporary/stamp.txt', 'a') #write
to stamp
    file.write(stamp)
    file.close()

state = 0 #0 = idle state, 1 = active state
timeout = 0 #counts seconds of inactivity
caltimer = 0 #counts seconds of activity for left sensor above calendar
doortimer = 0 #counts seconds of activity for right sensor above door
first = 0 #-1 = Invalid, 0 = Left, 1 = Right,
llast = 0 #boolean, true = left sensor was active last check
rlast = 0 #boolean, true = right sensor was active last check
calstop = 0 #boolean, true = user stopped at calendar
doorstop = 0 #boolean, true = user stopped at door
```

```
cal_is_on = 1 #boolean, true = monitor is on
cal_is_active = 1 #boolean, false = idly displaying ads, true = active

while(1):
    file = open("/sys/class/gpio/gpio17/value", "r") #read pins
    L = int(file.read())
    file.close()
    file = open("/sys/class/gpio/gpio27/value", "r")
    R = int(file.read())
    file.close()

    if state == 0: #idle
       #print 'idle'
       if(L == 1 and R == 1): #left and right activity
             state = 1
             first = -1 #did not expect both sensors to turn on at same time
             llast = 1
             rlast = 1
             caltimer = 1
             doortimer = 1
             timeout = 0
       elif(L == 1): #left activity
             state = 1
             first = 0
             llast = 1
             rlast = 0
             caltimer = 1
             doortimer = 0
             timeout = 0
       elif(R == 1): #right activity
             state = 1
             first = 1
             llast = 0
             rlast = 1
             caltimer = 0
             doortimer = 1
             timeout = 0
       else: #no activity
             timeout = timeout + 1
    elif state == 1: #activity
       #print 'activity'
       L_visited = 1
       if(L == 0 and R == 0): #no activity
             state = 0
             timestamp(first, llast, rlast, doorstop, calstop)
             calstop = 0
             doorstop = 0
```

```
        if L == 1: #left activity
              caltimer = caltimer + 1
              llast = 1
        else:
              caltimer = 0
              llast = 0

        if caltimer >= 4:
              caltimer = 4
              calstop = 1

        if R == 1: #right activity
              doortimer = doortimer + 1
              rlast = 1
        else:
              doortimer = 0
              rlast = 0

        if doortimer >= 6:
              doortimer = 6
              doorstop = 1

    if(do_monitor == 1):
        if(timeout < 900 and cal_is_on == 0): #turn monitor on if activity
              os.system("python startup.py --monitor-on &")
              #print 'turn on'
              cal_is_on = 1
        elif(timeout >= 900): #turn monitor off if no activity
              timeout = 900
              if(cal_is_on == 1):
                    os.system("python startup.py --monitor-off &")
                    #print 'turn off'
                    cal_is_on = 0

    if(timeout < 90 and cal_is_active == 0): #if calendar becomes active
        file = open('var/www/Smart-Calendar/HTML/temporary/message.txt', 'w')
        file.write('b')
        file.close()
        cal_is_active = 1
    elif(timeout >= 90 and cal_is_active == 1): #if calendar becomes idle
        file = open('var/www/Smart-Calendar/HTML/temporary/message.txt', 'w')
        file.write('i')
        file.close()
        cal_is_active = 0

time.sleep(1)
```

startup.py

```
import os, sys, serial, time

option = str(sys.argv[1:2])

if(option == '[\'--start\']'):
    #create communication file
    if(not
os.path.isfile('/var/www/Smart-Calendar/HTML/temporary/message.txt')):
       file = open('/var/www/Smart-Calendar/HTML/temporary/message.txt', 'w+')
       file.close()
    #create timestamp file
    if(not os.path.isfile('/var/www/Smart-Calendar/HTML/temporary/stamp.txt')):
       file = open('/var/www/Smart-Calendar/HTML/temporary/stamp.txt', 'w+')
       file.write('File Created ' + time.asctime() + '\n')
       file.close()
    #turn on pins, set to input
    os.system("echo 17 > /sys/class/gpio/export")
    os.system("echo in > /sys/class/gpio/gpio17/direction")
    os.system("echo 27 > /sys/class/gpio/export")
    os.system("echo in > /sys/class/gpio/gpio27/direction")
    os.system("python /var/www/Smart-Calendar/Background/pins.py &")

elif(option == '[\'--stop\']'):
    os.system("kill $(ps -eF | grep pins.py | grep -v grep | awk '{print
$2}')") #end pins.py
    #turn off pins
    os.system("echo 17 > /sys/class/gpio/unexport")
    os.system("echo 27 > /sys/class/gpio/unexport")
    #format timestamp file
    timevar = time.asctime()
    file = open('/var/www/Smart-Calendar/HTML/temporary/stamp.txt', 'a+')
    file.write('File Closed ' + timevar + '\n')
    file.close()
    timevar = timevar[4:7] + '_' + timevar[8:10] + '_' + timevar[20:] +
'_stamp.txt'
    os.rename("/var/www/Smart-Calendar/HTML/temporary/stamp.txt",
"/var/www/Smart-Calendar/HTML/temporary/" + timevar)
    #email file

elif(option == '[\'--monitor-on\']'):
    os.system("tvservice -p; chvt 6; chvt 7") #turn on HDMI port
    #close power to monitor through serial
    ser = serial.Serial('/dev/ttyUSB0', 9600, timeout=1)
    code = 'A00101A2'
    ser.write(code.decode('HEX'))
```

```
elif(option == '[\'--monitor-off\']'):
    os.system("tvservice -p; tvservice -o") #turn off HDMI port
    #open power to monitor through serial
    ser = serial.Serial('/dev/ttyUSB0', 9600, timeout=1)
    code = 'A00100A1'
ser.write(code.decode('HEX'))
```