6-15-2017

SmartMirror: A Glance into the Future

Jason Chen
Santa Clara University, jchen@scu.edu

Matthew Koken
Santa Clara University, mkoken@scu.edu

SANTA CLARA UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

Date: June 14, 2017

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Jason Chen
Matthew Koken

ENTITLED

SmartMirror: A Glance into the Future

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

Thesis Advisor

Department Chair

SmartMirror: A Glance into the Future

by

Jason Chen
Matthew Koken

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Science and Engineering
School of Engineering
Santa Clara University

Santa Clara, California
June 15, 2017

SmartMirror: A Glance into the Future

Jason Chen
Matthew Koken

Department of Computer Engineering
Santa Clara University
June 15, 2017

ABSTRACT

In todays society, information is available to us at a glance through our phones, our laptops, our desktops, and more. But an extra level of interaction is required in order to access the information. As technology grows, technology should grow further and further away from the traditional style of interaction with devices. In the past, information was relayed through paper, then through computers, and in todays day and age, through our phones and multiple other mediums. Technology should become more integrated into our lives - more seamless and more invisible. We hope to push the envelope further, into the future. We propose a new simple way of connecting with your morning newspaper. We present our idea, the SmartMirror, information at a glance. Our system aims to deliver your information quickly and comfortably, with a new modern aesthetic. While modern appliances require input through modules such as keyboards or touch screen, we hope to follow a model that can function purely on voice and gesture. We seek to deliver your information during your morning routine and throughout the day, when taking out your phone is not always possible. This will cater to a larger audience base, as the average consumer nowadays hopes to accomplish tasks with minimal active interaction with their adopted technology. This idea has many future applications, such as integration with new virtual or augmented reality devices, or simplifying consumer personal media sources.

Table of Contents

1 Introduction **1**
- 1.1 The Problem . . . 1
- 1.2 Other Solutions . . . 1
- 1.3 Our Solution . . . 2

2 Functional Requirements **3**
- 2.1 Functional Requirements . . . 3
- 2.2 Non-Functional Requirements . . . 4
- 2.3 Design Constraints . . . 4
- 2.4 Requirements Justification . . . 4

3 Use Cases **5**
- 3.1 Use Case 1: . . . 5
- 3.2 Use Case 2: . . . 6

4 Activity Diagram **7**

5 Current Model **8**
- 5.1 Hardware Design . . . 9
- 5.2 Software Design . . . 9
- 5.3 Application Structure . . . 9
- 5.4 Widget Class Implementation . . . 10
- 5.5 Grid Layout . . . 10

6 Architectural Diagram **12**

7 Technologies Used **13**

8 Costs **14**

9 Design Rationale **15**

10 Testing Plan **16**
- 10.1 Alpha Testing . . . 16
- 10.2 Beta Testing . . . 16
- 10.3 Test Results . . . 17

11 Risk Analysis **18**

12 Development Process **19**

13 User Manual **21**
13.1 Initial Setup . . . 21
13.2 Stopping the Application . . . 21
13.3 Alexa Voice Service . . . 21
13.4 Creating a Widget . . . 21

14 Lessons Learned **23**
14.1 Scoping . . . 23

15 Suggested Changes **24**
15.1 Integration with Amazon Alexa . . . 24
15.2 Further Simplification . . . 24
15.3 Quality of Life . . . 24
15.4 Alternative Inputs . . . 24

16 Conclusion **25**
.0.1 Constructor . . . 28
.0.2 Item Index . . . 28
.0.3 Methods . . . 28
.0.4 Item Index . . . 29
.0.5 Functions . . . 29

List of Figures

3.1 Use Case for Project 5

4.1 Activity Diagram for Consumer (left) and Developer (right) 7

5.1 Finished Hardware Prototype 8
5.2 Outer Structure 9
5.3 Inside 9
5.4 Application Structure 10
5.5 Widget Class Structure 10
5.6 Conceptual Model 11

6.1 Architectural Diagram 12

11.1 Risk Analysis for Project 18

12.1 Timeline for Project 20

Chapter 1

Introduction

1.1 The Problem

In our rapidly developing world, information is always right at your fingertips - on your phone, on your computer, maybe even on your watch. Staying connected with new information is both important for entertainment and daily life. With such a variety of options, there is difficulty in following all of your data streams. Often, during your day, you may end up in a position where it is inconvenient, or even impossible, to take out your phone or computer and check the newest update. You cannot commit to a slower interaction. You need a display to glance at, with the information you need ready to go. However, aesthetics are just as important as displaying information. Keeping an extra computer in your bathroom or hall would be inconvenient, and would not fit well with the look of a modern room. A sleek, simple display, easy for an average consumer, is a necessity in todays world.

1.2 Other Solutions

There are several products in the market that attempt to be your attractive hub of daily information. The Amazon Echo and the upcoming Google Home present themselves as a small speaker that can relay information through sound. You can request news or music, fulfilling your need to obtain media content in a hands-free manner. However, not all data is suitable for conveyance by voice. Both designs lack the key ability to convey information visually. Asking for the morning traffic can give you a time estimate, but it barely comes close to a detailed map with your route information. Having the news read to you is convenient, but many prefer reading the news at their own pace. A smart display would be a product that would be able to answer all of these concerns, while staying smoothly modern. The Nest thermostat has a small display for information. However, it is not intended for interaction. The interface can be clunky, and not something an average consumer would interact with on a day-to-day basis. The recently Kickstarted Perseus hides a screen and computer behind a two-way mirror. This allows users to interact with the mirrors applications via touch

screen, voice, and camera controls. Perseus, however, is a finished product, and does not allow user hardware customization. It claims to have an available API and third party applications, but currently, there is little to no information or documentation on this matter. With months to go before its delivery, the success of this product remains uncertain. A few Do-It-Yourself (DIY) alternatives are also currently available. Both the MirrorMirror and Smart-Mirror projects provide an application to display information on a monitor behind a mirror. However, these require legwork on the user end, as not all users are willing to manually construct the project from scratch. Manual configuration and tinkering with modules can be a tedious and difficult process.

1.3 Our Solution

Our solution is an open platform for discrete display development. We offer an aesthetically pleasing mirror, with a hidden smart display underneath. With a generic display, the mirror can be built to any size so the information can be both in your face while showing you your face. Our product differs from the competition with an easy-to-use interface that is both simple for the average user and open for the advanced developer. A sleek display gives all levels of users a modern hub of technology for their personal daily interaction, one which both displays visually all the information you could need or want, and operates with a simple interaction that you could fit into your daily routine. By creating a platform open to modification, developers will also be add new functionality at their own pace. This will allow our display to be a tailorable and adaptable platform. A web application provides the interface that the user sees and interacts with. An online configurator will relieve the frustration and difficulty of personalizing your information, as well as allow streamlined development of new modules. Powered by a small computer, the smart mirror will have great potential for expansion by developers. As an open platform, consumers and developers will be able to easily build, adapt, and hack their smart mirror to fit their own needs. Our product will be a step in the future of IoT, connecting your daily mirror to your tech-savvy world.

Chapter 2

Functional Requirements

The following requirements define the goals of the project outlined in the introduction. The functional requirements define features that must be done for the project to be considered a success, while the non-functional requirements define how the functional requirements are achieved. Requirements are categorized into critical, recommended, and suggested. Critical requirements are absolutely necessary, recommended are highly desirable, and suggested requirements are not necessary but would be very nice to add.

Design constraints are criteria that the solution must adhere to. The constraints are set by the client and are non-negotiable.

2.1 Functional Requirements

Critical:

1. Must be able to display information on screen
2. Must be controlled by something without requiring direct input
3. Must be connected to the web to receive incoming data
4. Must be module-based and contain sample default modules
5. System defaults in low power sleep mode
6. Must be able to scale to multiple screen sizes

Recommended:

1. Controlled by alternative input methods
2. Live RSS feed displays
3. Integrate more advanced web modules, perhaps a browser
4. Sleeps when certain time has passed

Suggested:

1. Allow users to integrate their own web modules

2.2 Non-Functional Requirements

Critical:

1. A simpler user interface than a computer
2. System has good performance for users
3. System maintains good reliability for users
4. Display disappears and becomes a mirror

Recommended:

1. A friendly user interface that works by selecting modules
2. System remembers user name and can reply to user by name

Suggested:

1. Ability to augment a reflection

2.3 Design Constraints

Constraint

1. Solution must be standards compliant for popularly supported interfaces
2. Solution must be open-source
3. System must be accessible for hobbyists
4. System must be scale-able, extensible, and extendable

2.4 Requirements Justification

Due to the limited time and manpower available for this project, we were not able to implement all requirements. Each requirement will be fulfilled based on level of priority in order to create a minimum deliverable system. The minimum system for delivery will serve as a baseline for further discrete display development. As an open-source system, SmartMirror will be a platform for developers, hackers, and makers. Here, developers will be the main target as they will be able to add additional features to the extendable and extensible system.

Adding recommended features will improve usability and functionality for hackers and makers. A more user-friendly system will allow greater access to the platform for the public. Advanced features such as alternative input methods are extremely important for long-term usability, but were much too difficult to implement and would have required too much additional research.

Chapter 3

Use Cases

Below are a few use cases for our platform. Use cases are examples of how our system is to be used. The main actors of our system are the consumer, who plans to interact with widgets, and the developer, who plans to create new widgets. Figure 3.1 displays our use case diagram for our system.

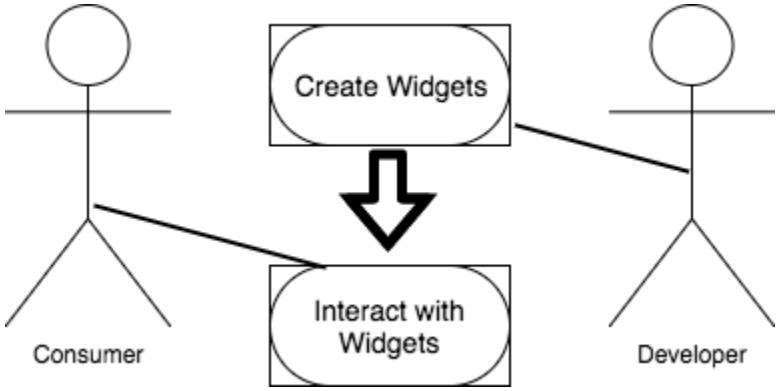

Figure 3.1: Use Case for Project

3.1 Use Case 1:

Name: Interact with Widgets

Goal: Consumer is able to access provided interface modules (i.e. app, widget)

Actors: Consumer

Pre-conditions:

- System is displaying GUI
- Consumer is present in front of the screen

Post-conditions:

- Selected application is opened

Steps:

1. User selects module
2. Selected Application is displayed on the screen for user interaction

Exceptions: N/A

3.2 Use Case 2:

Name: Create and Add Widgets

Goal: Developer is able to create and add application to system

Actors: Developer

Pre-conditions:

- System is currently functioning

Post-conditions:

- Developer's Application running on widget

Steps:

1. Developer writes application functionality (in JavaScript)
2. Developer creates custom widget class
3. Developer adds functionality to custom widget class
4. Developer adds custom widget class to app.js

Exceptions: N/A

Chapter 4

Activity Diagram

The general work-flow of the planner can be graphically represented in an activity diagram. Figure 2 (Pg. 12) shows how an average user will use the SmartMirror, and the step by step process they will go through as they progress through the site. The diagram shows the work flow for all average users. The system will load the user GUI. The user is then able to interact with selected modules, or open new modules.

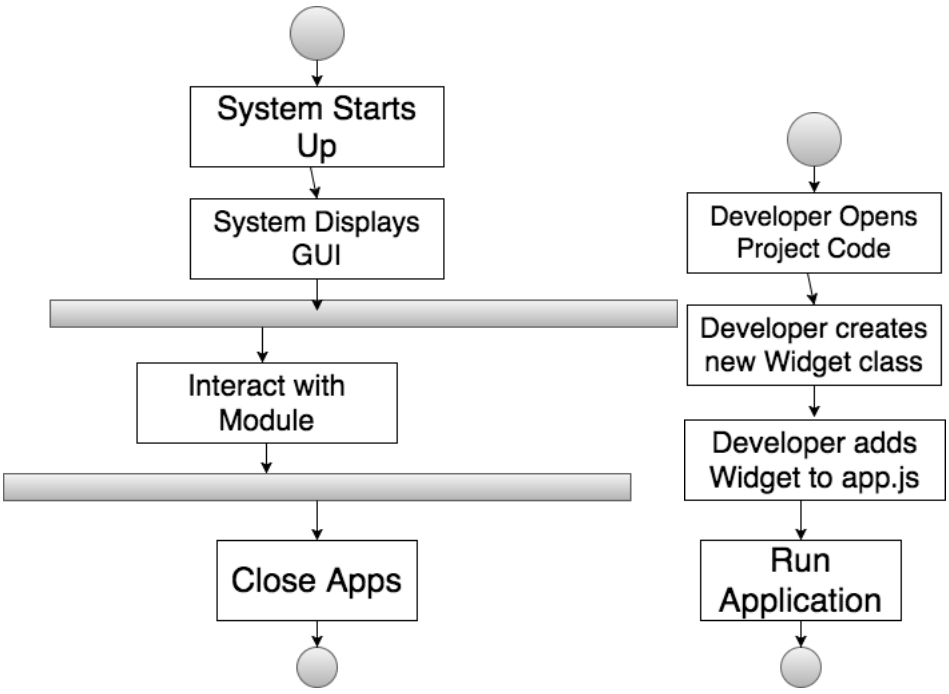

Figure 4.1: Activity Diagram for Consumer (left) and Developer (right)

Chapter 5

Current Model

The overall system has two main components - a hardware component and a software component. The hardware model is a 2 way mirror covering a LCD screen, both of which approximately measure 23” diagonally. The LCD screen is connected to a Raspberry Pi 3, although other computing devices may be suitable. The Raspberry Pi 3 has a microphone connected in order to collect audible input. All components are encapsulated within a wooden frame. On the software side, we host a local server that generates web content to be displayed on the screen.

Figure 5.1: Finished Hardware Prototype

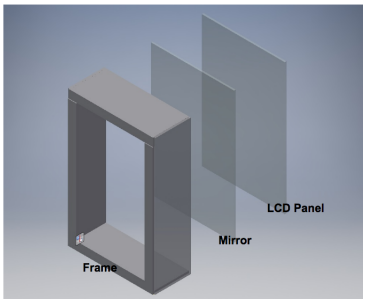

Figure 5.2: Outer Structure

5.1 Hardware Design

On the outside, the hardware is encapsulated within a wooden frame. On the front, a two-way mirror is placed in front of a LCD monitor. This way, the system can act as a mirror when not currently in use, while the LCD projects through the mirror when in use. The wooden frame has a bezel on the front which the mirror and LCD panel are pressed against. Cutouts for dowels are added in line with the back of the LCD panel to keep the components snug against the bezel.

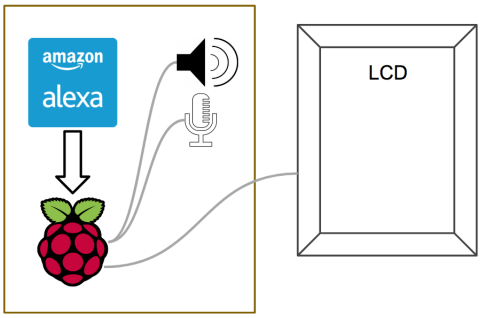

Figure 5.3: Inside

Behind the LCD panel is the heart of the product, the Raspberry Pi, which is connected to the LCD for visual display. We also have a microphone and speaker attached for audio input and output. The Raspberry Pi runs our software and also allows us to connect to the Internet for web services, such as Amazon's Alexa Voice Service.

5.2 Software Design

The goal of our project is to create an open platform for development, and all software components must fit to that goal. The software is designed to run on many platforms and fit many display types.

5.3 Application Structure

The application makes use of two main sections - a server and a local client. The server, built using Node.js provides the back end for the project. The hosts a simple web page and opens a web browser directed to that page. The HTML content of the page contains the grid layout for widgets to be placed in. Tied to this web page is the local client, built using JavaScript. The local client communicates with the server and handles all display activities. The client loads and refreshes widgets as needed and provides the primary interface for users. Using IPC, the Node.js server is able to send commands to the client in order to extend functionality. This can be used to communicate with other IoT devices and display data such as current temperature in a house as reported by a smart thermostat. The same could be done in reverse, where information is sent to an IoT device or Internet service.

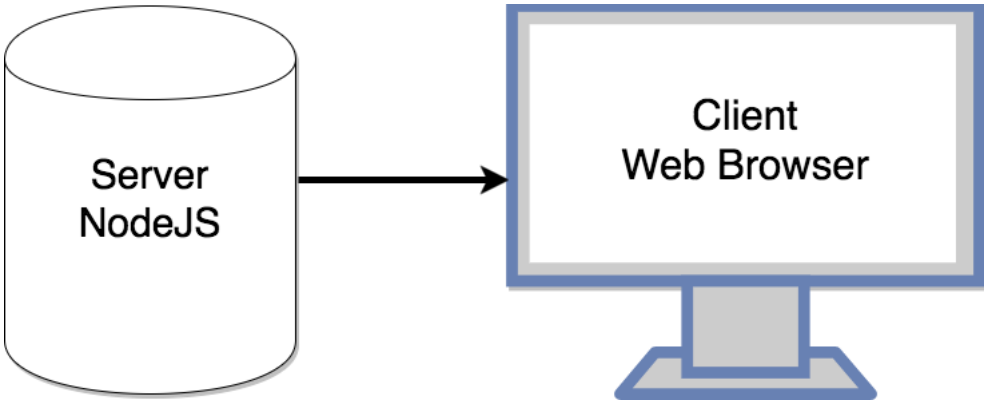

Figure 5.4: Application Structure

5.4 Widget Class Implementation

The Widget class is the basis of information display for the SmartMirror. A Widget is a self-contained module that injects HTML content into a specified location. A primary loop function is called periodically based upon a specified refresh rate. This is handled by the client main. The loop function is modeled to function similarly to the loop used in Arduino development. This provides a familiar workflow for hobbyists and eases the learning curve.

All widgets derive from the default Widget class, and are imported and instantiated within the client main. Each widget class's logic is defined by a "loop" function, which contains the main functionality of each widget. This will be called by the widget handler in app.js, to be run in the specific location that is given to the widget. Each widget will also contain a main constructor, which will contain vital information for the running of the widget, such as the name and the rate the information called needs to be refreshed. Extending the widget class to create a custom module allows for user customization to fit many needs. Additional functions may be added by the user as needed without affecting other widgets. However, these functions will are internal to the widget and will not currently be used by the client main without modification.

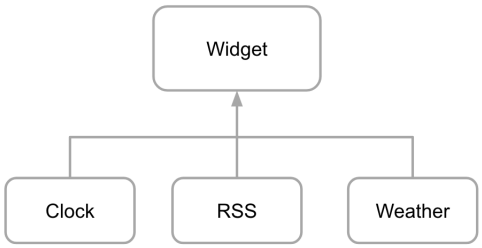

Figure 5.5: Widget Class Structure

5.5 Grid Layout

In order to best fit several widgets on the screen, we chose a simple grid layout to house separate pieces of information. Using CSS and HTML divs, widgets are given a liquid space to fill. This ensures that widgets will resize as needed to keep content visible without running over another widget. Each div tag contains a unique ID that specifies its location. This is given to a widget on instantiation for the widget to add HTML content to.

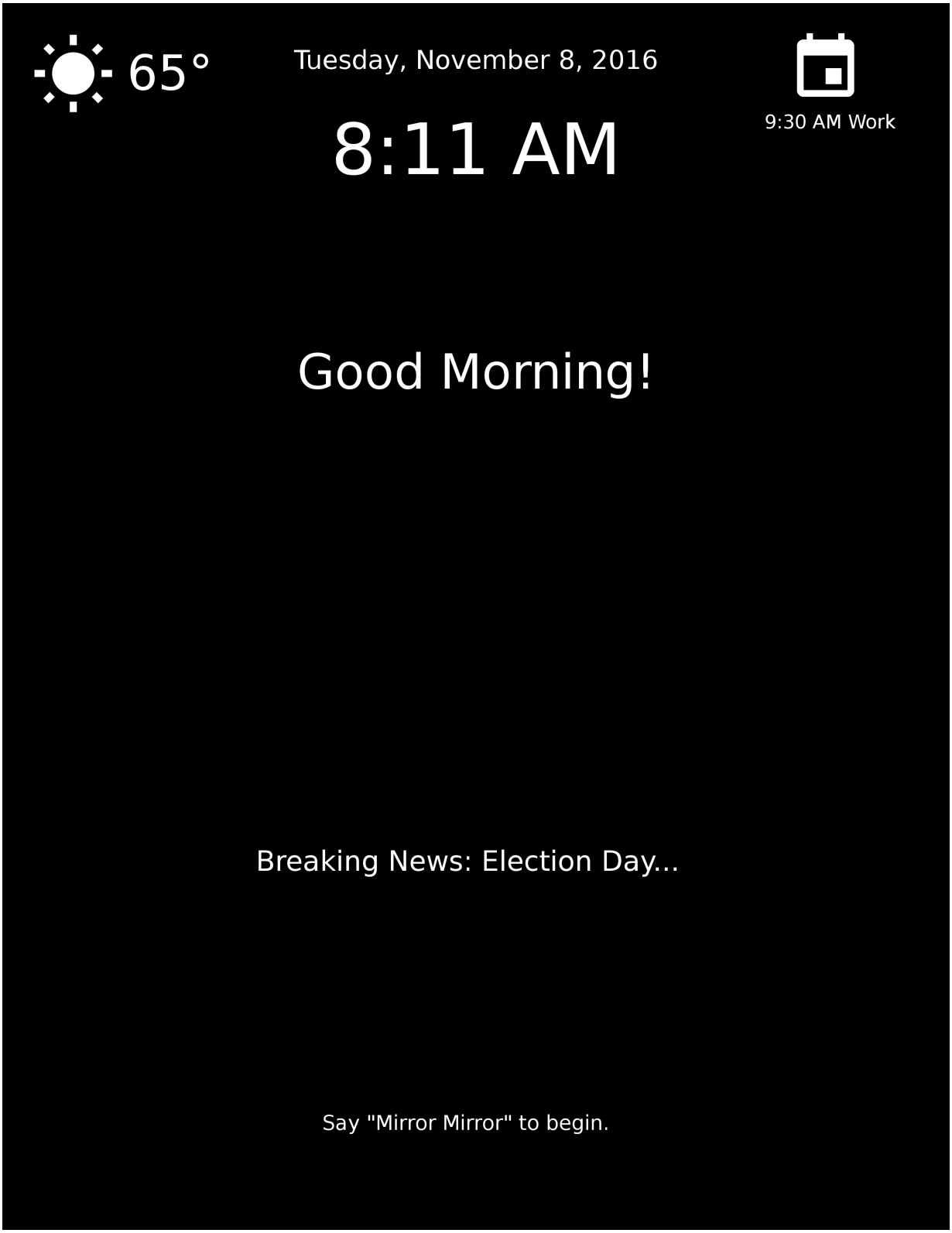

Figure 5.6: Conceptual Model

Chapter 6

Architectural Diagram

The design of the system is a combination of a layered architecture and a client-server architecture. The user interacts primarily with the GUI that is built upon the OS we use for the development for our system. When they attempt to make requests to edit their settings directly, they will be making a call as an application to the underlying OS, which will change the system accordingly. The system itself communicates with remote clients via the Internet. When receiving information for interface modules, the system acts as a client to the web services. In interacting with other smart devices, the system will act as a server for information. Figure 3 highlights an example of a layered architecture:

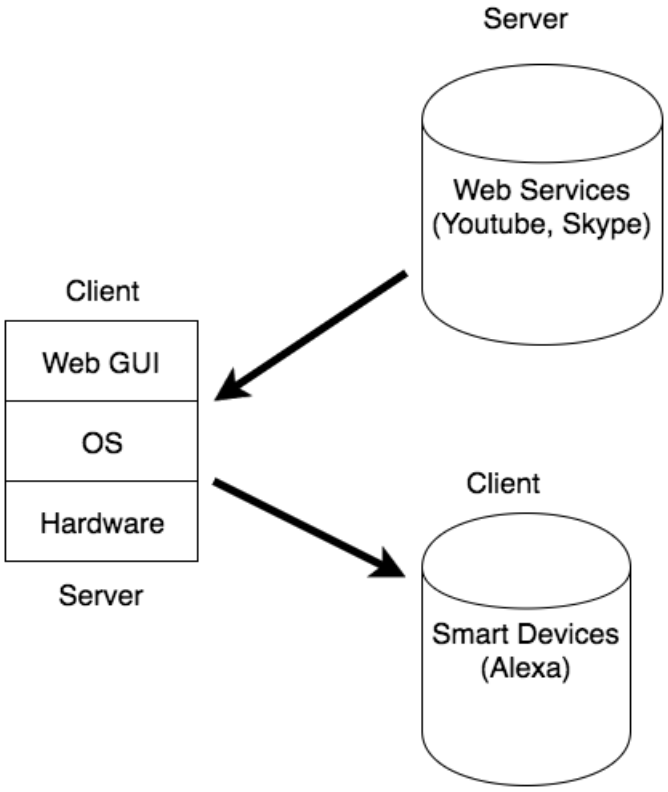

Figure 6.1: Architectural Diagram

Chapter 7

Technologies Used

Below are a list of technologies we plan to use for our project. Being a web-based application, we used a variety of web tools:

- HTML
- CSS
- JavaScript
- NodeJS
- Electron
- GitHub
- Travis CI
- Raspberry Pi
- Amazon Alexa Voice Service

The primary hardware component for our system is the Raspberry Pi. In particular, we use the Raspberry Pi 3 as it maintains the same price point but offers additional processing power, more RAM, and offers onboard bluetooth and WiFi for connectivity. The Raspberry Pi was selected for its ease of use and availability to the hobbyist community. The Raspberry Pi is capable of running several flavors of Linux, all of which should be capable of running our software platform.

In order to display content, NodeJS and Electron were chosen. Electron building multi-platform applications over a NodeJS base. NodeJS runs as the main server and hosts web content using JavaScript. The server also has extensions to interact with the Raspberry Pi's hardware GPIO so future users may add additional IoT connectivity. Using Electron, a minimal web browser is launched to display the content hosted by the server. Together, the application is able to be used on a wide variety of computers and is highly scale-able - exactly what was needed to meet the requirements.

Electron allows for the use of standard web technologies to implement a front end. This allowed the use of HTML, CSS, and JavaScript to implement the entirety of the system. Using standard and familiar technologies was important for speed of implementation as well as future users who may wish to modify the system. Using unfamiliar technologies and programming languages would increase the barrier for entry and deter potential users who may not have as much of a technological background. We also used an Electron package called 'electron-boilerplate.' This package provided a faster start to application development and allows for more integrated testing as well as the potential to build release executables for multiple systems.

In order to host the source code and other components of the project, a version control system was necessary. GitHub was chosen for its familiarity and compatibility with other technologies. GitHub allowed for multiple developers to work on the project at once and keep track of version history. Other users can access the project from GitHub and obtain any version of the project.

Because of GitHub's compatability with other software, we chose to use Travis CI to run tests on our system. On any change in the GitHub repository, Travis CI runs all tests we have written and reports the results. This helps track changes in the system's functionality and catch bugs as they are introduced.

Chapter 8

Costs

This project is self-funded. Therefore, we did not request funding for this project, as the up front costs are relatively inexpensive. However, the specific materials that are used will be listed below:

Raspberry Pi 3 Model B Starter Kit	$49.99
One-way Mirror	$40.00
Generic Monitor	$60.00
Wood for Frame	$20.00
Total	$170.00

Table 8.1: Costs Chart.

In order for someone to replicate this project, they would be required to pay a similar up-front cost; however, many hobbyists may already own some of the necessary materials. Choosing commonly used hobbyist pieces makes the project most accessible to the widest audience and also helps decrease initial investment costs.

Chapter 9

Design Rationale

We decided to make our model open source, due to the fact that it allows a plethora of users to enjoy the product on different levels: the average user will not concern themselves with a difficult world of development, and the talented programmer will be able to modify the product to suit their needs. We feel that this supports and builds a healthy community, as all other marketed renditions on the market today do not allow for customization. We feel that allowing our product to be extensible not only allows for a much easier integration, should a group wish to continue developing this product, but also allow for users to be able to create some truly unique modifications.

We decided to go with our languages, due to the fact that our design is a web module. Users will interact through the web module, and be able to access their data through said web module. This allows for a much easier integration into the web, and displaying constantly updated data will be much easier to retrieve through web modules.

Chapter 10

Testing Plan

In order to verify the functionality of the system, several rounds of testing is necessary. Testing at each stage of development ensures that errors are caught early and addressed before they cause delays in development and delivery.

10.1 Alpha Testing

The first stage of testing for the system is Alpha Testing. This includes testing with knowledge of the internal functionality (white box) and without knowledge of internal functionality (black box). Both styles of testing are carried out continuously - at all stages of implementation and beyond. Testing is carried out across multiple platforms to that system will be usable for a wider audience.

For White Box Testing, the primary testing method is provided by Travis CI, a continuous integration service with access to the GitHub repository. As features are developed, tests are written to verify the functionality of each module of the system and the system as a whole. Upon completion of a feature or part of the system, each developer will write a test using the test platform provided by Selenium Webdriver. Upon any changes in the GitHub repository, all written tests are run to verify the system. If any test fails, the build is marked as a failure and developers are notified to address the failing test cases. A test on a module will attempt to pass incorrect values or correct values and verify that the expected functionality is error-free.

In order to carry out Black Box Testing, Travis CI is also used. Selenium Webdriver was used to develop tests that verify module functionality without code knowledge. Tests written for this style provide inputs such as click actions on the web application and verify that the expected outputs are received. These tests verify results by checking for HTML tags and id's that can be used to programatically determine the functionality of a system. In essence, they verify that the application displays the correct data without visually inspecting the data. However, visual inspection is still useful and necessary in smaller quantities. After a feature is submitted, a developer who did not work on the feature tests the system, ensuring that the system behaves as expected from an outside standpoint. Any discrepancies are noted as an issue using GitHub's tracking system to make sure that the proper behavior is achieved.

10.2 Beta Testing

The second stage of testing is Beta Testing. After the system is operational for use, Beta Testing will be important. This includes inviting volunteers to work with our system and provide feedback for improvements to the design and user interface. This is an important step in order not only to determine design, aesthetics, and functionality, but also to root out bugs with different requests from different people. We will then accommodate user preference and client requirements, and then perhaps move on to adding extra small functionalities. As developers, we have a certain view of how our system should behave and be used. However, not all users will have the same perspective. By utilizing the outside perspective, we modify the system to be more suitable for end users.

Outside feedback was useful for catching bugs that we as developers saw as features. It also verified and reinforced our concepts of how particular systems should work. The most important feedback we received was concerning the implementation of our module system for developers. It was referred to as having a similar development style as the Arduino platform. The Arduino system is popular within the hobbyist community so having familiarity for users will reduce the learning curve and increase ease of use and development.

10.3 Test Results

By testing throughout the development process, issues with development stopping bugs were avoided. Several errors that could potentially prevent another developer from continuing work were caught immediately by the test net provided by Travis CI. Feedback from other users was also able to guide the further development and improvement of the system. Test results from Travis CI are shown on the GitHub repository so users can know if the latest codebase is safe for use. Travis CI also keeps track of all builds - providing a comprehensive test history to help identify problem areas that may be in need of special attention and improvement.

Chapter 11

Risk Analysis

For our risk analysis table, as seen in Figure 4, it is broken down into six different categories.

- Name of Risk
- Consequences
- Probability
- Severity
- Impact
- Mitigation Strategies

Each risk has consequences for what happens if the risk actualizes. It also carries the probability of occurring, along with the severity of the risk. Probability and severity factor into the impact the risk has on the project. Finally, the table outlines mitigation strategies our team has prepared to help lessen either the probability or severity of the risk. This helps lessen the impact of the risk should it still happen.

A large risk to our project is our team members becoming ill, impacting our schedule for our project. This relates to our other major risk that we will not be able to finish on time. It is still possible that the final deliverable will not be ready by the due date, as the system might still have bugs that critically affect performance.

Most risks, however, were not faced. The only risk that we ended up facing was the failure to meet the deadline, which we solved by cutting several features from the end product, such as the ability for alternative input. While these features would have been interesting to have in the project, they were deemed unnecessary, and cutting them allowed us to be able to deliver a finished project by the deadline.

Risk	Consequences	Probability	Severity	Impact	Mitigation Strategy
Bugs	Project does not run; Project not finished	.8	10	8.0	-Rigorously test project -Develop code in steps -Leave comments for team members to troubleshoot
Failure to satisfy client	Project is unusable in desired setting	.7	10	7.0	-Regularly check-in with client -Seek outside feedback
Sickness	Inability to work, push deadlines back on project	.8	7	5.6	-Eat Healthy -Regular sleep schedule -Follow good hygiene
Failure to meet deadline	Project is not finished on time	.5	10	5.0	-Follow development time-line -Do not put off tasks
Lack of Language Knowledge	Progress is slowed, project not finished on time	.5	7	3.5	-Become familiar with languages -Plan which languages will be needed

Figure 11.1: Risk Analysis for Project

Chapter 12

Development Process

As shown in Figure 6, our group prepared a schedule for meeting our project goal and deadlines. Each team member is assigned tasks to help ensure project success. The legend above identifies which team member has which task, along with group tasks, and deadlines. While most of the group is working on tasks together, certain group members have been appointed leads on different areas, such as customer liaison, implementation lead, and testing lead. The timeline has been broken into four different sections: Requirements, Design, Implementation, and Testing.

For Requirements, this is divided into two parts. The first part is the initial requirements gathering from the client, during weeks 8 and 9. The requirements supplied by the customer help formulate our design idea. The second part, from weeks 8 through 7, is a period of customer interview and feedback on our product, to help mitigate the risk of failing to meet customer expectations.

For Design, this phase is conducted during weeks 8 and 9. Using the information from the Requirements phase, our group will develop a design for the online interface that users will experience.

For Implementation, this phase is conducted during weeks 8 through 6. Building off on our design, our group built the code to program the web modules, then do the research to develop voice recognition and motion recognition.

For Testing, this phase is conducted during weeks 1 through 9. The group will start with basic cases and functionality testing, before moving on to our alpha and beta testing. The feedback from the beta testing will be used to fix any design issues that may not have been prevalent during design and implementation.

In practice, we met around once a week, mostly to discuss the scoping of the project, as well as work on the actual code itself. In the end, while the timeline was optimistic, more time than expected was spent on the actual coding, and thus, the implementation portion ran into the last few weeks, overlapping with the testing. Thus, we tested at a similar time as when we were implementing portions of the code. Matthew developed most of the structure of the project, as well handling the testing with Travis CI. Jason handled some implementation, as well as some implementation of test widgets to run with the project.

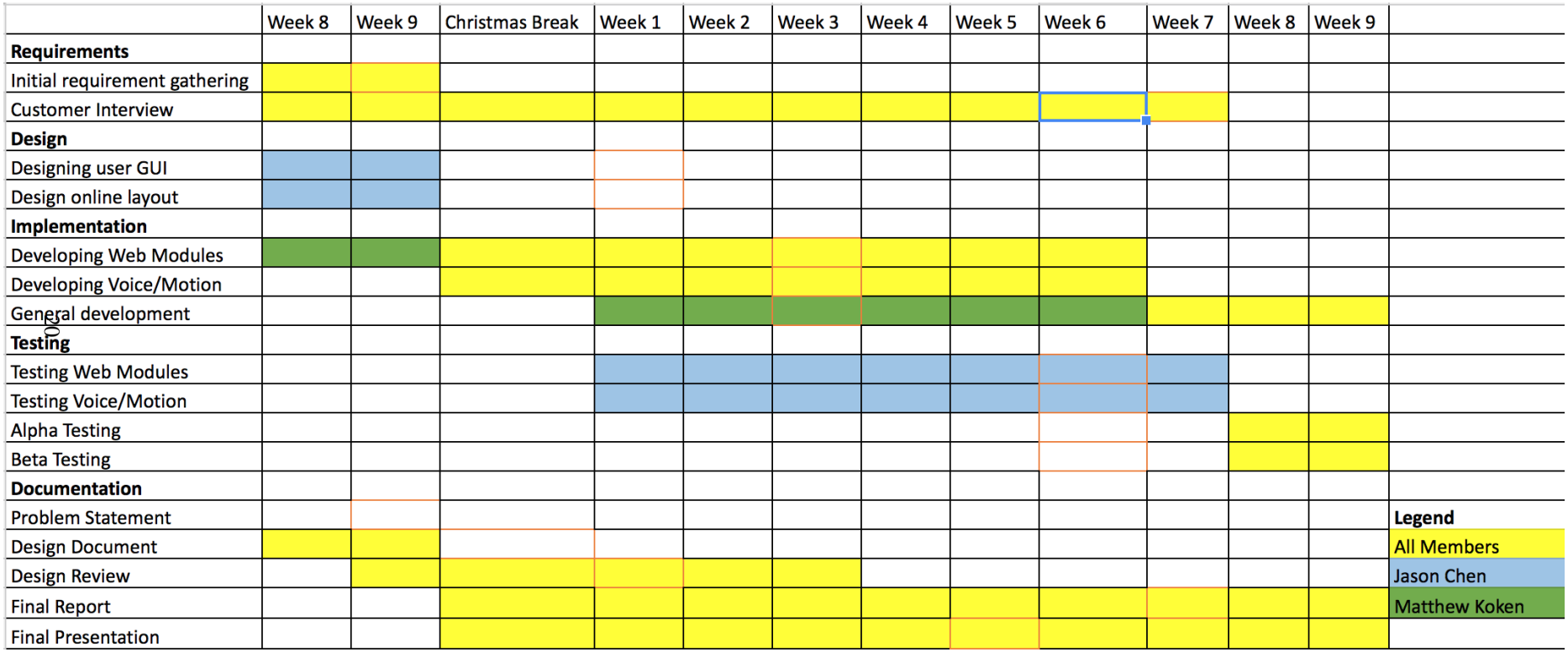

Figure 12.1: Timeline for Project

Chapter 13

User Manual

There are several main tasks related to the use of the SmartMirror system. The user manual provides relevant steps for the primary uses of the system as well as updating and developing for the system.

13.1 Initial Setup

In order to use SmartMirror, it must first be installed. Installation requires Node.js to be installed on the target system.

1. Download the latest source code from `https://github.com/patback66/SeniorDesign-JK`
2. Navigate to "Project/electron-boilerplate-master/"
3. Run "npm install" from your command line within that directory.
4. To start the application, run "npm start" from the command line within that directory.

13.2 Stopping the Application

Once the application is running, it may need to be stopped. The basic procedure is as follows:

1. With the application open, press the "ALT" key to bring up the browser menu.
2. Select the "Window" menu.
3. Select the "Close Window" option.

This may also be accomplished by the keyboard shortcut "CTRL+W". Some systems may also quit the application using "ALT+F4"

13.3 Alexa Voice Service

Our system currently uses the Amazon provided applications for the Alexa Voice Service. Follow the setup provided by Amazon available at `https://github.com/alexa/alexa-avs-sample-app`

13.4 Creating a Widget

1. Copy "widget.js" to a new file in Project/electron-boilerplate-master/src/clientjs
2. Uncomment the import statement at the top of the file.
3. Alter the class name to extend the widget class and name your class. Example: "export class Clock extends Widget"
4. Uncomment the call to the super constructor within "constructor()"
5. Set the name of your widget in "this.name = 'widgetName'".
6. Set the refresh rate in milliseconds. Default is 500.
7. Generate your HTML content in "loop()".

8. Set the HTML content using ”element.innerHTML = elementHTML”
9. In ”app.js”, import your widget.
10. In the ”loadWidgets” function, add your widget and its location.

Chapter 14

Lessons Learned

Throughout the year, there were many difficult challenges we faced. Through these challenges, we learned lessons that would help us deal with similar problems sure to be faced in the future.

14.1 Scoping

In particular, one of the greatest problems we faced was the scoping of the project. When trying to create a specific product, we first came up with an idea that was far out of our reach, especially as students with a schedule to keep up. We did not carefully consider how much time this idea would take to implement, as well as the difficulty of getting specifics down between only two group members. Therefore, through the course of time, our idea had to change a lot, discussions had to be made, before we were able to successfully settle on a final idea to work towards. However, the process of finalizing an idea, as well as minimizing the scope of the project in order to match our current team size, could have taken much less time, had we originally addressed this issue.

Chapter 15

Suggested Changes

There were many features we had hoped to integrate into our system, but we were compelled to cut them due to time constraints. However, given more time to work on the product, there are a few changes that we would make.

15.1 Integration with Amazon Alexa

One specific application of our device we wanted was to integrate the product with Amazon's Alexa, a very popular smart home device. We aimed to have the product be able to display information that the Alexa would return, upon a user's request. After further research, we realized that there were no applicable solutions to this implementation, as the only feasible option would be to have an outwards facing IP address, which cannot be done on the school network. We believe that in the future, there will be open source APIs that allows for easier integration with modules such as our own.

15.2 Further Simplification

While we feel our current model is quite sufficient, there are a few improvements that we felt would further simplify our design. For example, eliminating redundant functions and having everything work in a smaller amount of class functions would be less daunting on the developer side.

15.3 Quality of Life

Our model, while working, is far from polished. There are many things we could possibly add to make the experience much better for the consumer.

- CSS changes One issue in particular that we would like to work towards is polishing the display of widgets. While our model is scalable, the display can occasionally overlap each other, resulting in some widgets taking up too much room, outside their allocated grid space. This issue would be solved by further developing our CSS and HTML.
- Visual Settings One feature we did not successfully implement in time was the ability to have a settings page, allowing users to have the ability to decide where each widget would be placed in the grid space, without having to delve deep into code. This feature would primarily help the less tech-savvy users, and overall improve the design and functionality.
- Faulty App Detection One interesting feature for developers would be the ability to detect if applications were faulty off the bat, instead of having it affect the entire system. This way, developers would have an easier time testing their applications.

15.4 Alternative Inputs

In our original plan, we aimed to have alternative input sources, in order to allow a more smooth experience for users. Most users today are accustomed to touchscreen or voice controlled appliances, and we were hoping to have our system work in a similar fashion. However, this idea was far outside the scope of our project, and to implement this feature would take perhaps double the time. This feature, however, would highly improve the quality of this project.

Chapter 16

Conclusion

Our finished product, as shown above, is demonstrated on our mirror hardware model. The hardware is simple, and elegant enough to demonstrate our widget structure. Our real focus was towards the design of our overall widget class structure. Our final project has a few demonstrations of the widget class, such as the weather widget and the RSS feeder widget. These widgets, while simple, do demonstrate the feasibility of being able to create most ideas in JavaScript. As of right now, our platform is able to support simple JavaScript applications, and because of its scalability, it can be run on more than just our current hardware model.

Originally, we aimed to create a mirror that would act as a personal assistant to both developers and general consumers alike. However, the sheer scope of the idea, along with the vagueness of such a complex concept, forced us to rethink our idea into a more viable option. Eventually, we settled on the idea of focusing on the creation of the open source platform, specifically tailored for DIY developers. We felt that there was no effective open source projects that allowed for a hobbyist to create their own mirror, similar to ours, and thus we focused on creating a model that would allow for easier application development. We feel that our goal in creating such a model was achieved, as the feedback we have received helps confirm that this platform looks and feels easy to develop with. We hope that in the future, perhaps more work and more effort can be added, either by us or others, in order to further improve our platform.

Bibliography

[1] H. Mcheick, "Ubiquitous computing and context-aware applications: Survey and contributions," in *2016 IEEE First International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, pp. 394–397, June 2016. This paper discusses the current research into the field of ubiquitous computing, a field which tackles similar material to our current senior project. The paper discusses some of the most important properties when developing ubiquitous computing/IoT.

[2] A. Yousfi, A. de Freitas, A. K. Dey, and R. Saidi, "The use of ubiquitous computing for business process improvement," *IEEE Transactions on Services Computing*, vol. 9, pp. 621–632, July 2016. This paper discusses the current state of our computing era, one in which business as a whole is being heavily affected by the current state of rising IoT technology. IoT technology is affecting all business, due to the rise of need for businesses to further optimize their performance in competition. This paper provides an interesting knowledge towards trying to focus our product towards a business consumer market.

[3] G. Xu, Y. Ren, G. Zhang, B. Liu, X. Li, and Z. Feng, "Hycpk: Securing identity authentication in ubiquitous computing," in *2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom)*, pp. 239–246, Aug 2015. This paper discusses HyCPK, a new identity authentication algorithm created specifically for ubiquitous computing products. It provides an interesting direction in terms of cybersecurity, and also provides a good look into security and authentication for our senior project.

[4] B. Lee, "Hybrid-style personal key management in ubiquitous computing," in *2014 11th International Conference on Security and Cryptography (SECRYPT)*, pp. 1–6, Aug 2014. This paper discusses ID-based cryptography for multiple computing devices. It provides an interesting look into cryptographic methods for a user communicating with multiple devices, which would be useful for a project such as ours which introduces another IoT device to the world.

[5] S. Yagmur, "A literature review: Usability aspects of ubiquitous computing," in *2016 International Conference on Platform Technology and Service (PlatCon)*, pp. 1–6, Feb 2016. This paper discusses usability in ubiquitous computing. It mentions the most important aspects to consider when designing a new ubiquitous computing item, as well as discussing the research trends towards new development in the field of ubiquitous computing. This article is useful to both the development aspect, as well as reading up on trends of IoT development.

[6] P. O'Donovan, "The tv's vanishing act: radical new display and content-delivery technologies will doom the television set.," *IEEE Spectrum*, vol. 53, pp. 48–53, May 2016. This paper describes the history behind the rise and fall of our "home television, and the trends of consumers as the idea of the "home television is soon coming to an end. This paper is not a technical concept, rather more of a study into consumer market trends and history. It will allow us to understand what exactly our product is aiming to replace, and what functions and needs it should seek to replace for the modern consumer.

[7] N. Montfort and I. Bogost, "Random and raster: Display technologies and the development of videogames," *IEEE Annals of the History of Computing*, vol. 31, pp. 34–43, July 2009. This paper discusses the changes we see in the development of graphics in the video game world. It summarizes the development of random-scan, raster-scan, and now "flat-scan technologies that are used nowadays for displaying games. While not a direct requirement for our project, it could prove useful, especially for a side module that could become one of the ideas we end up advertising about our project.

[8] T. Nakajima, "How to reuse exisiting interactive applications in ubiquitous computing environments?," in *Proceedings of the 2006 ACM Symposium on Applied Computing*, SAC '06, (New York, NY, USA), pp. 1127–1133, ACM, 2006. This article discusses how to reuse GUI toolkits for new ubiquitous computing development. It discusses how separating the GUI from the output device are important for creating a model that is extensible. This will be specifically useful if we try to connect different types of interactions to our specific GUI.

[9] V. Heun, S. Kasahara, and P. Maes, "Smarter objects: Using ar technology to program physical objects and their interactions," in *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '13, (New York, NY, USA), pp. 2939–2942, ACM, 2013. This article discusses an idea on combining a TUI with a GUI for a simpler user interface. This type of information could be useful in regards to our project, because our idea also markets to a similar user, and information in regards to this idea could be implemented in our own development.

[10] J. Hounshell and T. Emma, "The invisible interface: Traditional design principles in modern electronic design," in *2016 20th International Conference Information Visualisation (IV)*, pp. 375–379, July 2016. This article describes the steps in order to create an interface that allow a users experience to be as smooth as possible. It discusses spacing, continuity, color usage, and game design, with the term "invisible interface used to describe the relation the user ends up having with the display. This is useful due to the fact that we are designing our own display, and to have an optimized design would be extremely beneficial for our product.

Appendix

APIs

Source code for the project and full documentation is available at `https://github.com/patback66/SeniorDesign-JK`. The essential API elements for the Widget class and the client main are included here.

Widget

Widget provides functionality for an HTML widget. Extend the widget class to create a new widget.
Class Defined in: clientjs/widget.js:2

.0.1 Constructor

Widget

Widget provides functionality for an HTML widget. Defined in clientjs/widget.js:2

.0.2 Item Index

Methods

- close
- display
- getname
- getRefresh
- loop
- open
- setLocation
- setRefresh
- setup

.0.3 Methods

close

Defined in clientjs/widget.js:51
[WIP] Closes a widget that has been made fullscreen. Remember to close else you have 3 million pages open

display

Defined in clientjs/widget.js:79
[WIP] Gets the widget's display status **Returns:** String:
Gives the widget display status

getname

Defined in clientjs/widget.js:42
Gets the widget's name as a string **Returns:** String:
returns the widget's name, as defined by this.name

getRefresh

Defined in clientjs/widget.js:93
Gets the widget's refresh rate in milliseconds. **Returns:** Number:
Returns the widget's refresh rate

loop

Defined in clientjs/widget.js:31
Loop is called periodically based on this.refresh. Updates to the widget should be performed here.

open

Defined in clientjs/widget.js:111
[WIP] Fullscreens a widget using a new window.

setLocation

- location

Defined in clientjs/widget.js:70
Set the location id for the widget. **Parameters:**

- location String
  - the div id for the widget to be placed in.

setRefresh

- rate

Defined in clientjs/widget.js:102
Sets the refresh rate for the widget in milliseconds. **Parameters:**

- rate Number
  - the time in milliseconds between calls to loop

setup

- location

Defined in clientjs/widget.js:20
Setup is called once at the instantiation. Use for code that must run after the constructor but before the main loop. **Parameters:**

- location String
  - a string to specify the id of the div location for the widget

Client Main

The client main handles the primary functions of the web application on the client side. This loads and runs widgets for display.
Defined in app.js

.0.4 Item Index

Functions

- loadWidgets

.0.5 Functions

loadWidgets

Loads all widgets and sets their location for the widget handler. Users should place new widgets within this function. The available grid locations are:

- region-top
- region-top-left
- region-top-center

- region-top-right
- region-middle
- region-middle-left
- region-middle-right
- region-bottom
- region-bottom-left
- region-bottom-center
- region-bottom-right