Raspberry PI Smart Mirror Project Requirements Document

Scott English
Thomas Sanchez
Ryan Faulkenberry
Teegan Duong

Preface

The expected readership of this document is intended to be the development team, as well as the product owner. This document is designed to give an overview of the requirements for reference during product development and software evolution. This is the first version of this document. This version is to initialize the records of our system's functionality requirements and specifications. Here is a presentation of our systems functional and non-functional requirements:
https://docs.google.com/presentation/d/1k7xSSUYxYrGa0_cn64UF977MBaggikPGN2zPauNCR6I/edit#slide=id.g1eb50b5f59_0_0

1. Introduction

The Smart Mirror software provides a customizable user interface for the Raspberry Pi Smart Mirror Project. The Smart Mirror software will have several functionalities such as: customizable user interface that can be remotely accessed via a PC,a customizable clock, movable widgets, customizable weather feed from a weather database, and customizable text snippets.

1.1 Purpose of the Smart Mirror

We are developing this software because there is a lack of Raspberry Pi Smart Mirror software that both interfaces with a PC and is highly customizable. The lack of interfacing software that is both optimized and doesn't overheat the Raspberry Pi, is also a common problem with this project and we believe our software will fix this as well.

2. Glossary

- **Raspberry Pi**: is a tiny affordable computer. (https://www.raspberrypi.org)
- **DIY**: Do it yourself, common term when describing projects that involve doing all of the work yourself
- **One-way mirror**: A mirror that is partially reflective and partially transparent. Allows the user to see through while still seeing their reflection as if they were staring into a mirror
- **Smart Mirror Project**: a DIY project where you take a one-way mirror and put a monitor behind it and display images through the mirror as you look at yourself in the mirror.
- **SDL2**: Simple DirectMedia Library, we are using to enable our GUI and keyboard/mouse input
- **SQLite3**: Structured Query Language, we are using the SQLite3 version of the SQL software to implement a self-contained, serverless, zero-configuration,transactional database that is local to the Raspberry Pi. (https://www.sqlite.org/about.html)
- **Raspbian**: The operating system most commonly used on Raspberry Pi's
- **Widget**: an application, or a component of an interface, that enables a user to perform a function or access a service.
- **Step**: A single iteration of the program loop.
- **State**: A configuration of variables at a certain time in a program's execution.
- **GUI**: Graphical User Interface that we are implementing using SDL.
- **RAM**: Random Access Memory that we will be accessing when communicating with different widgets.
- **I/O:** An input or output device.
- **RSS feed:** a web channel that contains frequently updated data or information.

3. User Requirements Definitions

3.1 Functional Requirements:

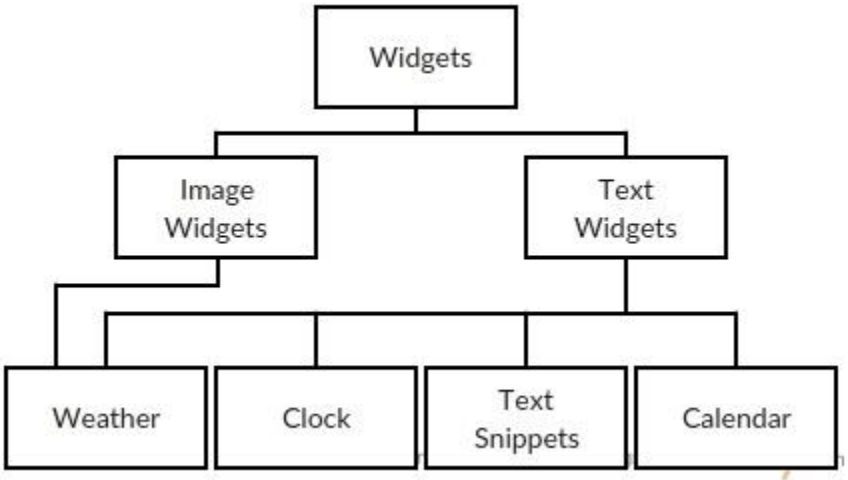
- Clock
  - The clock shall display the time based on user's location.
- Weather
  - The weather shall display weather conditions for the user.
- Changing Text Snippet
  - The text snippet shall display randomized text from a source of quotes.
- PC connectivity
  - A user should be able to connect to the software remotely with a personal computer.
- Text Snippet editor
  - A user should be able to add or remove quotes to be randomly selected.
- Calendar
  - The calendar should notify the user of the date and the appointments for the current day.

3.2 Non-functional Requirements:

- Efficiency
  - The system shall run efficiently to prevent overheating.
- Security
  - The system should remain secure with its user information to prevent data loss and privacy invasion.
- Adaptability
  - The system shall adapt to a display by resizing and changing the size of widgets.
  - The system should be adaptable to multiple platforms and operating systems

4. System Architecture

4.1 Class Definitions:



Class Hierarchy Diagram (figure 4.1)

Widget: A group of visible objects that will be managed by a common *widget handler*.

- Can be moved to a new screen position, hidden, or resized by a user.
- Linked to the system clock to optimize performance.
- Modified by user mouse or keyboard input, or from a remote PC.

Image Widgets: A group of widgets that display transparent images.

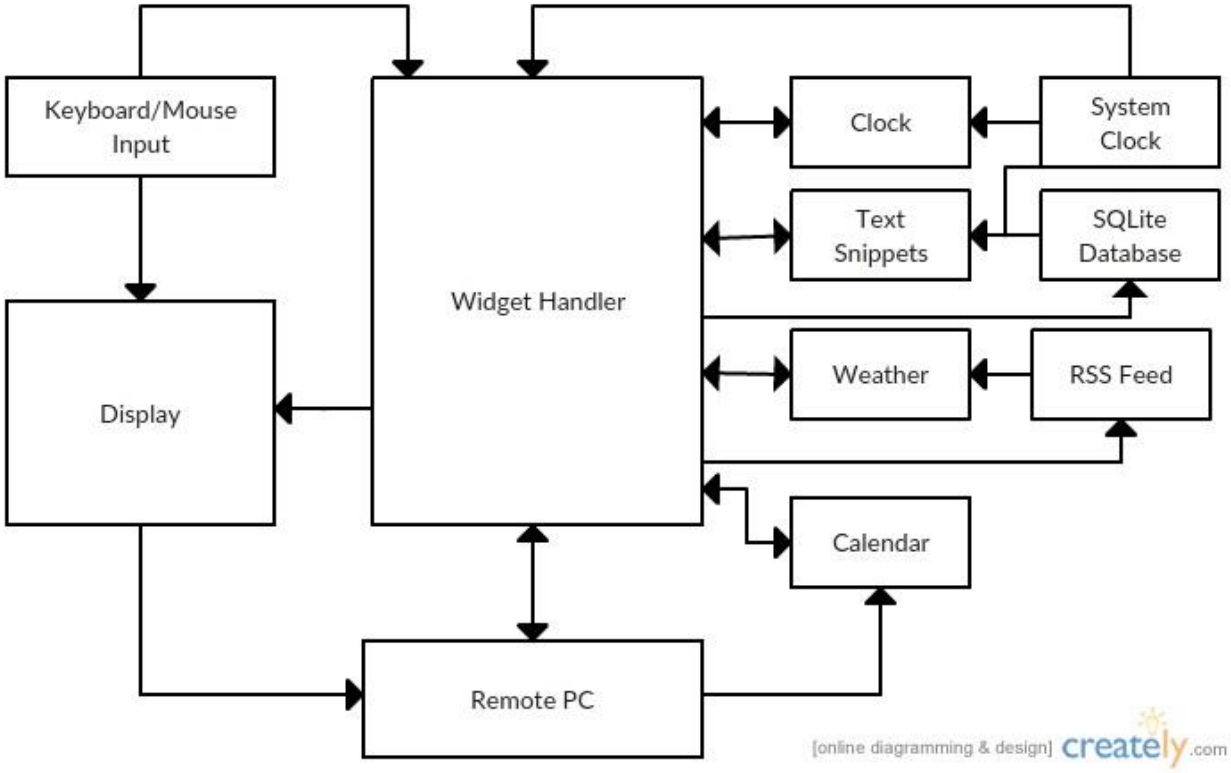
- Due to the need for image transparency, .png format shall be used.
- Shall allow for image changing.

Text Widgets: A group of widgets that display rich text.

- Shall allow for text or font change.

Weather: A linked pair containing an image widget and a text widget, which together display the current weather and temperature.
Clock: A text widget that displays the current local time.
Text Snippets: A randomized changing text widget that displays a random quote.
Calendar: A text widget that displays the current date and today's appointments.

4.2 Module interaction



(figure 4.2)

Outputs from each module, from left to right, top to bottom:

Keyboard/Mouse Input interactions:

- Display: Mouse interactions shall be telegraphed on the display.
- Widget Handler: The widget handler shall take keyboard and mouse input for use in modifying the widgets.

Display interactions:

- Remote PC: The display shall be echoed on the remote PC.

RSS Feed interactions:

- Weather: The RSS Feed shall, when prompted, provide the current local weather to the weather widget.

Widget Handler interactions:

- Display: The widget handler shall send the information about the current state of every widget in the system each step.
- Remote PC: Every change made to the widget shall be updated in the state of the remote PC.
- Calendar: The widget handler shall modify the position and state of the calendar based on the input from the user.
- Clock: The widget handler shall modify the position and state of the clock based on the input from the user.
- Text Snippets: The widget handler shall modify the position and state of the text snippets based on the input from the user.
- Weather: The widget handler shall modify the position and state of the weather based on the input from the user.
- RSS Feed: The widget handler shall change the user location for the RSS Feed based on user input.
- SQLite Database: The widget handler shall change the stored text snippets within the SQLite database based on user input.

Remote PC interactions:

- Widget Handler: The remote PC should act as a secondary source of keyboard and mouse input for the widget handler.
- Calendar: The remote PC should provide the data to populate the calendar widget.

Clock interactions:

- Widget Handler: The clock shall, when prompted, send data about the current time to the widget handler.

Text Snippets interactions:

- Widget Handler: The text snippets shall, when prompted, send data about the current quote to the widget handler.

Weather interactions:

- Widget Handler: The weather shall, when prompted, send data about the current local weather to the widget handler.

Calendar interactions:

- Widget Handler: The weather should, when prompted, send data about the current user appointments, and the current date to the widget handler.

System Clock interactions:

- Widget Handler: The System Clock shall continuously send information to the widget handler for use in any timers used in widget optimization.
- Clock: The System Clock shall, when asked, update the current time on the clock widget.
- Text Snippets: The System Clock update the text snippet timer, so that after a set period the text is changed to a new randomized text snippet.

SQLite Database interactions:

- Text Snippets: The SQLite Database shall, when prompted, provide a random quote.

5. System Requirements Specification

5.1 *Clock:*

Function: Displays the current time.
Description: Displays the current set system time and change the current set system time given system manager input. A system manager shall be able to move and hide the clock, and they should be able to resize the clock.
Input: Keyboard input; current time.
Source: I/O device keyboard; current time from console.
Output: Size and position adjustments; Current time in a 12 hr format.
Destination: Display screen.
Requirements: The system clock must be accessible. The display window must be running.

5.2 *Weather:*

Function: Displays the current local weather
Description: Takes weather data from RSS feed and displays a linked pair of widgets, displaying the local temperature and weather type. A system manager shall be able to move and hide the weather, and they should be able to resize the weather.
Input: Keyboard input; current time.
Source: I/O device keyboard; current weather from RSS feed.
Output: Size and position adjustments; Current local weather.
Destination: Display screen.
Requirements: Internet connection. Access to RSS feed (specific RSS feed still to be determined). The display window must be running.

5.3 *Changing Text Snippet:*

Function: Displays the currently selected text snippet, changing selection after a set period
Description: Displays the text snippet last selected from the SQLite database. If the time since last selection surpasses the changing period (default 1 hour), the SQLite database shall be queried for a new random text snippet. A system manager shall be able to move and hide the text snippet, and they should be able to resize the text snippet.
Input: Keyboard input; time since last update.
Source: I/O device keyboard; current time from console; stored last update time.
Output: Size and position adjustments; Current selected text snippet.
Destination: Display screen.
Requirements: SQLite Database must be open and populated. The display window must be running.

5.4 *PC Connectivity:*

Function: Allows a remote personal computer to connect to the system through a local area network.
Description: A system manager should be able to connect to the smart mirror GUI system while on the same local area network. While connected the PC should see the display and be able to make any modifications that a system manager could make if connected to the Raspberry PI directly.
Input: PC Keyboard input; PC mouse input.
Source: I/O device remote PC.
Output: Size and position adjustments of any system widgets.
Destination: Widgets within the system.
Requirements: Internet connection to a remote PC, display window must be running.

5.5 *Text Snippet Editor:*

Function: Edits the text in the database
Description: The system manager shall be able to edit the text snippets held inside of the local database. The system manager shall be able to remove text snippets, create new text snippets, and revise current text snippets. The software shall not have any data loss in this process.
Input: Keyboard input or connection via remote PC.
Source: SQLite Database.
Output: Updating current SQLite Database.
Destination: SQLite Database.
Requirements: SQLite Database must be open and secure.

5.6 *Calendar:*

Function: Displays the date and events
Description: Displays the current date and events happening on that day to the user. System managers can change calendar info and update events happening on that day. A system manager shall be able to move and hide the calendar, and they should be able to resize the calendar.
Input: Keyboard and mouse input or connection via remote PC.
Source: Current date and events from the remote PC.
Output: The current date followed by the events happening that day
Destination: Display screen.
Requirements: Internet connection to a remote PC, display window must be running.

5.7 *Efficiency:*

- The software shall never cause the system to overheat or overextend its current data storage capacity.
    - The heat shall not exceed 50℃while within a well ventilated area.
    - The software shall not exceed more than 1gb in storage.
    - The software shall not exceed more than 1/2gb in RAM.
- System managers shall be able to use the software constantly without it overheating. They shall also be able to use widgets without having to worry about using more system data storage then what is available.

5.8 *Security:*

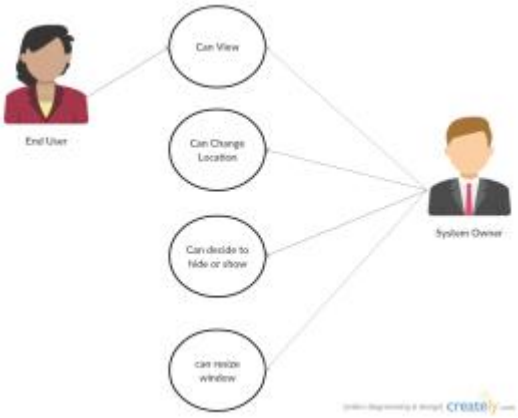
- All personal data obtained shall be stored within private fields and shall not be accessed while the device is actively accessing functionalities that connect to the internet.
- No module or subsystem shall hold personal info longer than is necessary to perform system functions.

5.9 *Adaptability:*

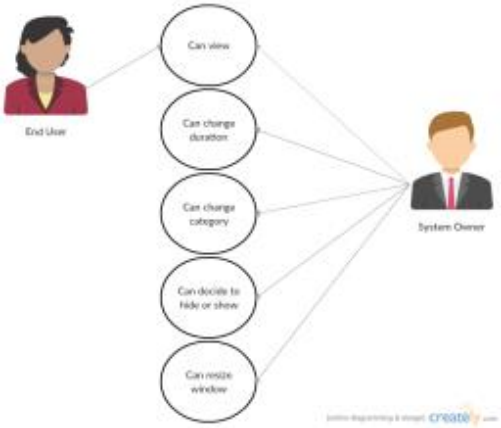
- The software shall adapt to display devices of any resolution.
- The software should run on multiple system architectures (ARM, x86).

6. System Models

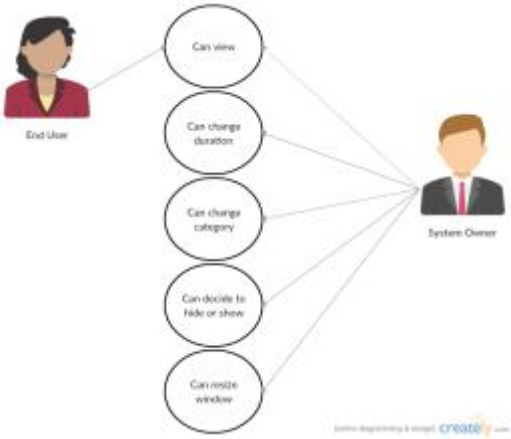
6.1 Clock Case Diagram



6.2 Weather Case Diagram



6.3 Text Snippet Case Diagram



7. System Evolution

7.1 Architecture

- The software shall work on any computer using x86 or ARM architecture, so the release of further Raspberry Pi versions is not an issue.
- Only minor tweaking shall be required to run the software on any non-conventional architectures.

7.2 Operating System

- The software shall run on Raspbian.
- Should a demand for Smart Mirrors using non-Raspberry Pi computers occur, we should release compatible software for popular desktop operating systems.

7.3 Software Features

- As the market for the product grows, there may be demand for more features.
- Low internet use is expected in the current system (30-60 minute refresh rate for rss), but the addition of more features may require more network use.
- The software should run efficiently to avoid overuse of the Raspberry Pi CPU, which could cause overheating. When adding features, overheating should be taken into consideration.

7.4 Non-Electronic Hardware

- The Raspberry Pi shall display widgets to a one-way smart mirror.
- The Raspberry Pi shall be stored in such a way that the necessary amount heat exchange with the environment is achieved. Should heat needs increase due to more CPU usage, the Raspberry Pi's container should be altered to ensure safe temperatures.

8. Appendices

8.1 Hardware Requirements

- The Raspberry Pi's hardware defines our minimum and optimal configurations for our software.
    - Our storage is limited by the Pi's limited SD Card space
    - Our software usage is limited by the Pi's single core processor and limited RAM
- Because of these limitations we are using SQLite and SDL to implement our GUI's and Data Management

8.2 SQLite Database

- We shall use SQLite as a database engine for the logical organization of our software data.
- SQLite is most appropriate for this situation because of its ability to utilize minimal system resources to access and store data.
- SQLite database shall be used in our software so we can implement the changing text snippet and our text snippet editor functions with little system resource usage.

8.3 SDL2

- We shall use SDL in our software to render our GUI and Widget functions
- SDL is the most cost effective way (in terms of system resources) to implement our widget functionality without the problem of overheating our single core processor on the Raspberry Pi.
- SDL is also the simplest directmedia library that is compatible with almost all operating systems.