Ryan Wiesenberg

rwies@umich.edu

Abstract

Software Requirements Specification (SRS) based on Bruegge & Dutoit for a smart mirror.  
The goal of the smart mirror is to aggregate and present daily and upcoming tasks, aggregated for the user approaching the mirror based on supported, linked accounts.

Smart Mirror

CIS 553 Term Project

Table of Contents

[1 Introduction 2](#_Toc88900986)

[2 Current System 2](#_Toc88900987)

[3 Proposed System 2](#_Toc88900988)

[3.1 Overview 2](#_Toc88900989)

[3.2 Functional Requirements 2](#_Toc88900990)

[3.3 Nonfunctional Requirements and Constraints 3](#_Toc88900991)

[3.4 System Models 3](#_Toc88900992)

[3.4.1 Scenarios 3](#_Toc88900993)

[3.4.2 Use Case Model 5](#_Toc88900994)

[3.4.3 Object Model 8](#_Toc88900995)

[3.4.4 Dynamic Model 8](#_Toc88900996)

[3.4.5 User Interface 8](#_Toc88900997)

[4 Glossary 8](#_Toc88900998)

# Introduction

The purpose of this document is to outline the software requirements and resultant system models for the **Smart Mirror** project. The goal of this project is to provide a centralized system for multiple users to interact with to see upcoming tasks for themselves and the group. This system is expected to contain a separate list of tasks for individual users and a list of tasks for the group. Additionally, the system should be able to differentiate users and present tasks and other information relevant to them.

The next section, Current System, details the current system state, based on user interviews. The Proposed System section then seeks to outline how the proposed system will seek to address the gaps in the current system, the system requirements, and any constraints on the system design. Finally, the Proposed System section will outline the resulting system models from the requirements specification and the goal user interface.

# Current System

Currently this system does not exist in an aggregated format and users must independently, manually view their upcoming tasks and mentally compare their deadlines and requirements. This creates undue stress for the user and detracts from making progress on the tasks themselves. Additionally, it is easy to forget one of the many systems used to track upcoming tasks and events and miss a task that is due or has been in the queue for an extended period without action.

# Proposed System

## Overview

The primary goal of the **Smart Mirror** system will be to minimize the obstructions to the users as rephrased from the above:

* Multiple task tracking systems or applications
* Maintaining a mental model of task priority
* Obscured visibility of tasks due to infrequent access

For ease of user access, the system should also be able to provide the following:

* configuration for different users
* method to authenticate and validate which user is accessing the system

This system is expected to run on low-end static hardware, affixed to a wall as a touch screen behind a mirror-like device. This document will not detail the physical element construction process but will aim to minimize computation overhead and design all UI elements and user interactions for a touch-screen device for ease of integration into the hardware.

## Functional Requirements

This section will organize itself now based on the goals listed above:

1. The system must be able to connect to multiple task tracking systems
   1. The systems should at least include access to a user’s Trello and Google Calendar.
   2. All task systems linked should be shown adjacent in the same calendar and task list
   3. A user should be able to complete tasks by accessing the mirror
2. Mental model maintenance
   1. All tasks should optionally be able to store a due date or no due date
   2. All tasks should be able to signal the user if
      1. They have not been worked on in some time
      2. The due date is coming up
   3. A user should be able to check current individual tasks and group tasks
      1. Sort by due date, name, last accessed, and people involved
      2. There should be a calendar view to see the soon to be due tasks
3. Obscured visibility of tasks due to infrequent access
   1. A user must be able to view tasks that have not been accessed or worked on in some time
   2. A user can request a random task with infrequent access to complete
4. Configuration for different users
   1. Multiple users, and the ability to add users
   2. Users must be able to access and configure multiple linked task systems
5. Method to authenticate and validate which user is accessing the system
   1. The user must choose a user from the menu and then sign in with a passcode
   2. If possible, the system should be able to authenticate the user with a camera

## Nonfunctional Requirements and Constraints

* The data storing the passcode does not have to be very secure, but the system should not be open to access from the internet.
* The system should minimally be supported by a RaspberryPi Zero or similar device that can support a touch screen monitor.
* A user needs access at any hour of the day
* The user must have a convenient way to manage the system as an administrator
* Uptime is not important this a non-critical component
* The product must be usable through a touchscreen and have a black background with primarily white text to promote the ability to use it as a **Smart Mirror**.
* Standby screen should be primarily black

## System Models

The following section seeks to define the different system models and resulting models. In these models two actors will be used:

* **MirrorUser**: Possesses the mirror and an account on the mirror and has connected their mirror account to other allowed task services.
* **TaskCreator***:* Creates tasks on an external system which are then retrieved by the **Smart Mirror**. This may be a mirror user as well, but they will be considered two separate actors for model design purposes.

### Scenarios

|  |  |
| --- | --- |
| *Scenario Name* | viewCuurentTrelloTasks |
| *Participating Actor Instances* | Alice: **MirrorUser** |
| *Flow of Events* | * + - 1. Alice wishes to look at the current Trello tasks need to be completed to decide what to work on today and if there are any time-sensitive tasks. Alice approaches the mirror which confirms her identity       2. Once her identity is confirmed, she activates the “View Tasks” function       3. Alice filters the tasks down to just Trello to view and sorts them by due date. She confirms her input and waits for the resulting task list.       4. Alice views the resulting tasks and presses on one to view more information. Alice reviews the information including any required sub-tasks and additional information before closing the task and walking away to begin on it       5. After a period of inactivity, the mirror realizes Alice has walked away and logs out of her task view. |

|  |  |
| --- | --- |
| *Scenario Name* | viewUnvisitedTrelloTasks |
| *Participating Actor Instances* | Alice: **MirrorUser** |
| *Flow of Events* | 1. Alice wishes to look at unvisited Trello tasks to decide what to work on today. Alice approaches the mirror which confirms her identity 2. Once her identity is confirmed, she activates the “View Unvisited Tasks” function. 3. Alice filters the tasks down to just Trello. She confirms her input and waits for the resulting task list. 4. Alice views the tasks displayed by the mirror as being untouched or unfinished the longest and presses on one to view more information. Alice reviews the information including any required sub-tasks and additional information before closing the task and walking away to begin on it 5. After a period of inactivity, the mirror realizes Alice has walked away and logs out of her task view. |

|  |  |
| --- | --- |
| *Scenario Name* | invitedToEvent |
| *Participating Actor Instances* | Rodrigo: **TaskCreator**  Alice: **MirrorUser** |
| *Flow of Events* | 1. Rodrigo would like to add a new task for Alice to complete, in this case inviting her to a get together. Rodrigo sends the invite to Alice’s Google Account. 2. The next time Alice uses the mirror, the mirror shows a notification alerting her of a new task since the last time she has viewed the **Smart Mirror** 3. Alice can use the “View Notifications” function and then view the new task that was added. Alice then selects the new task to view the details like location and date of the task 4. After a period of inactivity, the mirror realizes Alice has walked away and logs out of her task view. |

|  |  |
| --- | --- |
| *Scenario Name* | connectGoogleAccount |
| *Participating Actor Instances* | Alice: **MirrorUser** |
| *Flow of Events* | 1. Alice would like to connect a new account to her mirror profile. Alice approaches the mirror which confirms her identity 2. Once her identity is confirmed, she activates the “Connect Account” function. 3. The mirror then displays the account options that Alice has not connected yet of the available connection types. Alice selects her desired account to connect. 4. The mirror then prompts her to log into the service to like the account and retrieves the existing task list. 5. The mirror then displays the new tasks added in the “View Tasks” view filtered by the new account. 6. After a period of inactivity, the mirror realizes Alice has walked away and logs out of her task view. |

|  |  |
| --- | --- |
| *Scenario Name* | newUser |
| *Participating Actor Instances* | Alice, Morgan: **MirrorUser** |
| *Flow of Events* | 1. Alice would like to add a new user, Morgan, to the **Smart Mirror** to be able to view their tasks. Alice approaches the mirror which confirms her identity 2. Once her identity is confirmed, she activates the “Add New User” function for Morgan to fill in the information required to confirm his identity in the future 3. Morgan then fills this information in and confirms their information. If this information is not filled in before the period of inactivity the add user session is canceled. 4. The mirror then transfers to Morgan’s new profile and prompts them to connect their first account. |

### Use Case Model

|  |  |
| --- | --- |
| *Use case name* | **ValidateUser** |
| *Participating actors* | Initiated by and Communicates with **MirrorUser** |
| *Flow of events* | 1. The **MirrorUser** approaches the mirror and interacts with it    1. The **Smart Mirror** responds by presenting the login screen and validating the user with a passcode or automatically validating with a video camera. The **Smart Mirror** displays the user’s **HomeScreen** with all the valid actions for the user to take. |
| *Entry condition* | * None |
| *Exit conditions* | * The user can now browse the available **Smart Mirror actions** |
| *Quality requirements* | * The passcode entry is secure and does not display the fully entered passcode visibly |

|  |  |
| --- | --- |
| *Use case name* | **ViewTasks** |
| *Participating actors* | Initiated by and Communicates with **MirrorUser** |
| *Flow of events* | 1. The **MirrorUser** activates the “View Tasks” function of the Smart Mirror    1. The **Smart Mirror** responds by presenting a list of tasks to the user, automatically sorted with the most recent at the top of the list. 2. The **MirrorUser** activates the “Filter Tasks” function in the “View Tasks” menu and selects the task source filters to view the limited list based on the requirements.    1. The **Smart Mirror** responds by presenting a list of tasks to the user with the new sorting and filtering as specified by the user |
| *Entry condition* | * The **MirrorUser** is validated by the **Smart Mirror** |
| *Exit conditions* | * The **MirrorUser** can view and explore the tasks that have been filtered by the **Smart Mirror** |
| *Quality requirements* | * The **Smart Mirror** must be able to detect inactivity and log the user out to preserve personal task privacy |

|  |  |
| --- | --- |
| *Use case name* | **ViewUnvisitedTasks** |
| *Participating actors* | Initiated by and Communicates with **MirrorUser** |
| *Flow of events* | 1. The **MirrorUser** activates the “View Unvisited Tasks” function of the Smart Mirror    1. The **Smart Mirror** responds by presenting a list of tasks to the user, pre-filtered by date since last interaction 2. The **MirrorUser** activates the “Filter Tasks” function in the “View Tasks” menu and selects the task source filters to view the limited list based on the requirements.    1. The **Smart Mirror** responds by presenting a list of tasks to the user with the new sorting and filtering as specified by the user |
| *Entry condition* | * The **MirrorUser** is validated by the **Smart Mirror** |
| *Exit conditions* | * The **MirrorUser** can view and explore the tasks that have been filtered by the **Smart Mirror** |
| *Quality requirements* | * The **Smart Mirror** must be able to detect inactivity and log the user out to preserve personal task privacy |

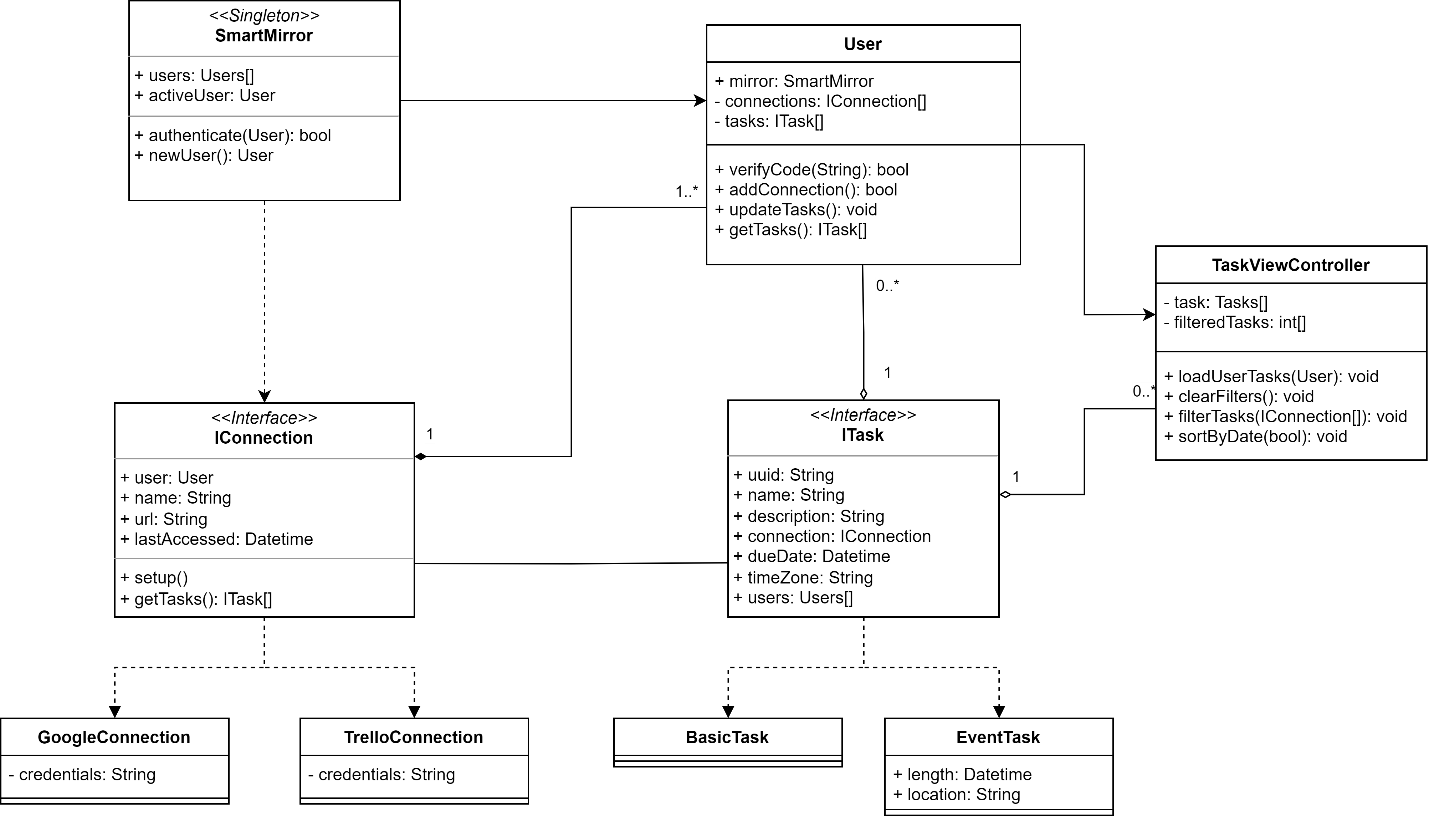
|  |  |
| --- | --- |
| *Use case name* | **AddNewTask** |
| *Participating actors* | Initiated by the **TaskCreator**  Communicates with **MirrorUser** |
| *Flow of events* | 1. The **TaskCrator** creates a task and invites the **MirrorUser** to complete the task    1. The **Smart Mirror** retrieves the new task from the external service and prepares a notification for the next time the **MirrorUser** approaches the Smart Mirror 2. The **MirrorUser** approaches the mirror    1. The **Smart Mirror** detects that the **MirrorUser** with an outstanding notification has approached the mirror and displays the notification for the user to open 3. The **MirrorUser** can choose to open the notification or ignore it to open later    1. If the notification is opened, the **Smart Mirror** will not show the notification in the future, otherwise the notification will be highlighted the next time the **MirrorUser** approaches the mirror |
| *Entry condition* | * The **MirrorUser** is validated by the **Smart Mirror** |
| *Exit conditions* | * The **MirrorUser** can view the newly added task through the **Smart Mirror** |
| *Quality requirements* | * The **Smart Mirror** must be able to detect inactivity and log the user out to preserve personal task privacy |

|  |  |
| --- | --- |
| *Use case name* | **ConnectAccount** |
| *Participating actors* | Initiated by and Communicates with **MirrorUser** |
| *Flow of events* | 1. The **MirrorUser** activates the “Connect Account” function of the **Smart Mirror**    1. The **Smart Mirror** responds by presenting the list of valid account types that have not been connected yet 2. The **MirrorUser** selects the type of account they wish to add    1. The **Smart Mirror** responds by presenting a form requesting the valid information to connect the new account type 3. The **MirrorUser** enters their account information, reviews it, and then submits to connect to the account    1. The **Smart Mirror** confirms the account information and attempts to connect to the account. If unsuccessful the **Smart Mirror** will restart the above step and re-prompt the user to enter the account information. When successful, the **Smart Mirror** then displays the tasks in the newly connected account in a pre-filtered “View Tasks” view |
| *Entry condition* | * The **MirrorUser** is validated by the **Smart Mirror** |
| *Exit conditions* | * The **MirrorUser** can now browse the tasks from the newly connected account |
| *Quality requirements* | * The account connection process is secure and does not reveal the account’s private information * The **Smart Mirror** must be able to detect inactivity and log the user out to preserve account and task privacy |

|  |  |
| --- | --- |
| *Use case name* | **AddNewUser** |
| *Participating actors* | Initiated by an existing **MirrorUser**  Communicates with a new **MirrorUser** |
| *Flow of events* | 1. The existing **MirrorUser** activates the “Add New User” function    1. The **Smart Mirror** responds by opening a prompt asking for the new **MirrorUser** to enter their information for their new account including a secure passcode 2. The new **MirrorUser** enters their desired account information, reviews it, and submits the information    1. The **Smart Mirror** responds by following the **ConnectAccount** function above |
| *Entry condition* | * The existing **MirrorUser** is validated by the **Smart Mirror** |
| *Exit conditions* | * The new **MirrorUser** can view and explore the tasks of the initially added account |
| *Quality requirements* | * The passcode entry is secure and does not display the fully entered passcode visibly * The **Smart Mirror** must be able to detect inactivity and log the user out to preserve personal privacy * The **Smart Mirror** should prompt the new **MirrorUser** to confirm the new account information twice to ensure the correct information has been entered into the system |

|  |  |
| --- | --- |
| *Use case name* | **AddInitialUser** |
| *Participating actors* | Initiated by and Communicates with the first **MirrorUser** |
| *Flow of events* | 1. The first **MirrorUser** approaches the **Smart Mirror**    1. The **Smart Mirror** responds by following the **AddNewUser** steps above, treating the initial user as pre-validated by the **Smart Mirror** |
| *Entry condition* | * No **MirrorUser** has been added to the **Smart** **Mirror** |
| *Exit conditions* | * The new **MirrorUser** can view and explore the tasks of the initially added account |
| *Quality requirements* | * The passcode entry is secure and does not display the fully entered passcode visibly * The **Smart Mirror** must be able to detect inactivity and log the user out to preserve personal privacy * The **Smart Mirror** should prompt the new **MirrorUser** to confirm the new account information twice to ensure the correct information has been entered into the system |

### Object Model



### Dynamic Models

### User Interface

# Glossary