Ryan Wiesenberg

rwies@umich.edu

Abstract

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

Contents

[1 Introduction 1](#_Toc90392541)

[1.1 Current System 1](#_Toc90392542)

[2 Project Requirements 2](#_Toc90392543)

[2.1 Overview 2](#_Toc90392544)

[2.2 Functional Requirements 2](#_Toc90392545)

[2.3 Nonfunctional Requirements and Constraints 2](#_Toc90392546)

[3 Software Design 2](#_Toc90392547)

[3.1 Object Model 3](#_Toc90392548)

[4 Verification and Validation 3](#_Toc90392549)

[5 Software Quality Assurance Planning 3](#_Toc90392550)

[6 Project Metrics 4](#_Toc90392551)

[7 Design Patterns Used 4](#_Toc90392552)

[8 Possible or Actual Refactoring 4](#_Toc90392553)

[9 What was Accomplished 4](#_Toc90392554)

[10 Conclusion 4](#_Toc90392555)

[11 Appendix 1 – Screenshots 5](#_Toc90392556)

[12 Appendix 2 – Selected Code 6](#_Toc90392557)

[12.1 Google Connection 6](#_Toc90392558)

[12.2 Trello Connection 6](#_Toc90392559)

[13 Instructions – 7](#_Toc90392560)

# 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.

# Project Requirements

## 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

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 and name
      2. Filter by source connection

## Nonfunctional Requirements and Constraints

* 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 but 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 physical mirror

# Software Design

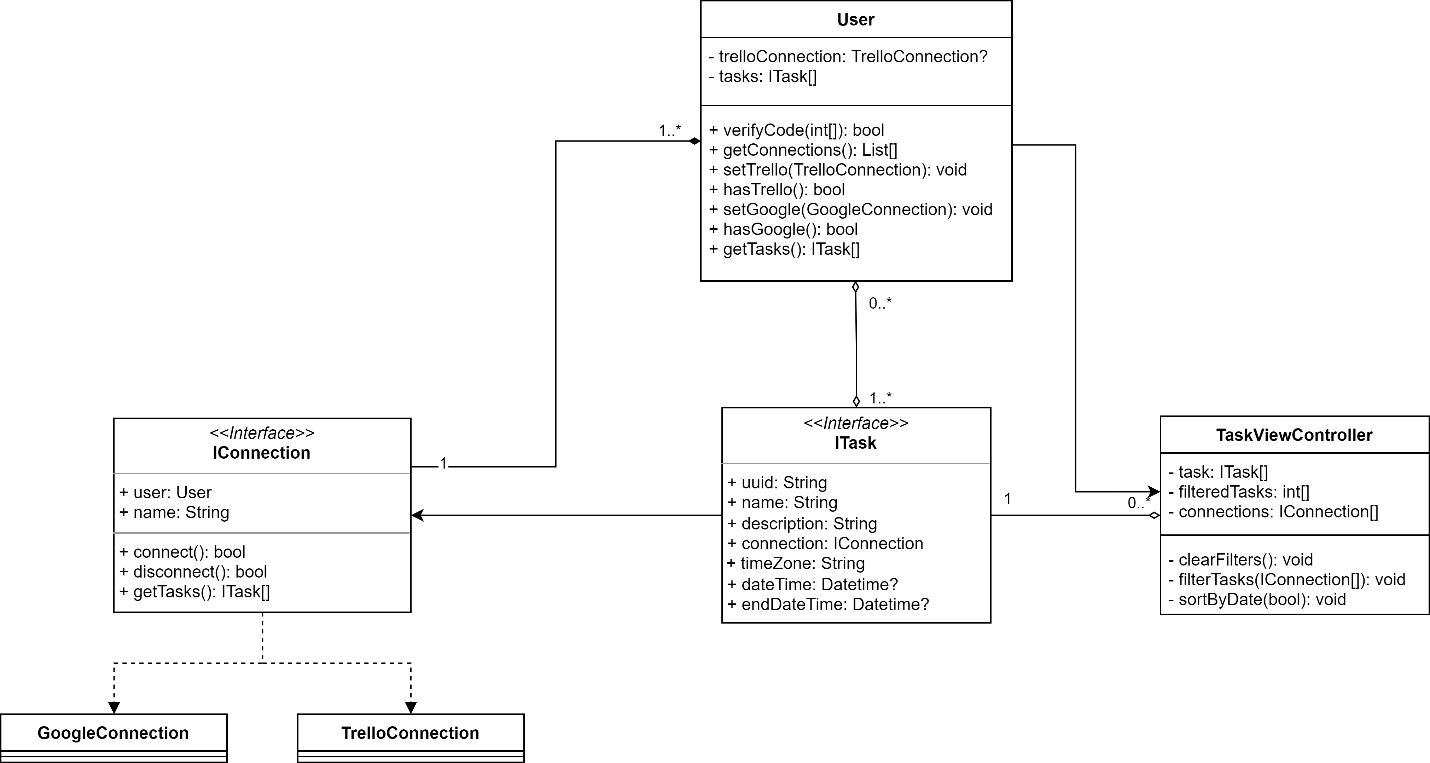
In the identification step, two human actors were discovered:

* **MirrorUser**: Possesses the mirror and has connected to other allowed task services
* **TaskCreator***:* Creates tasks on an external system which are then retrieved by the **Smart Mirror**

Additionally, there are two non-human actors:

* **SmartMirror:** Responsible for collecting and organizing the tasks through the use of various user-centered connections to external task services
* **ExternalTaskService:** Collects and stores tasks for retrieval by the **SmartMirror.** Not regulated as a part of this design document but can be accessed directly by the **SmartMirror**

## Object Model



# Verification and Validation

**Verification:** The system has been both with the customer in the form of code inspections, walkthroughs, and design reviews and with a suite of unit tests designed to test function requirements, non-functional requirements, and constraint adherence.

**Validation:** The system has been validated through integration tests in which the users interacted with the different system components and user interfaces both through self-guided interaction and by following a mutually prepared testing script.

# Software Quality Assurance Planning

The bulk of the software quality planning will consist of interviewing the users of the **Smart Mirror** in the future as the mirror receives more use and the users become more familiar with the system. At various future deadlines: minimally 3 months, 6 months, 1 year, 2 year, the users will have the option to revisit the steps outlined above in the Verification and Validation system to understand if the system is meeting their current needs. After this inspection the project will undergo an internal review and the developers will propose improvements and work together with the **Smart Mirror** customer to maintain quality and assure that the mirror is still helping to minimize the issues discovered in the Project Requirements section.

# Project Metrics

The primary metrics for this product, designed and developed by one employee are as follows:

* Person Months: 1 person months (0.75 for requirements and design, 0.25 for implementation)
* Lines of Code (LoC): 274 as per Rule 2: Each delimiter is one statement
* Function Point: 23.92

# Design Patterns Used

The primary design pattern used in this project was an adapter pattern around the connection classes, allowing for connections through Trello and Google to use the same externally facing functions of connect, disconnect, while allowing the two individual instances the ability to handle their own custom connection and authentication paradigms. Additionally, the individual connection classes are responsible for what they consider to be a valid task layout, reinterpreting the data passed back from the external site as internal Task objects. The use of this design pattern should make adding additional functionality to both tasks and connections trivial.

# Possible or Actual Refactoring

The main refactoring needed to make a functionally complete smart mirror is tying the mirror into the underlying operating system’s validation processes, ideally through a facial recognition system as minimizing user input is a desirable trait given the limited input nature of a wall-mounted “mirror”. Additionally, adding more connections to the Smart Mirror is ideal, but the connection interface implementation is already re-usable so adding additional connection types in the future should be trivial.

# What was Accomplished

This project successfully accomplished all the function and non-functional requirements outlined in the presentation above. The only item that was dropped from the initial software proposal was the integration of a TODY connection: this decision was made because TODY does not have any public-facing APIs, which is in direct violation of the requirements specified of the **ExternalTaskService.**

# Conclusion

In conclusion, I have used the unique skills that I have gained in this course to successfully plan, develop, and deploy a successful software engineering platform. I look forward to applying my software engineering skills in the future and continue to advance my knowledge by utilizing this coursework as a guide for future endeavors.

# Appendix 1 – Screenshots

Graphical user interface, text, application, chat or text message

Description automatically generated

Graphical user interface, application

Description automatically generated

# Appendix 2 – Selected Code

## Google Connection

@override  
Future<Map<String, Task>?> getTasks() async {  
 var calendars = await getCalendars();  
 if (calendars == null) return null;  
  
 Map<String, Task> tasks = {};  
 for (var calendarId in calendars.keys) {  
 var name = calendars[calendarId];  
 print('Pulling events from $name');  
  
 var httpClient = (await *\_googleSignIn*.authenticatedClient())!;  
 var calAPI = googleAPI.CalendarApi(httpClient);  
  
 String? pageToken;  
 do {  
 var events = await calAPI.events.list(  
 calendarId, pageToken: pageToken, timeMin: DateTime.now().toUtc());  
 if (events.items == null) continue;  
 events.items!.removeWhere((element) => element.status == 'cancelled');  
 tasks.addEntries(events.items!.map((e) {  
 var task = Task(  
 uuid: '${this.name}-${e.id!}', name: e.summary!,  
 description: e.description, connection: this,  
 dateTime: e.start!.dateTime ?? e.start!.date!,  
 endDateTime: e.end!.dateTime ?? e.end!.date);  
 return MapEntry(task.uuid, task);  
 }));  
 } while(pageToken != null);  
 }  
 return tasks;  
}

## Trello Connection

@override  
Future<Map<String, Task>?> getTasks() async {  
 var boards = await getBoards();  
 if (boards == null) return null;  
  
 Map<String, Task> tasks = {};  
 for (var boardId in boards.keys) {  
 var name = boards[boardId];  
 print('Pulling events from $name');  
  
 var url = Uri.*parse*('$*\_baseUrl*/boards/$boardId/cards${getParams()}');  
 var response = await http.get(url);  
  
 if (response.statusCode != 200) return null;  
  
 var body = json.decode(response.body);  
  
 for (var entry in body) {  
 var lastActivity = DateTime.*parse*(entry['dateLastActivity']);  
 var due = entry['due'];  
 var task = Task(  
 uuid: entry['id'], name: entry['name'],  
 description: entry['desc'], connection: this,  
 dateTime: due != null ? DateTime.*parse*(due)  
 : DateTime(lastActivity.year + 1, lastActivity.month, lastActivity.day)  
 );  
 tasks[task.uuid] = task;  
 }  
 }  
 return tasks;  
}

# Instructions –

Should only be one step: Spin up an http server at 8080, pointed at the directory with the index.html file.

If there are any issues, check that your firewall isn’t blocking responses on different points.