Google Calendar for MagicMirror

Architecture Document

Team 8

September 19, 2016

Ethan Gill
University of Kentucky
ethan.gill@uky.edu

Arthur Silveira
University of Kentucky
arthur.silveira@uky.edu

Ethan Smith
University of Kentucky
ethan.smith1@uky.edu

Matthew Turner
University of Kentucky
matthew.turner813@uky.edu

Customer: Bill Danhauer

Table of Contents

Architecture Diagram

Our program has three major layers: presentation, data, and external. The presentation layer involves presentation logic and the user interface. The user interface package handles the taking in information from the user and displaying the relevant data to them. The user interface on the exterior is a part of the MagicMirror source code, with some modification for our project's specifics. The user interface has a dependency upon the presentation logic package. The presentation logic package handles filtering the data based on information as received from the user, and then passing it to the user interface once the data is in a human readable form. Specifically, this package will handle formatting calendar data from both the Google Calendar and iCal Calendar formats.

The presentation layer itself has a dependency upon the data layer. The data layer consists of the data interpretation and data access packages. The data interpretation package handles compiling the data received from Google Calendar and iCal Calendars that were downloaded. This package will consolidate those calendars into one data structure such that the presentation logic package can format it into a human readable form. The data interpretation package depends upon the data access package. The data access package handles authentication for the user as well as downloading the calendar data from Google Calendars and iCal Calendars. This layer depends on the external layer for error handling.

Lastly, there is the external layer. This layer is the external data sources such as the Google Calendars and iCal Calendars services. This package handles the errors that might arise from authentication and downloading calendar information. Errors such as connectivity problems or improper credentials for authentication are handled in this package. As it handles the error handling, this is also the layer where waiting for connection occurs. This layer only depends on external APIs and code bases. Namely, it depends on the Google Calendars API and the MagicMirror source code which handles iCal Calendars interaction.

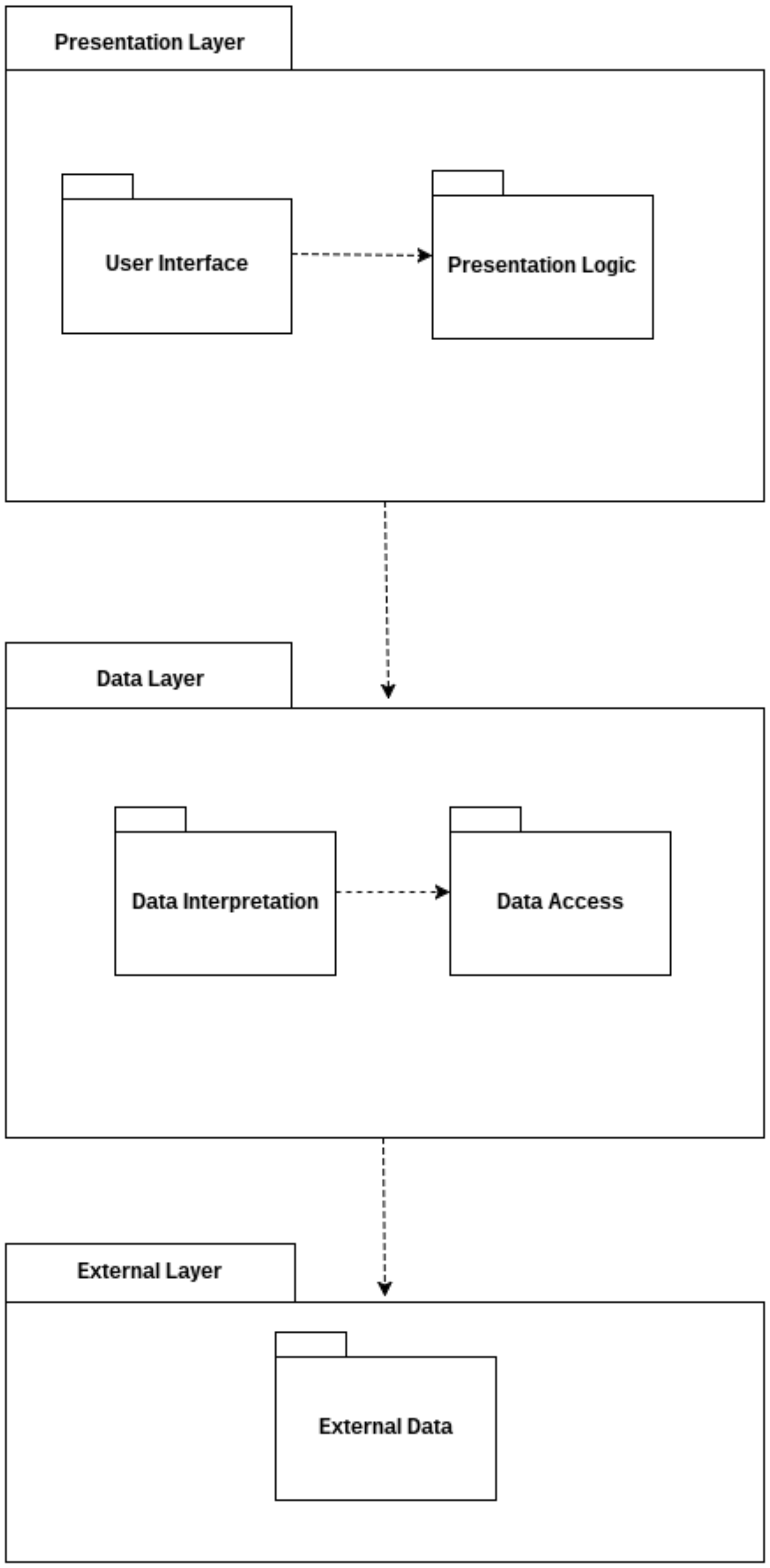


Figure 1: Project Architecture Diagram

Detailed Diagram

Class Diagrams

Most versions of JavaScript do not represent the concept of class inheritance, which is present in languages such as Java and C++. While JavaScript ES6 introduces inheritance, support for this new standard is slow to reach browsers and is currently only usable with special compilation steps, which are not viable for usage in the context of MagicMirror. JavaScript also does not support compile-time types, and as such it has no concept of an interface enforced by a compiler. Because of this, the classes in this project are mostly independent of each other; they only communicate with each other by calling static class methods.

Calendar Class

Our “Calendar” class acts as the main point of contact between calendar data downloaded from the internet and the application that displays this data. It handles the interpretation of the calendar configuration objects from the app configuration file. Once interpreted, the calendar class uses this configuration to expose display related settings, such as fade and animation speed. It also iterates between each individual configured calendar source and determines how to fetch the associated calendar type. Finally, it exposes a class method which the MagicMirror parent application can use to start fetching calendar data at an appropriate point in its application lifecycle.

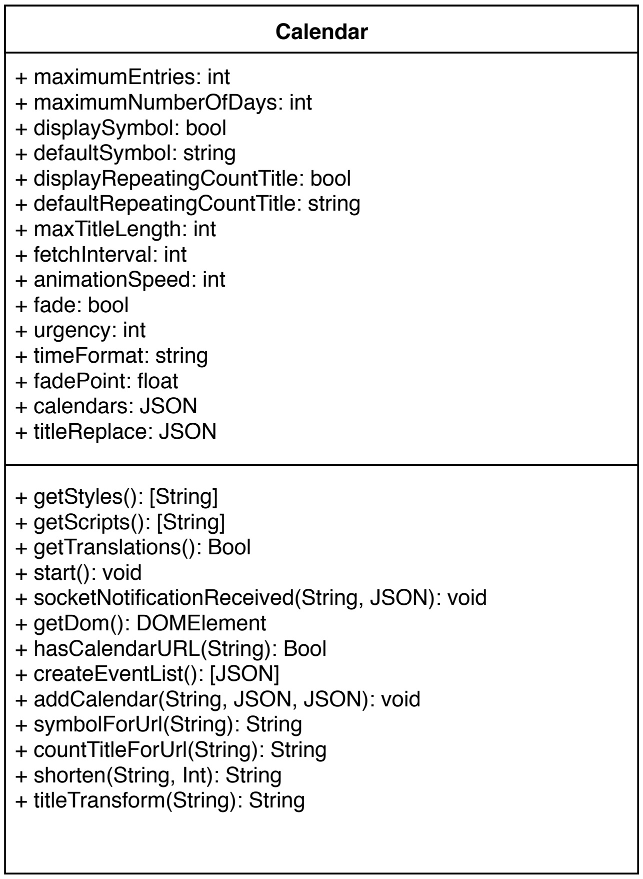

Calendar
+ maximumEntries: int
+ maximumNumberOfDays: int
+ displaySymbol: bool
+ defaultSymbol: string
+ displayRepeatingCountTitle: bool
+ defaultRepeatingCountTitle: string
+ maxTitleLength: int
+ fetchInterval: int
+ animationSpeed: int
+ fade: bool
+ urgency: int
+ timeFormat: string
+ fadePoint: float
+ calendars: JSON
+ titleReplace: JSON
+ getStyles(): [String]
+ getScripts(): [String]
+ getTranslations(): Bool
+ start(): void
+ socketNotificationReceived(String, JSON): void
+ getDom(): DOMElement
+ hasCalendarURL(String): Bool
+ createEventList(): [JSON]
+ addCalendar(String, JSON, JSON): void
+ symbolForUrl(String): String
+ countTitleForUrl(String): String
+ shorten(String, Int): String
+ titleTransform(String): String

Figure 2: “Calendar” Class Diagram

iCal Calendar Fetcher Class

The “iCalCalendarFetcher” Class handles the downloading and processing of iCal Calendar files. This is a URL to an iCal file that can be downloaded through an HTTP GET request. Once downloaded, each iCal file is parsed, and the fetcher builds a list of events from the resulting JSON. This list of events can then be accessed by the fetcher’s calling class by hooking into its event callbacks. The fetcher has two events: onError and onReceive. An onReceive event is triggered when the calendar has been successfully downloaded and parsed into a list of events. OnError is triggered if an error occurs somewhere along this pipeline.

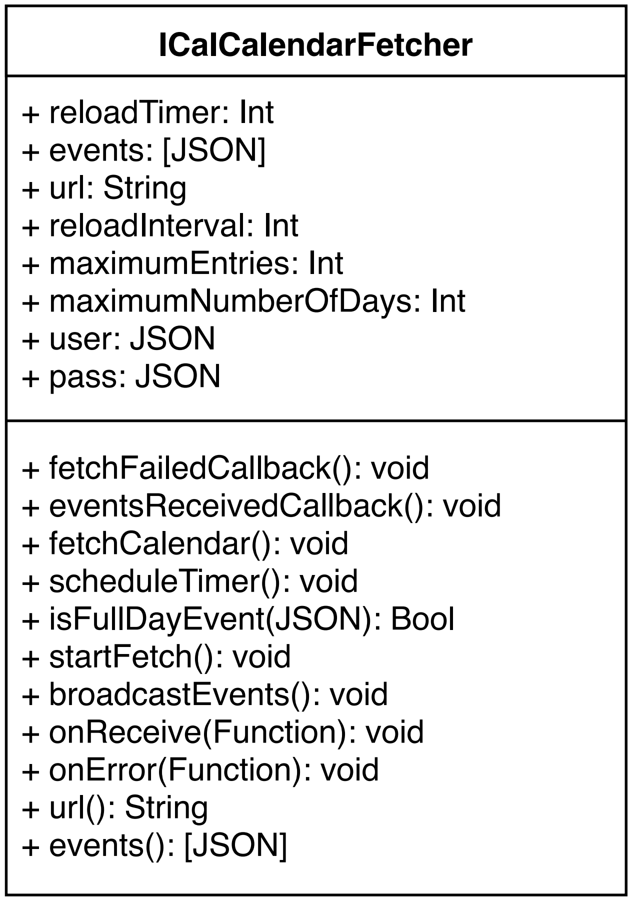


Figure 3: “iCalCalendarFetcher” Class Diagram

Google Calendar Fetcher Class

The GoogleCalendarFetcher requires a configuration object that encodes a valid client secret to access the Google Calendar API through OAuth. Using this configuration object, it attempts to authorize with the API. If successful, it downloads the necessary data from the Google Calendar API through its REST HTTP interface. This data is formatted as a JSON object that is then converted to a list of events. Other than these internal methods, the public API of this fetcher is similar to the iCalCalendarFetcher. It exposes two callbacks that its consumer hooks into to handles its result, onReceive and onError. OnReceived is called with a list of events if the process completes successfully. OnError is called with the error that occurred in the execution pipeline, if applicable.

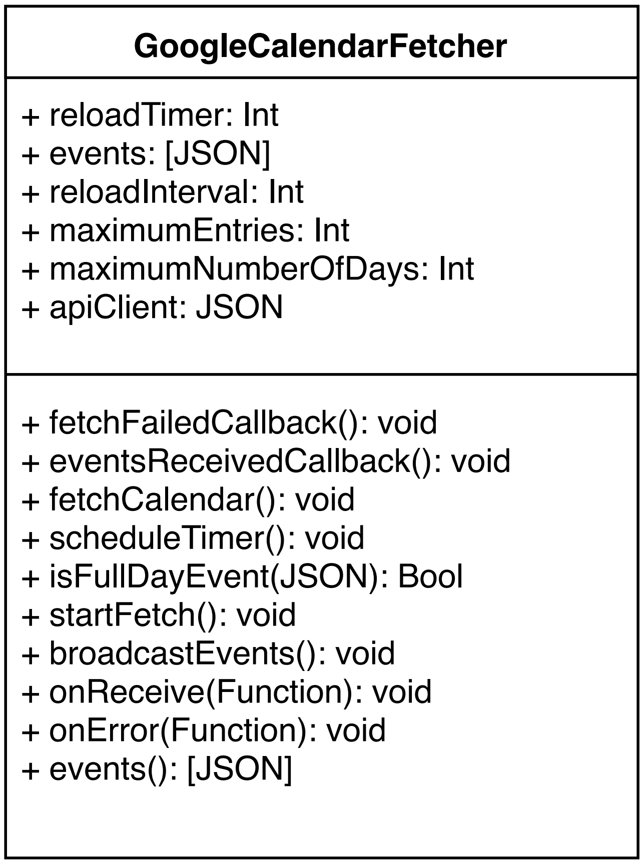


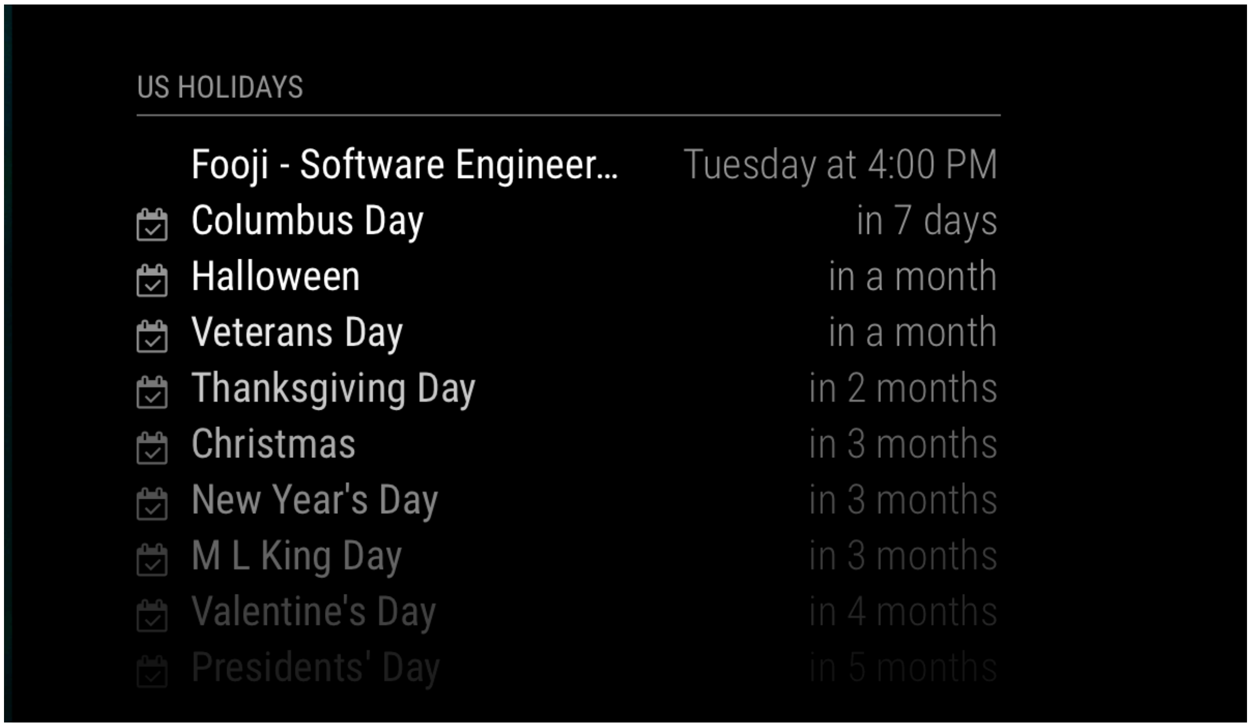
Figure 4: “GoogleCalendarFetcher” Class Diagram

User Interface Design (including Wireframes)

The MagicMirror UI is already well developed and opinionated in its stylings. The job of this project revolves around integrating with this UI so that the new Calendar interface appears synonymous with the current native look and feel. With this in mind, this project's UI focuses on the same principles that the greater MagicMirror community emphasize: chic, minimalism, and clarity. To achieve these goals, the code used to style MagicMirror's current UI elements was examined. Care was taken to ensure that the key features of this interface were incorporated into this project's code so new and current elements look as similar as possible.

Wireframes

Below is this project's wireframe, which is based on the MagicMirror UI. Notice the use of icons to distinguish different calendars from each other. Each calendar has its own configurable icon to be displayed to the left of its events. Individual calendars also make use of different colors of text to visually enforce the semantic difference between elements. For instance, the main title is in a larger font and is using a white text color; however, the less important labels, such as the date of each event, are presented with a grey color and a smaller font. A fade effect is gradually applied to dates as they get farther away from today until they are no longer visible.



US HOLIDAYS

Fooji - Software Engineer...	Tuesday at 4:00 PM
Columbus Day	in 7 days
Halloween	in a month
Veterans Day	in a month
Thanksgiving Day	in 2 months
Christmas	in 3 months
New Year's Day	in 3 months
M L King Day	in 3 months
Valentine's Day	in 4 months
Presidents' Day	in 5 months

Figure 5: MagicMirror Calendar Module Wireframe

Design Patterns

We implement two design patterns in our project, Model-View-Controller (MVC) and Singleton. Recognizing Object-Oriented design patterns is a slightly awkward in JavaScript, as the language does not have a concept of object classes. Additionally, JavaScript does not offer inheritance in the usual fashion. Instead, it makes use of prototype-based inheritance, which is not the traditional expectation when applying design patterns.

The project's calendar class is used as a singleton. It is instantiated from the calendar configuration file once at application startup; it is then maintained throughout the application lifetime. The singleton methods of the calendar class are all accessed through this static instance. The instance makes use of shorter lived fetcher class instances to coalesce calendar data and keep it in a central location so that the rest of the application can make use of it.

The application as a whole also uses the MVC pattern to organize the workflow the application follows. The calendar class' serves the View Controller role in this design pattern. The class is responsible for querying the iCal and Google Calendar Fetcher classes, which act as Models. The Calendar class then performs logic on the data, and passes the formatted results to the View for display. This design paradigm has the benefit of enforcing the clean decoupling of each section of code; the View Controller doesn't know how the View displays data, only what data it wants. Nor does the View Controller know how the data is retrieved or stored by the Model, just how to ask for it. The same barriers exist for the Model and View with respect to the View Controller, identifying this layout as the MVC design pattern.

Testing

Software testing plays a vital role in the developmental cycle of a project. Two main types of testing will be used to validate successful implementation of this MagicMirror project's specifications. First, End-to-End Testing will be performed from the role of an end user to verify that the system behaves correctly from start to finish for a standard use case. Additionally, functional testing requirements outlined in this section will be used to confirm that specific functions of the created Calendar module work correctly.

Unfortunately, the current MagicMirror project does not implement any automated testing whatsoever. Because of the inherent restrictions associated with contributing to an open-source repository, implementing automated testing for the Calendar module will prove a trying task. It will be possible to create a test data class for the user interface portion of the project, and testing the Google Calendar API code automatically may also be achievable. However, no suitable automated test options exist for the MagicMirror core project. Any code modified within MagicMirror itself will be tested manually using the functional testing requirements listed below.

Functional Testing

Functional tests are closely related to a project's functional requirements, often on a one-to-one basis. During this project, functional testing will be performed after each development cycle, or sprint. **Table 1** contains a list of the functional tests for the updated Calendar module of MagicMirror. Requirements (and their associated User Stories) will not be considered complete unless their applicable functional tests produce the expected outcome.

Table 1: Functional Test Cases for MagicMirror Calendar Module

Test Number	Configuration Parameters	Test Case	Success Conditions	Failure Conditions
1	• Client ID • Client Secret • Redirect URL	Credential authentication for a Google Calendar using Google Calendar API	Authentication is successful.	Authentication fails, and an error is displayed in the UI.
1b	• Client ID • Client Secret • Redirect URL	Invalid credential authentication for a Google Calendar using Google Calendar API	Authentication fails, and an error is displayed in the UI.	Authentication is successful.
2	• None	Authentication without any credentials in the configuration file	A note is displayed in the UI asking the user to provide credentials.	An empty calendar module is displayed with no informational message.
2b	• Client ID • Redirect URL	Authentication without client secret in the configuration file	A note is displayed in the UI asking the user to provide credentials.	An empty calendar module is displayed with no informational message, OR authentication is successful.
3	• Client ID • Client Secret • Redirect URL	OAuth Token generation using valid credentials	An OAuth Token is generated from provided valid configuration parameters.	No OAuth token is generated. An error is displayed in the UI.

Test Number	Configuration Parameters	Test Case	Success Conditions	Failure Conditions
3b	• Client ID • Client Secret • Redirect URL	OAuth Token generation using invalid credentials	No OAuth token is generated. An error is displayed in the UI.	An OAuth Token is generated from provided invalid configuration parameters.
4	• iCal URL	Downloading iCal Calendar	System behaves as it did before development began; iCal is downloaded	iCal is not downloaded OR there are substantial differences in behavior from the previous version.
5	• Client ID • Client Secret • Redirect URL	Initial download of Google Calendar	System stores full future calendar data from Google Calendar API.	System stores only partial future data from Google Calendar OR system doesn't store any calendar data
5b	• Client ID • Client Secret • Redirect URL	Network Failure during initial download	System displays network error in UI.	An empty calendar module is displayed with no informational message.
6	• Downloaded Google Calendar Data	JSON formatting of Google Calendar	Google Calendar events are converted into a JSON array/dictionary	Google Calendar events are not converted correctly OR only some events are converted
6b	• Downloaded iCal Calendar Data	JSON formatting of iCal Calendar	System behaves as it did before development began; iCal format is converted to JSON	iCal is not formatted OR there are substantial differences in formatting from the previous version.
7	• Converted JSON Calendar Data from iCal and/or Google Calendar	Chronological display of calendar data	Calendar data is displayed in chronological order	Calendar data is displayed out of order

Test Number	Configuration Parameters	Test Case	Success Conditions	Failure Conditions
8	• Converted JSON Calendar Data from iCal and/or Google Calendar	Multiple calendar displays	Multiple calendars are displayed with events in chronological order, regardless of originating calendar	Calendars are displayed sequentially instead of interweaved OR only one calendar is displayed
9	• Client ID • Client Secret • Redirect URL • Converted JSON Calendar Data from Google Calendar	Follow-up sync of Google Calendar	Google Calendar data is updated from API without re downloading entire calendar (only updated or new events are downloaded)	Entire Google Calendar is re downloaded on change, OR changes are not synced at all
10	• Client ID • Client Secret • Redirect URL • Converted JSON Calendar Data from Google Calendar	Updated Google Calendar events displayed on MagicMirror after sync	MagicMirror Calendar Module UI updates to display new/modified/deleted events in correct chronological location	Calendar Module does not update OR module updates but new event is in wrong chronological location

End-to-End Testing

End-to-End Testing ensures correct execution of a program for a general use-case. While not as narrow in scope as functional testing, it is equally important; a program that completes individual tasks admirably may not fare well when tasks are completed in quick succession, for instance.

The MagicMirror Calendar module has only one intended user type. For the purposes of this section, a user will be interchangeably referred to as a customer.

User/Customer

A MagicMirror user will begin by running MagicMirror for the first time. They will be presented with the MagicMirror user interface, with the Calendar module on-screen. Because they have not yet configured calendars to display, an informational message will be shown in place of upcoming events.

Next, the user will create or modify the calendar module configuration file. They will create one dictionary for an iCal Calendar and one dictionary for a Google Calendar. Inside the Google Calendar dictionary, they will provide their client identifier and client secret, along with a redirect URL for calendar updates. The user will then restart MagicMirror.

MagicMirror's Calendar module will parse the configuration file correctly, and will immediately begin to download iCal Calendar files. At the same time, it will attempt to authenticate with the Google Calendar API using the information provided within the configuration file. If a network or authentication error occurs with either process, an error message will be displayed in the Calendar module UI, but the other calendars will continue to be downloaded and processed.

Once all calendar data from both iCal and Google providers has been downloaded, the module will convert both types of calendar data into a single JSON structure. Unnecessary metadata will be dropped. At the end of this process, a JSON file will exist that contains all upcoming events from every valid calendar configured by the user. If a calendar cannot be parsed correctly, it will be ignored and an error message will be shown; other calendars will continue to be processed.

Finally, MagicMirror will display all upcoming events from the converted JSON object onscreen in the Calendar section. The user can add or delete events from any connected Google Calendars or iCal public calendars using other calendar software. After MagicMirror syncs the calendars again, they will display updated information.

The above end-to-end scenario will be repeated with different types and quantities of calendars to ensure wide test coverage.

Quality Assurance

With the quality assurance review in mind, a personalized checklist was devised for the product system. The checklist, found below, was used in confirming that the overall architecture and each section of the document met all the necessary specifications required by the assignment prompt. Should the any of the checkmarks be absent, the corresponding item will be further discussed in **Section 5** (Defects).

- ☑ Title Page
- ☑ Table of Contents
- ☑ **Section 1**
    - ☑ **High Level Diagram -** Architecture diagram or Package Diagram or Domain model diagram
        - ☑ Team looks over entire section and discusses content.
        - ☑ Team compares section to requirements
        - ☑ Team checks section for content clarity
        - ☑ Necessary changes are made.
        - ☑ Team agrees on finalized diagram.
- ☑ **Section 2**
    - ☑ **Detailed Design - Detailed class diagram**
        - ☑ Team looks over entire section and discusses content.
        - ☑ Team compares section to requirements
        - ☑ Team checks section for content clarity
        - ☑ Necessary changes are made.
        - ☑ Team agrees on finalized diagram.
- ☑ **Section 3**
    - ☑ **Testing -** Test cases for unit and/or integration testing
        - ☑ Team looks over entire section and discusses content.
        - ☑ Team compares section to requirements
        - ☑ Team checks section for content clarity
        - ☑ Necessary changes are made.
        - ☑ Team agrees on finalized testing section.

Metrics

Overall System Complexity

This system has no inheritance due the nature of Java Script and its lack of an object-oriented infrastructure. Therefore, coupling between object classes is also not applicable since the three classes being worked with work very much decoupled from each other.

Product Size

This product has 2 user stories. There are a total of 15 unit test cases, which can be found in **Table 1.** There are 3 classes with around 34 methods.
The current product size can further be estimated by using Product Effort, described in the next section, or by number of Story Points, as described in the **Project Plan** section of the **Requirements Specifications** document.

Product Effort

The estimated product effort in terms of hours per person for this project has been broken down in **Table 2**:

Table 2: Estimated Product Effort

Activity	Number of Hours
Environment Setup	10
Coding	80
Product Testing	20
Nonfunctional Requirement Implementation	20
Bug Fixes, Client Changes	30

Defects

As coding work on the project has not been started yet, no defects have been discovered. No relevant defects were found in **Sections 1-3** of this document.