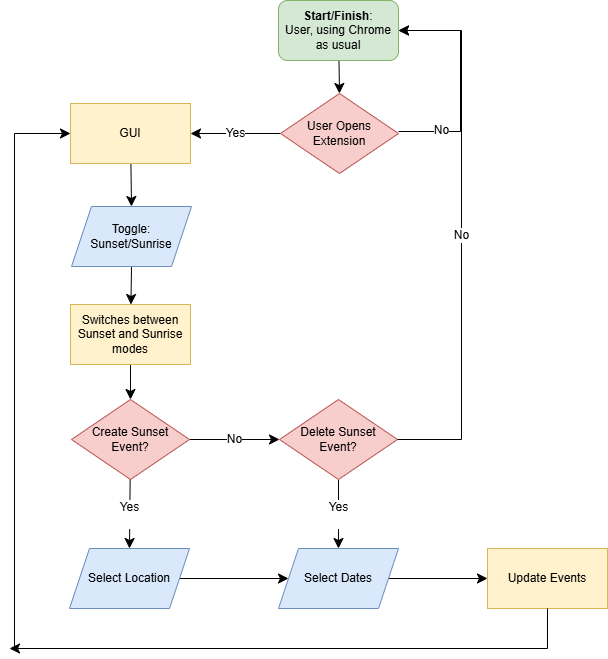
Sunset2Calendar

Sunset2Calendar is proposed to be a Chrome extension that allows users to add sunset and sunrise events to Google Calendar. Users can specify their location, base off geolocation, city, or address and Sunset2Calendar will add as many sunsets/sunrises as the user desires. This extension is designed for users who want to track sunset and sunrise times for personal, photography, or planning purposes. For the purposes of this development, an agile and iterative development style is being deployed. I propose to learn as I code throughout the development so numerous code changes and iterations are to be expected. This document is simply a documentation of my learning and development process.

# Planning

**Features:**

* Location Selection
  + Users can input their location in one of three ways
    - Coordinates
    - Address
    - Geolocation: Allowing the extension to access the user’s current location
* Date Selection
  + Users can specify the dates they want to add the sunsets/rises to by
    - Selecting dates on a calendar
    - Selecting months on a calendar
* Event customisation
  + Users should be able to choose a custom name of the sunset event
  + Users should be able to choose a colour

**User Flow:**

The user flow is simple, the user will be able to open the extension and select if they want to create or delete sunsets or sunrises, select the location and dates, then process the script. To visually display the user flow, a flowchart has been created.

This flowchart visualises user flow, starting and terminating in the same place. Although self-explanatory, this flowchart easily separates the program into two main components, the updating events processes, and the GUI.

**Technical considerations:**

Choosing a language and packages is an important step in development. Google offers easy python APIs for Google Calendar, which is vital within this project, however chrome extensions generally require HTML, CSS, and JavaScript front ends. Some considerations include:

* User Interface:
  + Should be intuitive and simple for a pop up extension using HTML and CSS
  + User input should be as limited as possible, using methods of input validation
* Google OAuth 2.0
  + Authentication per user, uses googles ‘google-auth’ and ‘google-auth-oauthlib’ libraries to handle OAuth 2.0 authentication for the Google Calendar API.
  + This also stores user credentials securely
* Token Expiry
  + Handles token expiration, possibly a requirement when dealing with specific user information like accessing calendars
* Error Handling
  + API errors may cause issue as these are new technologies I am working with and are not language specific, rather relying on googles developer portal
* Permissions
  + It’s highly likely that an extension that requires access to a user’s personal calendar and possibly location would have permission issues, which need to be researched and evaluated.
  + Location is not required for the functionality of the extension, however, google calendar is.
* Security and legality
  + Ensure any user files that could be accessed by the program are protected against, unlikely anything will appear as the nature of a chrome extension is browser side
  + If a backend is needed, every field should include XXS prevention
  + Data Collection: Clearly state what data is collected (e.g., location, calendar events) and how it is used.
  + User Consent: Obtain user consent before accessing their location or Google Calendar.

# Similar Products

The user interface is a vital part of the development of this program. Looking at similar products, other extensions, the most popular design is a little GUI pop up.

A screen shot of a computer

Description automatically generatedA screenshot of a qr code

Description automatically generatedSee TamperMonkey & UBlock:

Easy, simplistic buttons

Minimalistic design, few details besides only necessary information

Text hidden behind drop down boxes, to maintain simplicity

Both are simple designs, with buttons and minimal user inputs. Text is hidden behind drop downs to maintain the simplicity, and only neccesary information is kept on the page. This design is intentional as users do not want to have complex interfaces in google extensions – users do not want to open an extension and have the pop up be larger than it needs to be. These intentional design choices will be replicated in the final application.

To further visualise and explain the GUI concept, a mock up design has been created

A toggle to change between sunset and sunrise mode.

A screenshot of a computer

Description automatically generated

The remove button should remove all events labelled ‘sunrise’/’sunset’ events from the users calendar, searching for events and deleting them within the specified date ranges

The add button should add an event to each day selected in the calendar, with the name ‘Sunset’/’Sunrise’.

The search button should retrieve coordinates from a map API

Date selection box, the user should be able to select individual dates and drag across cells to perform a mass selection operation.

Current selected location – in final product this will only be one line, both lines are examples.

Address / Coordinates Input Box.

Auto geo location.

This button should act like a instructions button, hovering over it should provide instructions.

# Prototype Development

Given the flexible nature of this development methodology, an initial prototype for both the front-end and back-end will be developed prior to conducting further research and design. This approach was chosen due to my limited familiarity with the required Google APIs, as well as the need to solidify foundational skills in HTML, CSS, and JavaScript.

**Backend Prototype:**

To aid my limited knowledge of the Google APIs and higher level of knowledge in Python, the decision was made to use Python to create the first prototype. The aim of this first prototype is to learn the Google developer API features, and create a basic, yet functional, program. Python is a simplistic language and is easy to read. The first prototype uses five python functions to successfully add sunsets to my calendar, seen below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function** | **Purpose** | **Parameters** | **Outputs** | **Dependencies** |
| *authenticate\_google\_calendar()* | Authenticates the user with the Google Calendar API, using saved credentials from token.json or creating new ones with credentials.json. Returns a service object for interacting with the API. | *N/A* | service\_obj (Service Object) | google.oauth2.credentials, google\_auth\_oauthlib.flow, googleapiclient.discovery, |
| *get\_sunset\_time()* | Fetches the sunset time for a specific location and date using the "Sunrise-Sunset" API and returns it. | lat (Latitude float), lon (Longitude Float), date (Formatted Date String) | sunset\_time (Sunset Object) | sunrise-sunset.org API |
| *format\_sunset\_time()* | Converts the API's sunset time into the correct format (ISO 8601) required by Google Calendar. | sunset\_time (Sunset Object) | formatted\_date (Formatted Date String) | datetime |
| *add\_event\_to\_calendar()* | Creates a Google Calendar event for the sunset, setting the event to last one minute, and adds it to the user's calendar. | service (Built Service Object), sunset\_time (Sunset Object) | *N/A* | formatted\_sunset\_time |
| *run()* | Combines all functions to authenticate, fetch the sunset time, format it, and create a sunset event on the specified date. | day (Formatted Date String) | *N/A* | authenticate\_google\_calendar(), get\_sunset\_time, add\_event\_to\_calendar |

The successful operation of this first Python based prototype proves the use of the Google API, and the sunrise-sunset API. The success can be seen in the screenshot below:

A screenshot of a computer screen

Description automatically generated

As seen in the screenshot above, the sunset times have been added to my own personal Google Calendar, with accurate sunset times from February 2nd to February 8th. Each sunset time is different, according to the accurate sunset times of the Sunset-Sunrise API.

**Frontend Prototype:**

The initial front end prototype is seen below:

A screenshot of a phone

Description automatically generatedThis prototype is certain success, each individual component that does not require JavaScript is functional. With the inclusion of JavaScript, each button will have a unique and proper function.

The search button should be linked to the Google API to search for either location using the search, or through selecting the user’s current location.

The dates buttons should allow the user to select the date period, and the program should use this data to add/remove events.

The add/remove buttons should be the final trigger to run the code, data entry checks should be performed first to verify a location and dates have been inputted. ‘Add’ should add events, ‘Remove’ should remove events.

**Prototype Conclusions**

* Conclusion intro
* Backend
  + Pros
  + Cons
  + conclusion
* front end
  + pros
  + cons
  + conclusion
* moscow analysis for future iterations