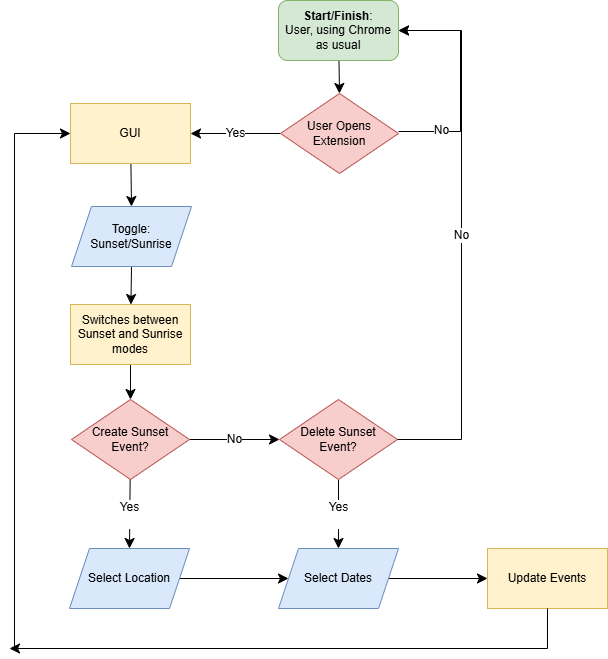
Sunset2Calendar

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:**

1. The user can add either sunrises or sunsets
2. Location Selection
   1. Users can input their location in one of three ways
      1. Coordinates
      2. Address
      3. Geolocation: Allowing the extension to access the user’s current location
3. Date Selection
   1. Users can specify the dates they want to add the sunsets/rises to by
      1. Selecting dates on a calendar
      2. Selecting months on a calendar
4. Event customisation
   1. Users should be able to choose a custom name of the sunset/rise event
   2. 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

After developing functional prototypes, this section focuses on drawing conclusions and evaluating the products. This section will focus on the front and back end features and that can be improved in future iterations.

**Backend Prototype**

The current backend prototype uses the python language and the Google Auth and Google Calendar APIs. This creates a working prototype which can add location dependent sunsets to a google calendar. Although successful this initial development is not acceptable for a final product as it does not meet many of the important requirements. The key features have been evaluated and explained on why or why not they have been met focusing on the back end.

|  |  |  |  |
| --- | --- | --- | --- |
| *Feature No.* | *Feature Description* | *Feature Met?* | *Explanation* |
| 1 | Users can add either sunsets or sunrises. | Partly | Sunsets can be added, not sunrises. |
| 2.1.1 | Users can input their location with an address. | No | Only coordinates can be inputted. |
| 2.1.2 | Users can input their location with coordinates. | Yes | Coordinates can be manually inputted in the backend. |
| 2.1.3 | Users can input their address with geolocation. | No | Only coordinates can be inputted. |
| 3.1.1 | Users can select dates on a calendar. | Partly | Dates can be selected specified the backend. |
| 3.1.2 | Users can select months on a calendar. | No | Feature not added. |
| 4.1 | Users can chose a custom event name. | No | Feature not added. |
| 5.1 | Users can chose a custom event colour. | No | Feature not added. |

In conclusion a portion of basic requirements have been met however the majority of features are partially or not met in its entirety. These features will be prioritised and focused on in the next iteration.

**Frontend Prototype**

The current iteration of the frontend prototype has been solely produced in HTML and CSS and therefore does not include any scripting, API interactions or calculations, however it does provide a good plan and idea for the final product. The prototype can be considered somewhat successful, however many features are currently absent. The following table shows how the main features are, or are not, met focusing entirely on the front end.

|  |  |  |  |
| --- | --- | --- | --- |
| *Feature No.* | *Feature Description* | *Feature Met?* | *Explanation* |
| 1 | Users can add either sunsets or sunrises. | Partly | The current prototype only has HTML and CSS for the sunsets |
| 2.1.1 | Users can input their location with an address. | Partly | An input box and appropriate form is available for the user to input an address |
| 2.1.2 | Users can input their location with coordinates. | Partly | An input box and appropriate form is available for the user to input coordinates |
| 2.1.3 | Users can input their address with geolocation. | No | No placeholder button or any JavaScript or API interactions have been included yet |
| 3.1.1 | Users can select dates on a calendar. | Yes | Users can select start and end dates |
| 3.1.2 | Users can select months on a calendar. | No | There is no current option for users to select full months |
| 4.1 | Users can chose a custom event name. | No | There is no current option for users edit event details |
| 5.1 | Users can chose a custom event colour. | No | There is no current option for users edit event details |

In conclusion a majority of features are not met, but key implementations of features have been started and given sufficient thought. Features have been partially met within the front end such as basic input boxes which can be worked upon to develop the front end into the final iteration.

# Iteration 1.0

After analysing and drawing conclusions from the initial front and back end prototypes, further and more in depth work can be completed to ensure the development of the desired programme. In this section more planning, analysis and design are undertaken. The initial documentation and development has been committed to help understand new ideas and potential analysis whilst also maintaining an active and iterative style to development.

After developing prototypes, a vast library of information and key ideas have been opened up, such as questions about the language usage and the final implementation of a Python based back end into a primarily JavaScript based Chrome extension. This question fundamentally changes the programming structure of the back end but may end up being more optimal for the final product, as the explored API’s (Google Auth, Google Calendar) also offer easy JavaScript implementation, and avoid the need to interpret languages from front end to back end using an extra framework such as Flask.

**Python or JavaScript?**

When evaluating the use of Python versus JavaScript for the development of the Sunset2Calendar Chrome extension, it is important to consider the strengths and limitations of each language within the context of the project’s requirements. Below is a detailed analysis of the pros and cons of using Python and JavaScript, based on the current development stage and the goals outlined in the documentation.

**Python**

**Pros:**

1. **Ease of Prototyping**: Python’s simplicity and readability make it an excellent choice for rapid prototyping. The initial backend prototype was successfully developed in Python, demonstrating its effectiveness for learning and implementing Google APIs, such as Google Calendar and OAuth 2.0.
2. **Strong API Support**: Python has robust libraries like google-auth, google-auth-oauthlib, and google API client, which simplify integration with Google services. This allowed for quick implementation of authentication and calendar event management in the prototype.
3. **Familiarity**: Given my higher level of familiarity with Python, it was easier to focus on understanding the Google APIs and core functionality without the added complexity of learning a new language during the initial development phase.

**Cons:**

1. **Browser Incompatibility**: Python is not natively supported in web browsers, making it unsuitable for direct use in a Chrome extension. Integrating Python with the front end would require additional frameworks like Flask or Django, adding unnecessary complexity and potential performance overhead.
2. **Limited Front-End Integration**: Python cannot directly interact with HTML and CSS, which are essential for building the extension’s user interface. This creates a disconnect between the front-end and back-end, requiring additional tools or languages to bridge the gap.
3. **Deployment Challenges**: Deploying a Python-based backend for a Chrome extension would require a server-side setup, which complicates the architecture and increases hosting costs. This is less ideal for a lightweight, client-side extension.

**JavaScript**

**Pros:**

1. **Native Browser Support**: JavaScript is the standard language for web development and is fully supported by browsers, making it the natural choice for building Chrome extensions. It allows for seamless integration between the front-end (HTML/CSS) and back-end logic.
2. **Efficient API Integration**: JavaScript offers native support for Google APIs, including OAuth 2.0 and Google Calendar, through libraries like google-auth-library and google APIs. This eliminates the need for language translation or additional frameworks, streamlining development.
3. **Asynchronous Capabilities**: JavaScript’s asynchronous programming model, using Promises and async/await, is well-suited for handling API requests and user interactions. This ensures a responsive and dynamic user experience, which is critical for a Chrome extension.
4. **Lightweight and Scalable**: JavaScript enables a client-side implementation, reducing the need for server-side infrastructure. This results in a lightweight and scalable extension that can run entirely within the browser.

**Cons:**

1. **Learning Curve**: While JavaScript is powerful, it may require additional time to learn, especially for developers more familiar with Python. Concepts like asynchronous programming and DOM manipulation may present initial challenges.
2. **Error Handling**: JavaScript’s dynamic typing can lead to runtime errors that are harder to debug compared to Python’s more explicit error handling. This may require additional attention during development to ensure robustness.
3. **Security Considerations**: As a client-side language, JavaScript exposes more of the code to the user, which could pose security risks if not properly managed. Careful implementation of security measures, such as input validation and XSS prevention, is essential.

While the initial prototype was successfully developed in Python, transitioning to JavaScript for the final implementation of the Sunset2Calendar Chrome extension is a more optimal and practical choice. JavaScript is the native language for web development and is inherently supported by browsers, making it the ideal choice for building Chrome extensions. Unlike Python, JavaScript seamlessly integrates with HTML and CSS, allowing for a cohesive and efficient development process without the need for additional frameworks or language interpretation layers. This native compatibility ensures smoother interactions between the front-end user interface and the back-end logic, reducing complexity and potential performance bottlenecks. Additionally, JavaScript’s extensive ecosystem, including libraries and APIs specifically designed for browser-based applications, provides robust support for features like OAuth 2.0 authentication, Google Calendar API integration, and geolocation services. By leveraging JavaScript, the extension can maintain a lightweight and responsive architecture, ensuring a better user experience and easier maintenance. Furthermore, JavaScript’s asynchronous capabilities, such as Promises and async/await, are well-suited for handling API requests and user interactions, which are critical for a dynamic and responsive extension like Sunset2Calendar. Ultimately, JavaScript aligns more closely with the requirements of a Chrome extension, offering greater efficiency, compatibility, and scalability for the final product.

**Revised key features**

**Backend features**

**Frontend Features**

**Iteration 1.0 Moscow Analysis**

**Iteration 1.0 Development**

**Iteration 1.0 Testing**

**Iteration 1.0 Conclusion**