**Technical write up**

**There are two APIs used in my project:**

1. NASA open APIs
   * This API provides beautiful imagery of astronomical objects. Every object has amazing stories behind it. Some objects also have YouTube video links provided.
   * NASA open APIs allows applications to query for APOD( [Astronomy Picture of the Day](http://apod.nasa.gov/apod/astropix.html)), which is what this project is based on.
2. Google drive API
   * Google drive API provides interfaces for storing files (in this project, images)
   * Google drive API uses OAuth2 for authenticating users

**Use case:**

* This project uses Three-Legged OAuth when application needs access to user’s google drive account. Users will be redirected to google drive authentication page to grant privileges to the application before they are served with any contents.
* After users grant their permissions to the application, a page with a date input field and search button will be rendered. Users can enter any date (e.g. 2012-10-10) in a format of Year-Month-Day and then hit the search date button. An astronomical object that is related to that date will show up in the page along with its story and image. If Users didn’t enter any date, it will default to current date.
* The image related to the search result will automatically be saved to user’s google drive account. There will be a link that takes users to view the saved image.
* There will also be a convenient link for revoking privileges.

**The authentication process:**

1. NASA open APIs
   * The authentication process for NASA open APIs is simple. Application owner first registers an api-key on NASA website. Every time the application queries from NASA open APIs, it must append this api-key to the end of query string.
   * The api-key will not expire or change unless application owner changes it.
2. Google drive API
   * The authentication process for Google drive API involves more steps than NASA open APIs
     1. Application owner registers a client ID and client secret from Google API website. A redirecting URI is set by application owner at this stage.
     2. When a user visits the site, the application asks Google drive API for protected access to client’s google drive account. The user then will be redirected to a page that asks him/her to grant permissions to this application.
     3. If the user approves it, he/she will be redirected to a dedicated URL initially set up by application owner at stage 1. An authorization code is included in the query string of the redirecting URL.
     4. Application grabs the authorization code and queries Google Drive API token endpoint for exchanging access token to this particular user.
     5. After successfully getting access token, the application must append the access token at the end of every request for accessing user’s protected data.

**Data Structure:**

* Similar to Wunderlist application, a data structure is maintained to store state and task pairs. This application uses a hash map data structure to do so. When users hit the search button, a task object with format {type: “task type”, data: “query data(date)”} is saved to a global variable called state\_tasks. We save this information because those data will be lost once users get redirected to Google Drive authentication page.

**Cache:**

* The application caches the api-key for NASA open APIs in a file in “./auth” directory
* The application caches the image downloaded from Open NASA APIs.

**Sequence Diagram of interactions:**

* Checkout sequence-diagram-1.png + sequence-diagram-2.png

**Problems & Technical Limitations & API Limitations:**

* When a chrome browser has multiple users signed in, every search will redirect users to authrization page.
* The initial plan was to save both image and explanation text together into a google drive doc file. Google provides npm libraries for doing this. However, google drive API documentation didn’t provide details for doing this in plain JavaScript (how to do plain http request). Without the google api npm libraries, I didn’t figure out how to save content to a doc file.
* I didn’t figure out how to cache access token after users granted their permission. In fact, I was not sure if it is a good idea to cache access token. The challenge was that I didn’t know how to identify the user to whom a specific access token belongs.
* NASA open APIs only allows 30 requests per hour and 50 requests per day.

**Areas of improvement:**

1. Cache

* In this application, we only cache images from NASA APIs. We could’ve also cached entire search results from NASA APIs (including the story, title and explanation of the image).
* We can also cache the access token that we get from google authentication server.

1. Avoid duplicate
   * We should avoid saving same contents twice to the user’s google drive account