CS 307 Team 34

Team Members: Clayton Lewis, Austin Lowell, Samuel Roberts, Brian Wong

Project Name: OverDrive

Design Document

Purpose

Overview

Multitudes of people use Google Drive, from individuals who want an easy way to access their files from multiple devices, to big organizations who need to have an efficient way to share important documents with their staff members. However, in situations where the free version would prove sufficient for the former audience, these big organizations often find it lackluster, missing key features that would make life much simpler for them. Things like managing permissions en masse, removing groups of people from a file, or changing someone's access to multiple files become very tedious when you're dealing with the big numbers organizations can come to possess. Google's solution for this, G Suite, can get quite pricey, and for organizations like the ones students run, this might simply not be feasible.

Our goal with this project is to create an accessible alternative for this last group, by providing them with a platform that makes these kinds of big-scale management operations much easier and much more intuitive than with the base Google Drive UI.

Requirements

Functional Requirements

1. As a user, I would like to install the OverDrive extension in Google Chrome.
2. As a user, I would like to install the OverDrive extension in Mozilla Firefox (if time allows)
3. As a user, I would like to be able to view my entire directory structure at once
4. As a user, I would like to visualize file permissions in that structure.
5. As a user, I would like to be able to see the number of files and subfolders in a folder
6. As a user, I would like to easily add a person to my files and folders.
7. As a user, I would like to easily remove a person from my files and folders.
8. As a user, I would like to easily change another user's permissions for a file or folder.
9. As a user, I would like to easily add multiple people to a file or folder at once.
10. As a user, I would like to easily remove multiple people from a file or folder at once.
11. As a user, I would like to easily change multiple people's permissions for a file or folder at once.
12. As a user, I would like to easily add a person to multiple files and folders at once.
13. As a user, I would like to easily remove a person from multiple files and folders at once.
14. As a user, I would like to easily change permissions of a person for multiple files and folders at once.

15. As a user, I would like to easily add multiple people to multiple files and folders at once.
16. As a user, I would like to easily remove multiple people from multiple files and folders at once.
17. As a user, I would like to easily change permissions for multiple people for multiple files and folders at once.
18. As a user, I would like to easily give ownership of a file or folder to another user.
19. As a user, I would like to easily give ownership of multiple files or folders to another user.
20. As a user, I would like to easily request ownership of a file or folder from other people.
21. As a user, I would like to easily request ownership of multiple files and folders from other people.
22. As a user, I would like to select whether editors can change permissions of a file or folder
23. As a user, I would like to select whether editors can change permissions of multiple files and folders at once.
24. As a user, I would like to choose which subfolders and files are affected by changes to a folder.
25. As a user, I would like a tutorial of the features the extension offers.
26. As a user, I would like to create groups of people with whom I can easily share files .
27. As a user, I would like to be able to change the color scheme of the extension (if time allows).
28. As a user, I would like to be able to give user feedback.
29. As a developer, I would like to receive user feedback.

Non-Functional Requirements

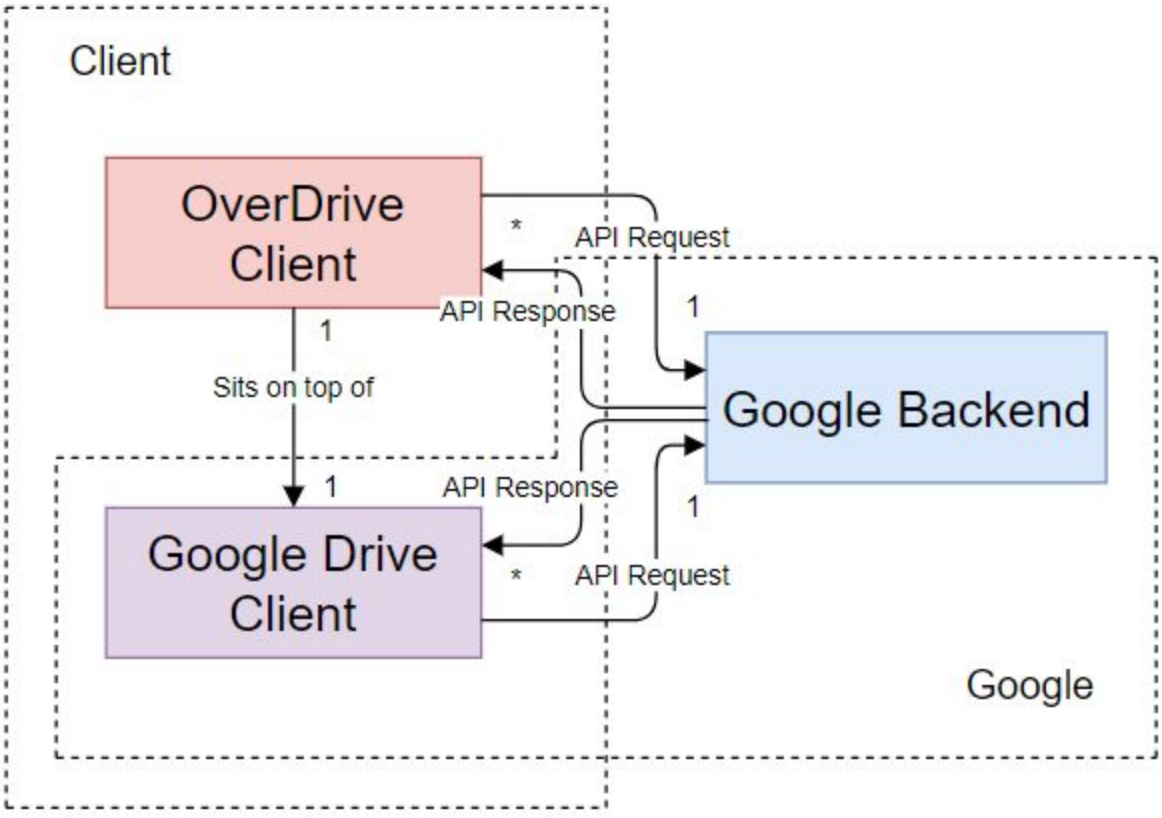
1. Performance: The user should not experience any noticeable lag (less than 500ms slower than if they had done the operation without the extension)
2. Usability: The interface should be intuitive, and someone not familiar with Google Drive should be able to figure it out quickly
3. Security: The extension should not affect the contents of any file, nor should it delete or create files or folders
4. Security: The extension should not collect any personal data from the user account or the contents of the user's drive
5. Security: The extension should use Google's login interface
6. Portability: Users should be able to download and install the extension in Windows, Mac, and Linux operating systems
7. Interoperability: Users should be able to run the extension on Chrome and Firefox (if time allows)

8. Scalability: The extension should support changing up to 500 files and managing the permissions of up to 50 users at once.

Design Outline

Overview of Existing Architecture

Our project functions as an extension to Google's existing drive architecture, which already functions using a client-server architecture. Google's existing Drive application consists of a website accessed via a web browser that communicates with Google's servers. Google's backend consists of a web server as well as some sort of database to store information. Google's web client accesses this information via Google API requests to the server, and receives data via API responses.



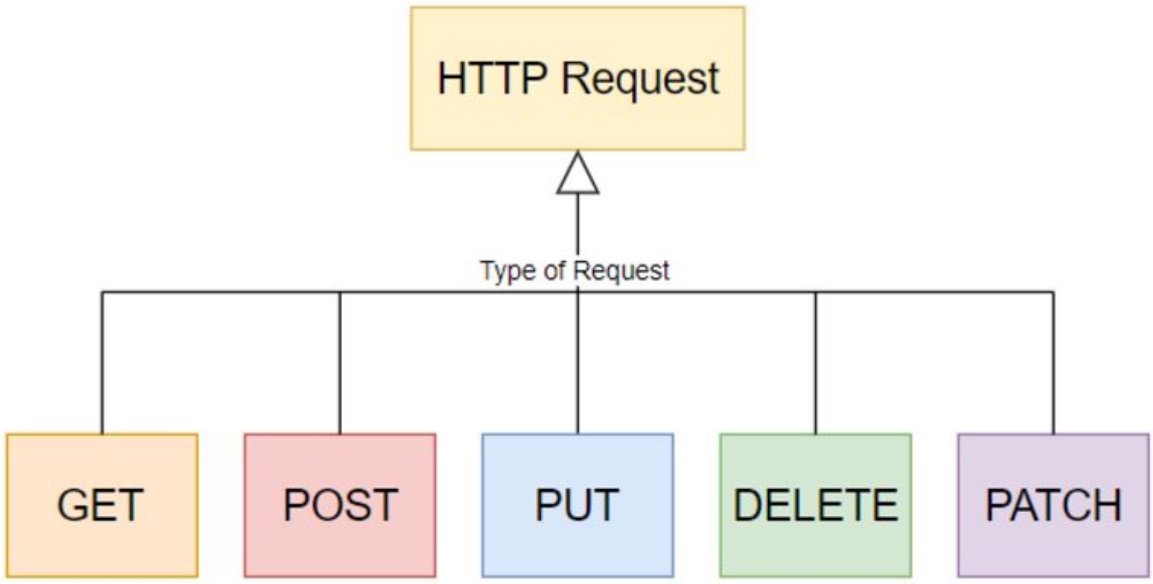
Overview of Our Project

Our project can be thought of as using a "modified-client-server model." The part we are creating consists of a web extension (can be thought of as a modified client) that sits on top of Google's existing Drive client. The client includes a UI that is built into the default Drive UI, as well as all of the functions necessary to add the functionality we want to Drive. Our client

communicates with Google's Backend in the same way as Google's client does (through API requests and responses), and does not interfere with any of the functionality of Google's client.

API Calls

All functions that are written by the client need to interface with the backend in some way. As mentioned earlier, this is done via Google's public Drive API. All API calls are made using HTML requests (executed through Javascript). Depending on the operation, the type of HTML request used will differ.



GET Requests:

- Used to receive information from the server
- Will be used to populate the UI with information

POST Requests:

- Used for creation of files and folders
- Used for moving of files and folders
- Used to start watching for changes to files and folders
- Used to set or update file or folder properties
- Will be used mostly for property updating purposes (the moving and creation of files is not something our application supports)

PUT Requests:

- Used to update information about files and folders
- Will be used to update permissions, properties, and metadata

DELETE Requests:

- Used to delete files, folders, properties, and permissions
- Will be used mostly for the deletion of permissions (the deletion of files is not something our application supports)

PATCH Requests:

- Updates properties, metadata, and permissions
- Allows for partial updates
- Will be used to update properties, metadata, and permissions without having to send the entire list of properties

Design Issues

Functional

1. What algorithm are we going to use to search for specific folders/files?
    a. **Depth-first search**
    b. Breadth-first search
    c. Iterative
    d. **Recursive**

    We decided that a recursive DFS approach would be the most optimal solution for this problem. We chose recursive because it'd make the code more visually easy to interpret, and we value this over the inherent difficulty in coming up with a recursive algorithm. We chose DFS because in the case of checked folders that we're not supposed to access, this approach would make it easier to simply stop going down a specific branch the moment we run into one of these folders, as opposed to BFS where we'd have to check for the checks every step of the way.

2. Do we plan on storing any data locally? If so, what and how?
    a. **Yes. We would have individual structures for files and folders, as well as for users to facilitate group operations.**
    b. No. We can just write a series of functions that pull the files from the Drive API and immediately populate the GUI, designing every other action to interact with the GUI and the API accordingly.

    We decided the simplest and most intuitive approach was storing the necessary information for the correct representation of the user's files in local storage and updating it accordingly whenever changes were made or detected. In addition to making the code more readable and easily comprehensible, this solution also makes debugging require less unneeded manipulation of said code, and reduces the number of API calls.

3. How will we know when to update our tree when files are changed through the Google Drive interface (as opposed to through OverDrive)?

a. Replace all the normal Google Drive buttons with custom OverDrive buttons to make this an impossibility.
b. Intercept every request the user makes and account for them in our OverDrive file display.
c. Send "get" requests to the API at set time intervals to maintain a file representation that's as accurate as possible.

We chose the second solution for a number of reasons. The first one seems unnecessary and inefficient, and defects could lead to unintended deletion or creation of files. The third solution might represent an additional tax on the performance of our system, and has the added disadvantage of having periods of time where the system is out of date. The second solution analyzes the sent request and the response from the API, so that we know what changed. This makes it significantly easier and less taxing to update our file display since we only have to update the parts of it that changed.

4. How should we display the folder/file structure graphically to the user?
    a. With a flowchart-like graphic showing the folders and the files they contain
    b. In a list with smaller icons, giving each folder an expand/collapse button to show/hide its contents, much like usual file browsers

    We believe using a familiar, condensed graphic representation is much more effective than something bigger and potentially more screen-hoarding. This implementation also gives us an easy way of providing more scalability, since a flowchart with hundreds or thousands of folders and files would get very unreadable very quickly.

Non-functional:

1. What type of software best fits our intentions with this project?
    a. Web app
    b. Desktop app
    c. Browser extension
    d. Mobile app

    Google Drive is already used in browsers, which means that making a browser extension would be by far the most accessible approach. We wouldn't have to worry much about installation, OS compatibility, machine specifications, and other things of this nature. It also makes it very easy for users to install it, since extensions can be easily installed with one click of the mouse, and ready to use in seconds.

2. What browser should we build the extension for initially?
    a. Mozilla Firefox
    b. Safari
    c. Internet Explorer

d. Google Chrome

The majority of our team members are more familiar with Chrome than with any of the other browsers, and given our extension is for a Google service too, it makes the most sense to start with Google's browser for this project. If time allows, we will look into expanding into other popular browsers, such as Firefox and Safari, and then perhaps Internet Explorer.

3. What language should we code this in?
    a. Python
    b. JavaScript
    c. TypeScript
    d. Other

    Since our intention is for OverDrive to be a Google Chrome extension initially, and these are written in JavaScript (also utilizing HTML and CSS), this is our coding language of choice. Our team also has a decent familiarity with the language, so we feel prepared to tackle whatever obstacles this decision might bring about.

Design Details

Description of classes (resources) used from Google Drive API:

Our implementation does not require or lend itself to the creation of many classes. The data handled will be organized in objects that act as classes and contain many fields. The following is a list of the relevant resources and a description of how they will be used in the extension. The class diagram includes only those fields which are relevant to this project. Find the full list of resources here. We will create one class of our own, which is the Group class.

File: Each File represents a file that is stored in Google Drive and to which the current User has access. A File contains data concerning its permissions, capabilities, and owner(s). Folders are also treated as files. Every File has:

- An ID
- A name
- A Permissions object which details the permissions for the current User
- A list of Permissions for Users with access to this File
- A Capabilities object which details what the User can and cannot do with the File
- An object pointing to the User that shared the File with the current User
- A list of all Users that are owners of the File
- A link to its parent folder

Permissions: The Permissions class contains the primary role of this user (organizer, owner, reader, or writer) and a list of additional roles (commenter).

Capabilities: Each Capabilities object is associated with a File. It contains several booleans which specify whether a user can perform certain actions on the given File. These are mostly determined automatically based on the User's Permissions.

User: This class represents a Google Drive user. Every user has a display name, email address and Picture.

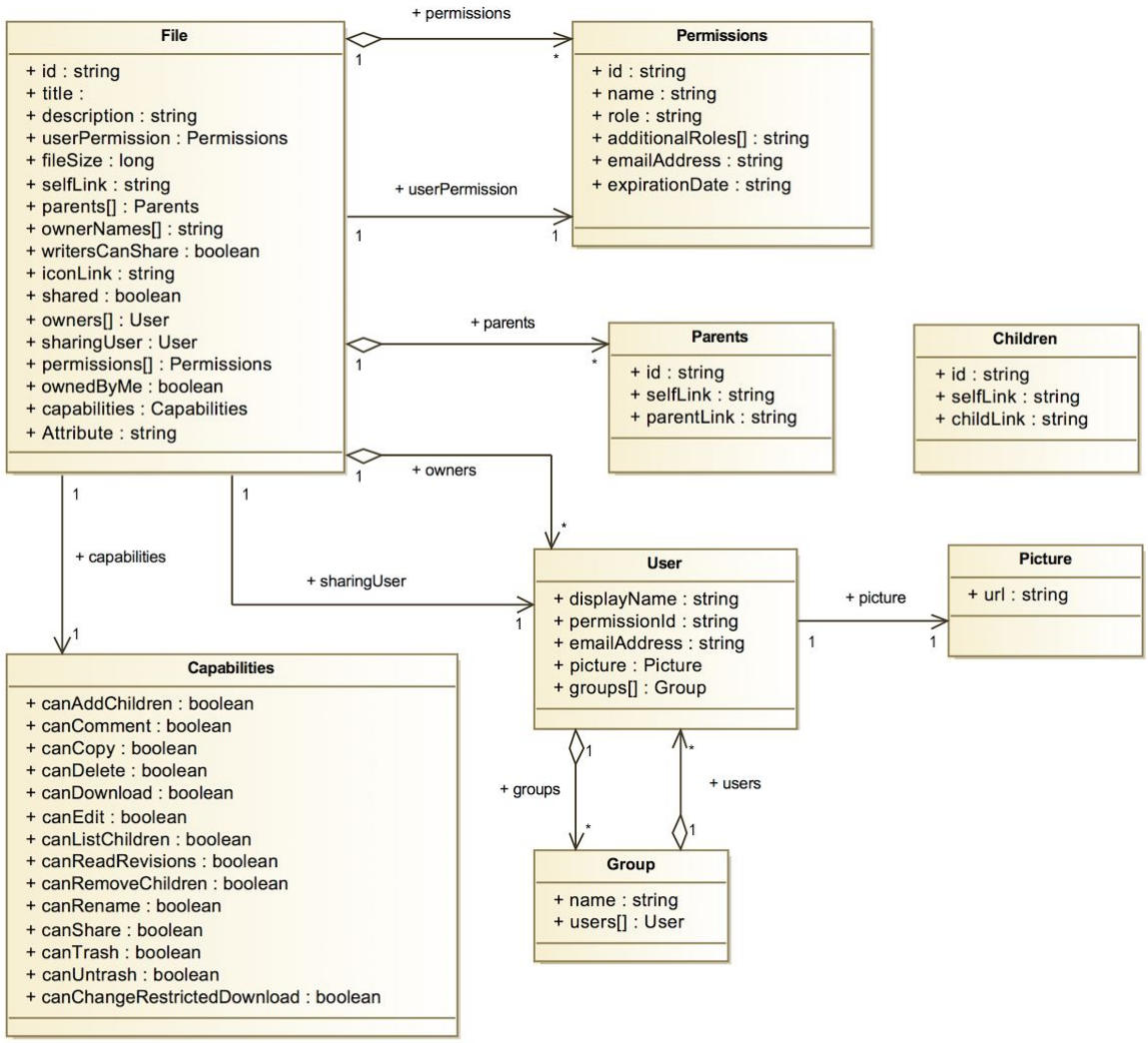
Picture: A User's Picture contains a url linking to the User's profile picture.

Parents: A Parents object is a reference to a File's parent folder.

Children: A Children object is a reference to a folder's child. Note: While this class is not associated with any others here, it will be used in calls to certain methods in the API.

Group: A group contains a list of users and a name. A user can create a group in order to more easily share files with the group members more than once.

Class Diagram



Description of methods

addToFile- Takes the parameters of a list of users, a role, a file ID. A list of all users with access to the file will be obtained from the owners list, and the user specified will be added on to the list with the given role. The users will only have access to the file specified, and will not be able to access any other file in the folder. This function will also be used to change permissions.

addToFolder- Takes the parameters of a list of users, a role, and file ID. A list of all users with access to the folder will be obtained from the owners list, and the users specified will be added on to the list with the given role. The users will only have access to the folder specified and its selected contents, and will not be able to access any other folders which contain this folder. This function will also be used to change permissions.

removeFromFile- Takes the parameters of a list of users and file ID. A list of all users with access to the file will be obtained from the owners list, and the users specified will be removed from the list. If the users had access to other files, they will keep all permissions for those files besides the one they are removed from.

removeFromFolder- Takes the parameters of a list of users and file ID. A list of all users with access to the folder will be obtained from the owners list, and the users specified will be removed from the list. If the user had access to other folders which contain the one specified for removal, they will still keep all permissions for those folders and its contents besides the one they are removed from.

addOwners- Takes the parameters of a list of users, file ID, and specified permission to give. A list owners with access to the file will obtained, and a list of permissions will be checked. The user specified will receive the permissions specified in the parameters. If the file is contained in a folder with other files and folders, the user will only have the specified permissions of that file.

removeOwners- Takes the parameters of user, file ID, and specified permission to remove . A list owners with access to the file will obtained, and a list of permissions will be checked. The user specified will lose the permissions specified in the parameters.. If the file is contained in a folder with other files and folders, the user will still have the specified permissions of those files, just not the file specified.

displayTree- Displays a tree of all folders and files associated with it. The tree will initially be be expanded meaning all folders will show all of it's subfolders and files inside of it.

collapseFolder - Takes in the parameter of fileID. If the folder contains subfiles that are expanded, the folder will collapse and hide such files in a file tree.

expandFolder - Takes in the parameter of fileID. If the folder contains subfiles that are not expanded, the folder will expand and show the files in a file tree.

checkForChanges - Catches outgoing requests and incoming responses, and if necessary, updates the tree data structure and calls displayTree.

createGroup - Takes in the parameter of a group name. A new empty group will be created with this name.

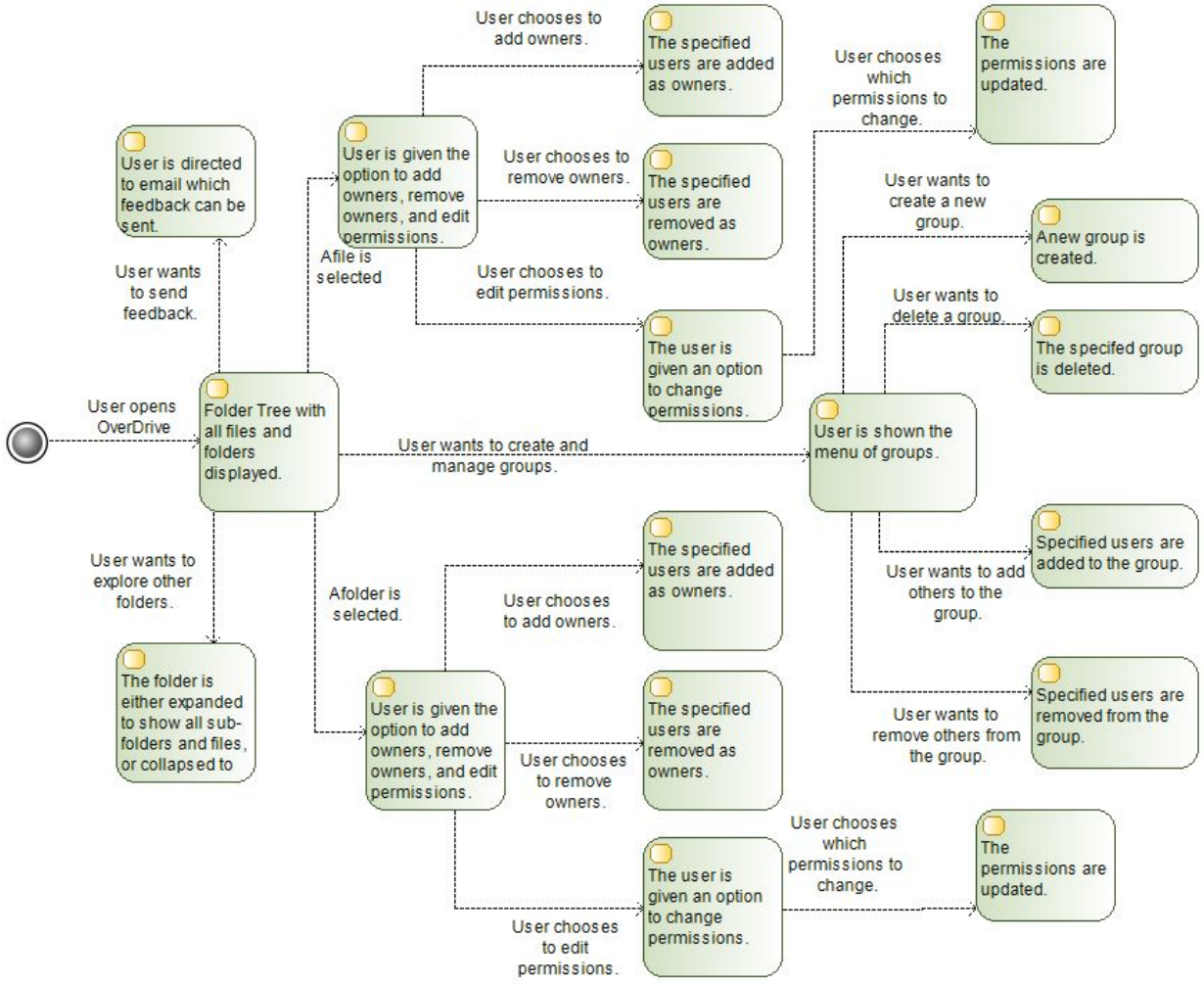
removeGroup - Takes in the parameter of the group name. The user's group with this name will be deleted. If said group does not exist, nothing happens.

addToGroup - Takes in the parameters of the group name and user ID. The specified user will be added to the group.

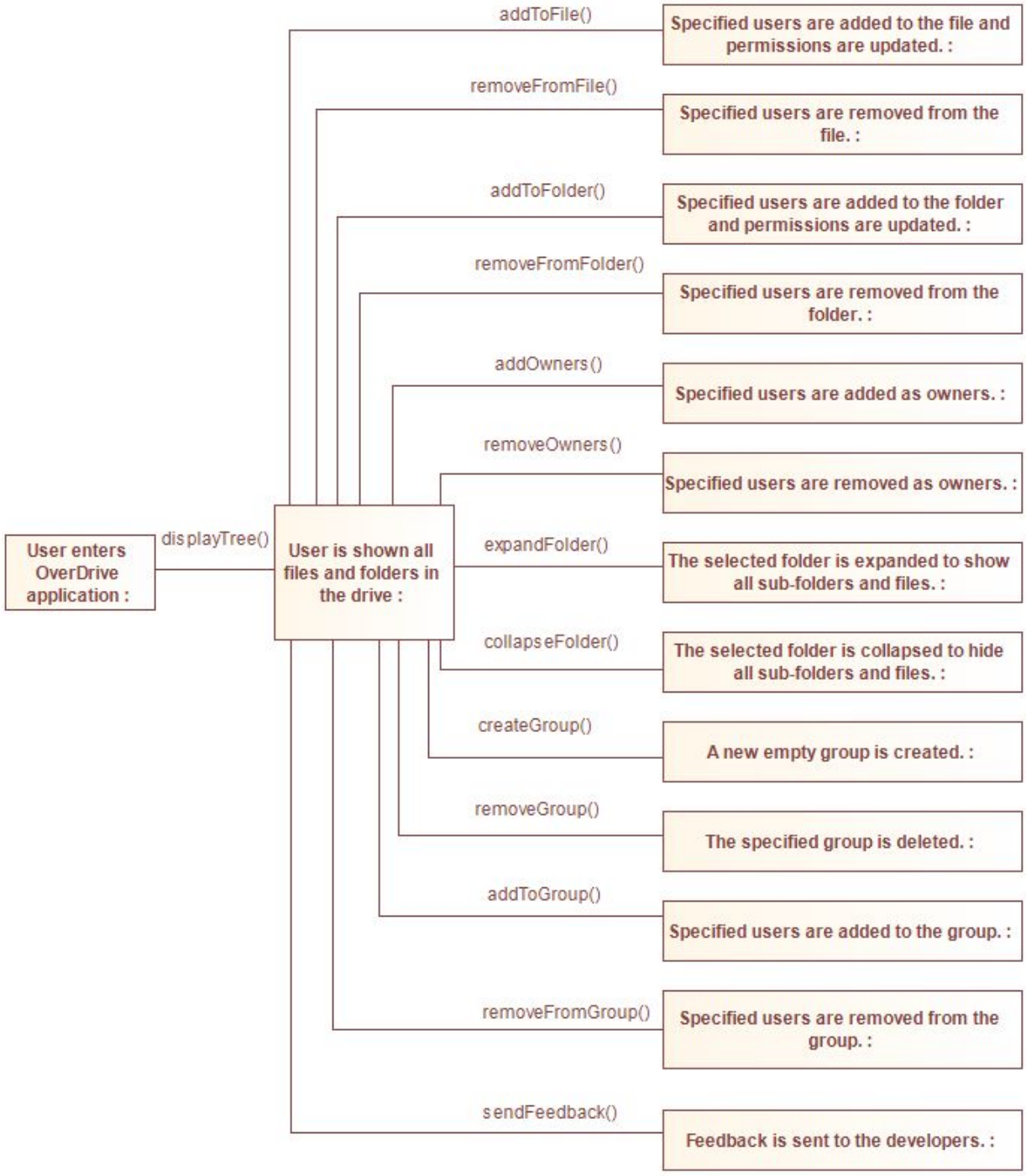
removeFromGroup - Takes in the parameters of the group name and user ID. The specified user will be removed from the group.

sendFeedback - Sends an email to the developers with any feedback

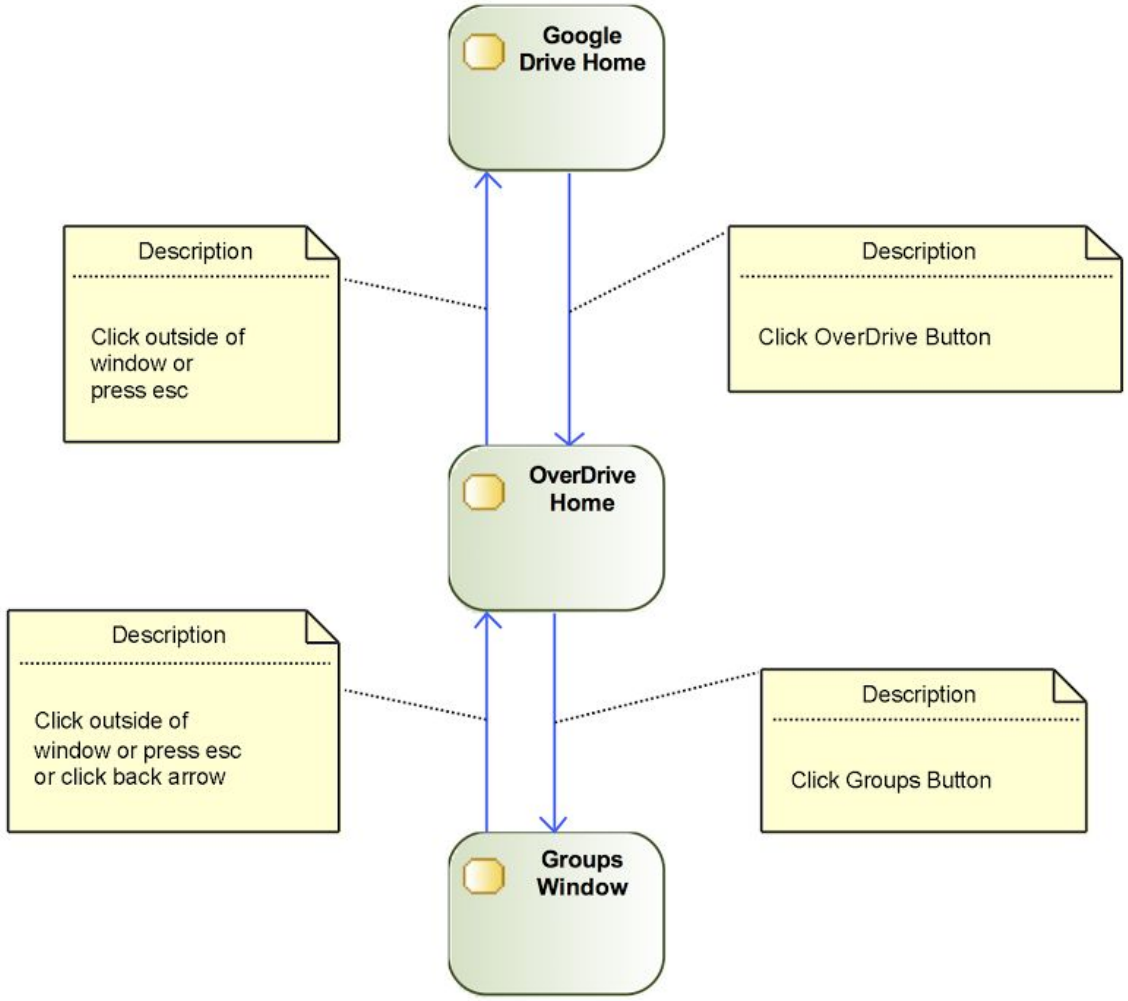
Activity Diagram



Method Diagram



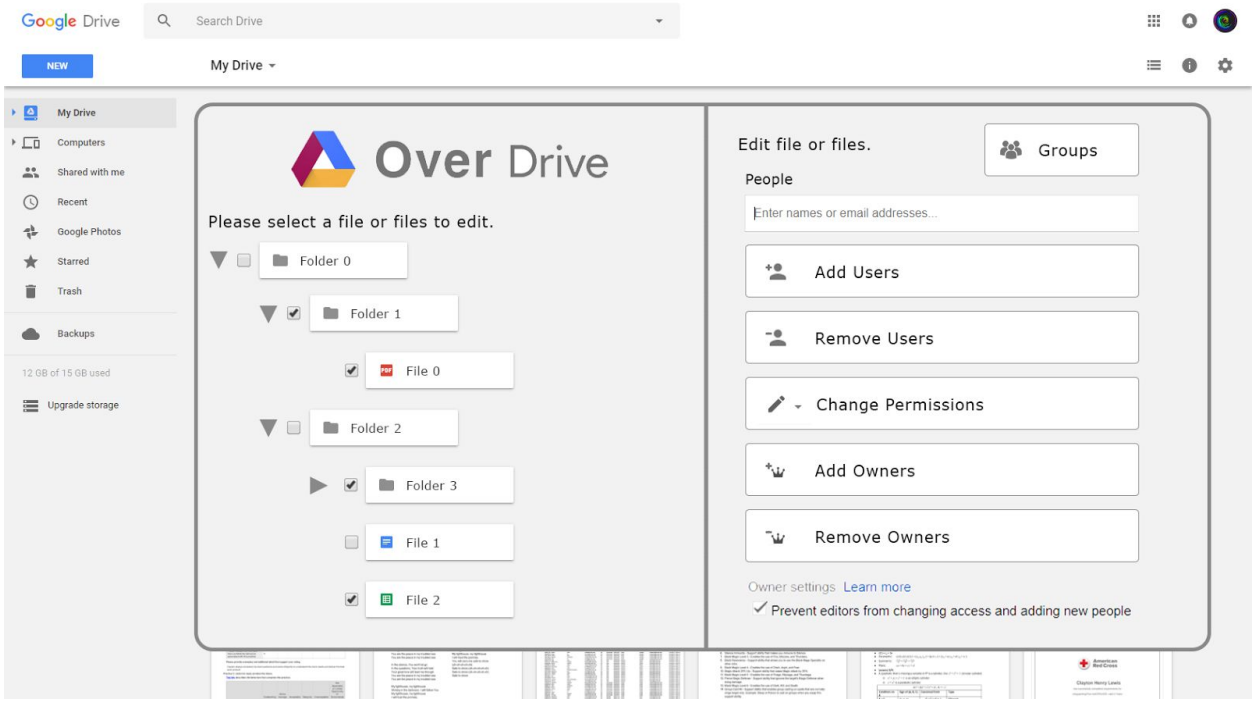
Navigation Flow Map



The user reaches the OverDrive home window by clicking on an OverDrive button on their Google Drive home page. Most functions of the application are available on this page. A user can click on the Groups button to open the Groups window. From there, a user can click a back arrow or click outside of the window to return to the OverDrive home window. From there, a user can click outside of the window to return to their Google Drive home page.

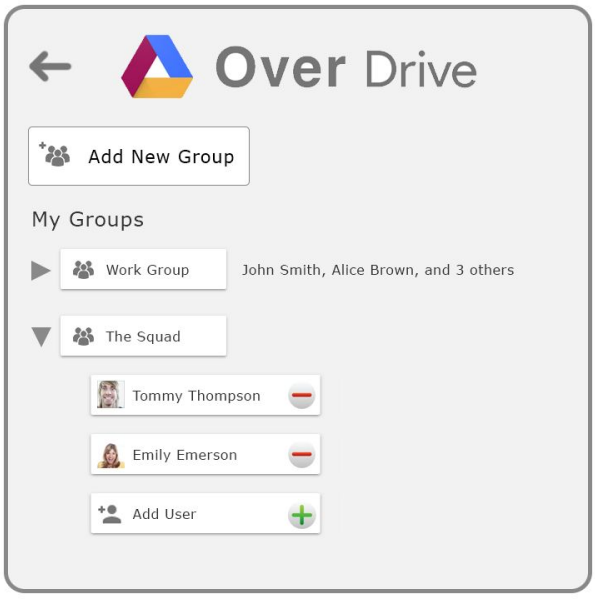
UI Mockups

OverDrive Home Window UI



The OverDrive UI offers the user a simple and elegant solution to managing files and teams. On the left side is a file browser. This view allows the user to see all of their files and folders. The user can expand or collapse a folder by clicking on the arrow next to it. The user can select a number of files and folders by checking the boxes next to them. After selecting the desired files, the user can go to the right side to manage those files. The user can type the names of Google Drive users in the text box. A user can also enter the name of a group to execute an action for all members of that group. The user can then click the button for the desired action. The action will be executed for the files selected and the users listed. The user can click the Groups button to open the Groups Window.

Groups Window UI



The Groups Window allows a user to manage their groups. The window contains a list of the user's saved groups. A group can be expanded or collapsed using the arrow next to it. When a group is expanded, a user can remove a group member by pressing the red minus, or add a group member by pressing the green plus. The user can click the Add New Group button at the top of the window to create a new empty group.