DropBucket

Team: Thomas Alder, Matthew Donovan, Ryan Loi

Project Overview: DropBucket is a Dropbox-like file-hosting service that uses GCP buckets. A user will be able to sign in using Google OAuth to upload, download, or delete files. We will have a client application that communicates to a server hosted on GCP. The client will watch a folder for changes and sync the files with the server when changes are made. The client will also check the server for updates on the folder and if any were made, it will download the changes.

Language/Toolsets for Code:

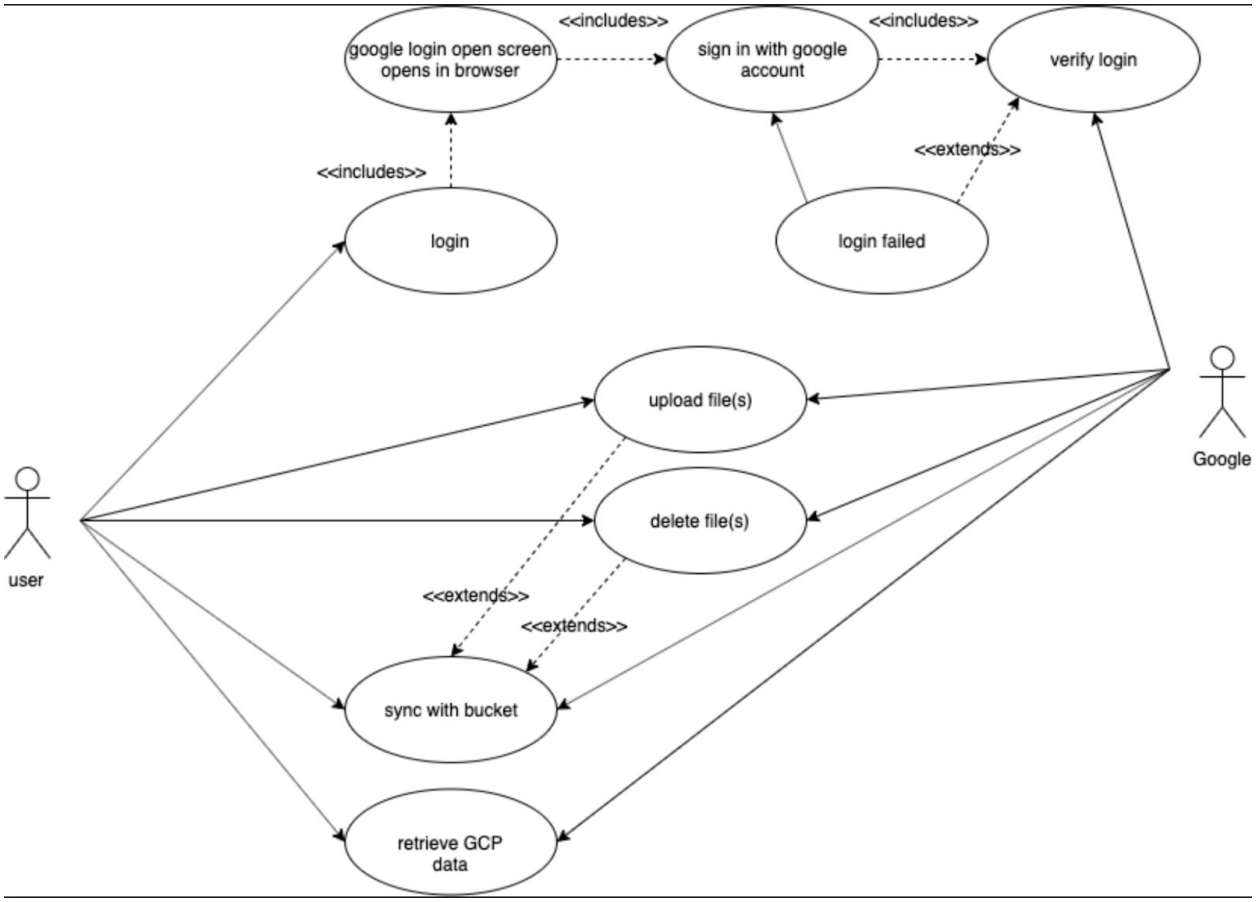
- Google Cloud Platform
- Qt/C++ (client)
- Python (server)

Requirements and Responsibilities of System:

- The client is required to 'watch' user defined directories for changes
    - When changes occur the client is responsible for syncing those changes to a GCP bucket
- The client is required to 'listen' for a 'new data' message from the cloud
    - When the client receives a 'new data' message from the cloud it is responsible for pulling these updates to make sure the local directory is up to date
- The server is required to 'listen' for a 'subscribe' message from clients (1 or more)
    - When the server receives a 'subscribe' message from a client it is responsible for associating this client with a GCP bucket and adding it to a 'group of clients' for this user (ideally there should only be one client per laptop)
- The server is required to 'listen' for 'new data' messages from clients
    - When the server receives a 'new-data' message from a client it is responsible for syncing the client's new data with the data in the bucket and the server
    - When the server receives a 'new-data' message from a client it is responsible for sending 'new data' messages to the other clients in this group

Use Cases

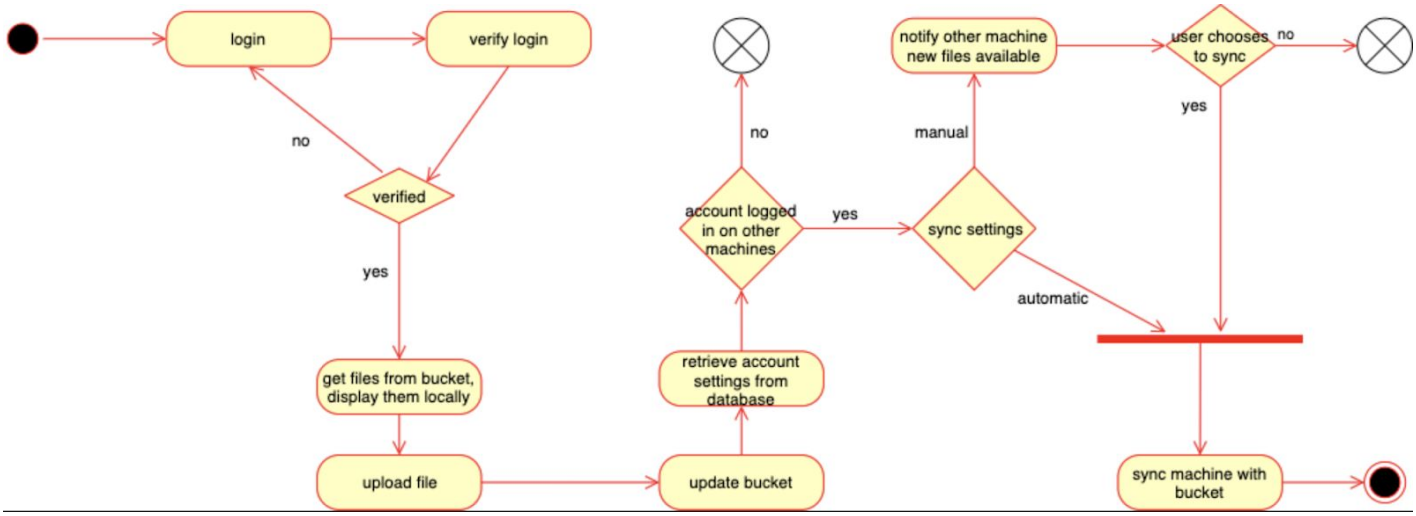
- User uploads a document
- User downloads a document
- User deletes a document
- User can pause/start a sync
- User can query data on documents stored on the cloud
    - Total space used
    - Total space per directory
- User can sign up with a Google account
- User can sign in with a Google account



Activity Diagram

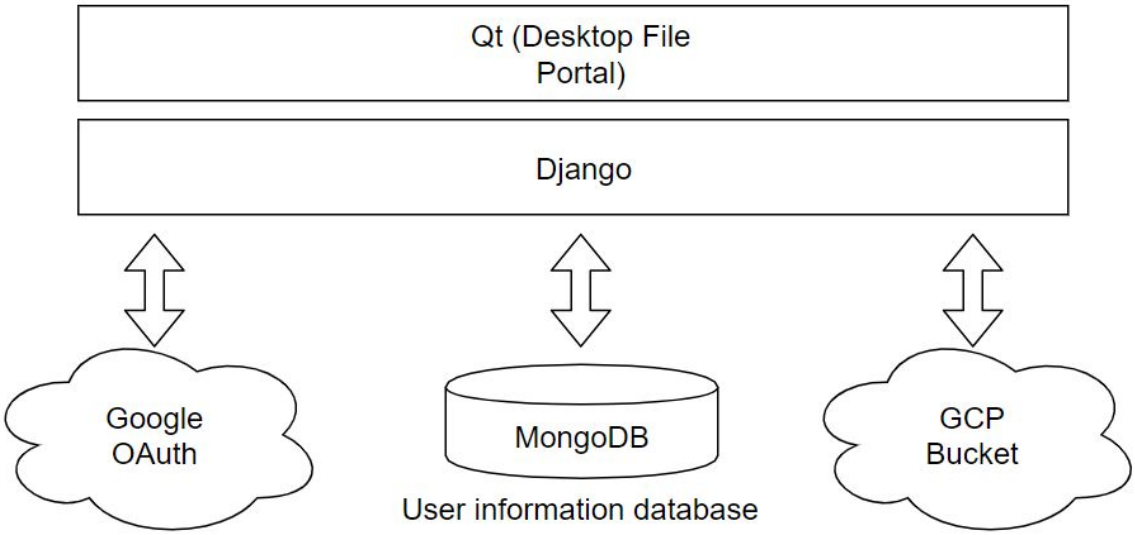
This activity diagram shows the steps for a user logging in to Dropbucket on a machine and uploading a new file.

After logging in using Google OAuth, the machine will pull down the contents of their bucket to make their local copy reflect the most recent remote version. The user can then upload a file, at which point the database will be queried to see if the user is logged in on any other machines. If they are, the sync settings on those machines will be checked. For machines with sync on, the most recent save will be pulled down. Machines with sync off will be notified that they are not up to date with the latest version and given the option to sync with the bucket.



Architecture Diagram

For our architecture, we mainly have two layers for a frontend and backend. Our frontend on the client side will be developed with the Qt framework and C++. This will interact on a Google Cloud Product hosted Django and Python server which will satisfy requests made by the user on the GUI. To hold user information, we are going to be using a MongoDB database which will include information about the user log in, devices, sync status, and directory manifests. In order for the files to be hosted on GCP the backend will work with a GCP Bucket in which the files will live. These will be synced to the user's local machine and vice-versa when needed. For sign in, we will be requiring the user to have a google account and sign in with Google OAuth. The frontend will direct the user to the browser for sign in and information will be passed to the backend for validation. The frontend won't have direct access to the MongoDB, GCP Bucket or Google OAuth as we want all of it to be handled by the backend Django server.



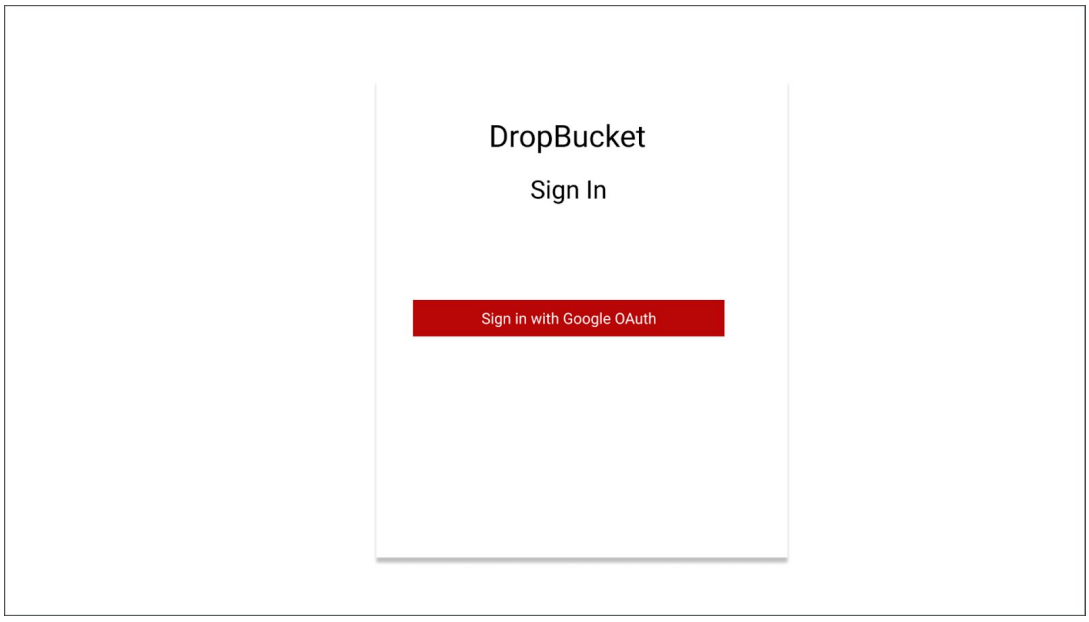
Data Storage

For our application we will need to store user account data and information that will allow us to sync files among many devices (if a user has many devices linked to one account). This data will be stored in a MongoDB database running on a GCP VM. We will create a Facade class to provide an interface to the database. This will give us a simple interface and allows us to limit the database accessibility from our API server. The interface should allow the API to query data and insert/delete data from the database.

UI Mockups/Sketches

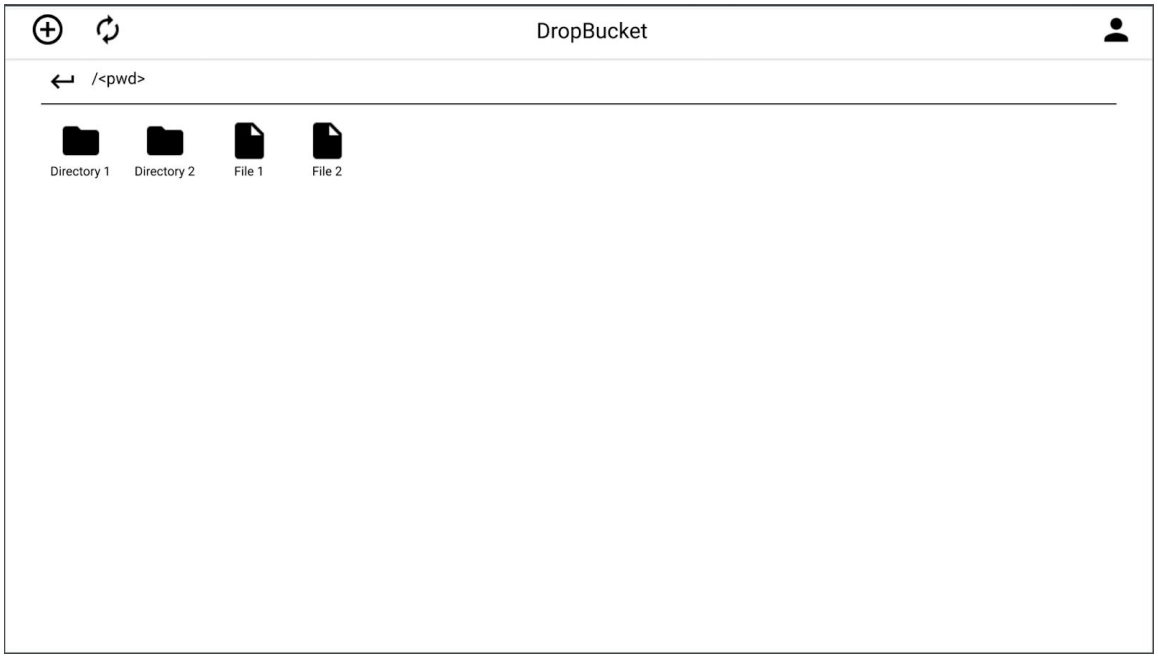
Sign In

First, we have a generic sign in with a link to the Google OAuth link that will open in the browser. We can extend this if we choose to include more sign in options using GCP's Firebase.



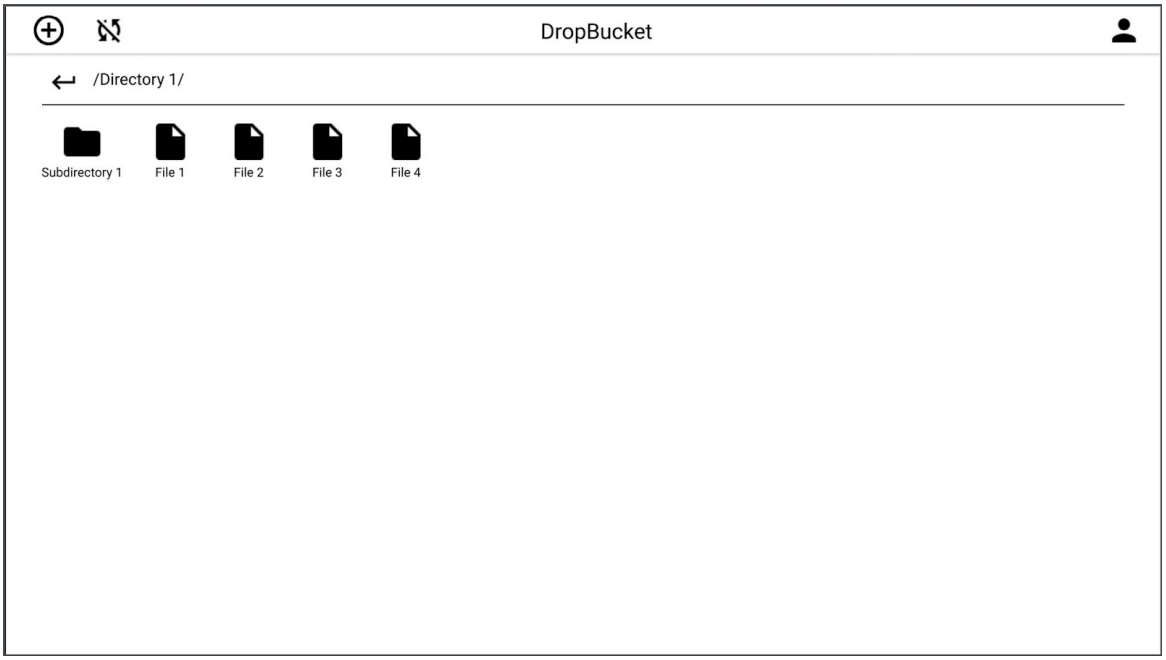
Main (Root) Directory

Once signed in, the user will be taken to the root directory of the Dropbucket hierarchy. There are three main buttons, the add button in the top left for uploading files, the sync button which turns the sync to on or off, and the profile button for sign out (last mockup). Files and folders will be rendered below the main header and current directory subheader.



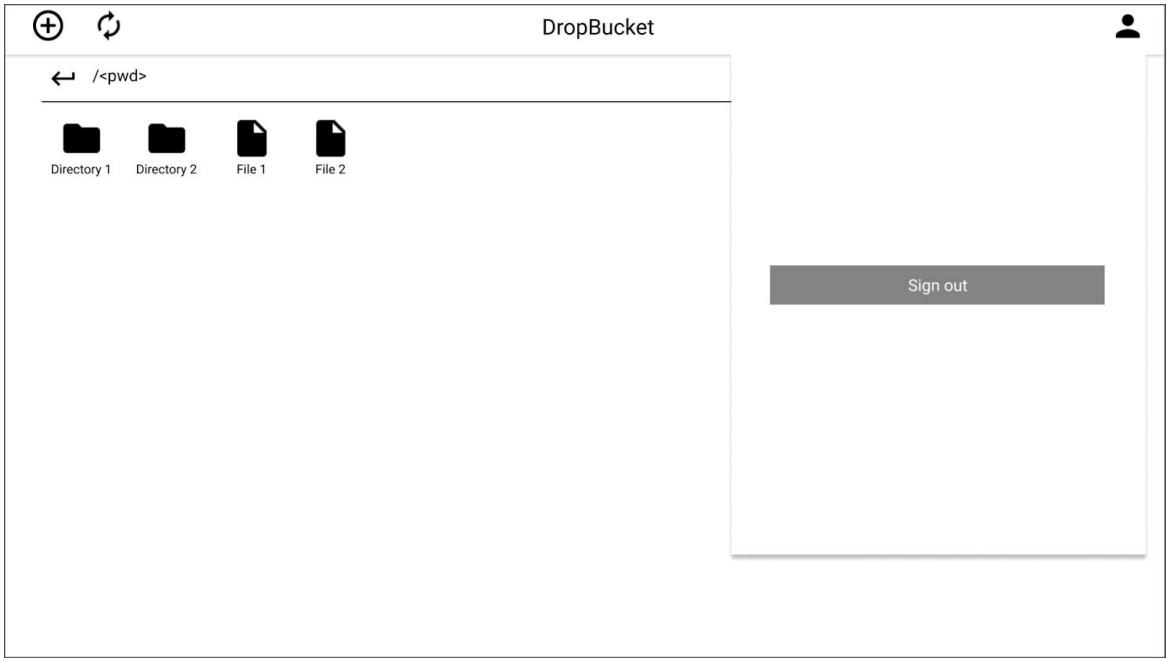
Subdirectory

Users can click on the directories to take them to the one selected and will render out the respective contents of the directory. This mockup also shows what happens when the sync button is turned off.



Profile Button Press/Sign out

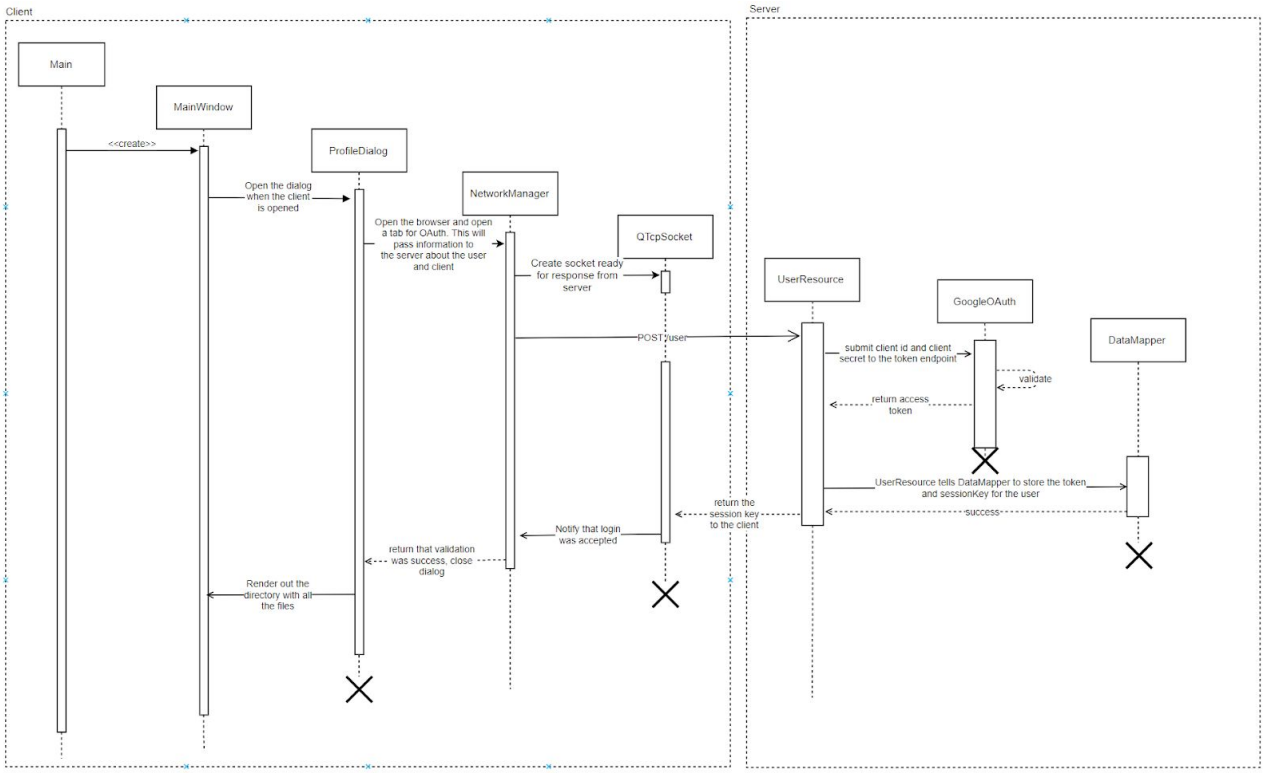
When the profile button is clicked the option to sign out is presented with a dialog. Once pressed this will take the user back to the sign in screen.



User Interactions/Sequence Diagram

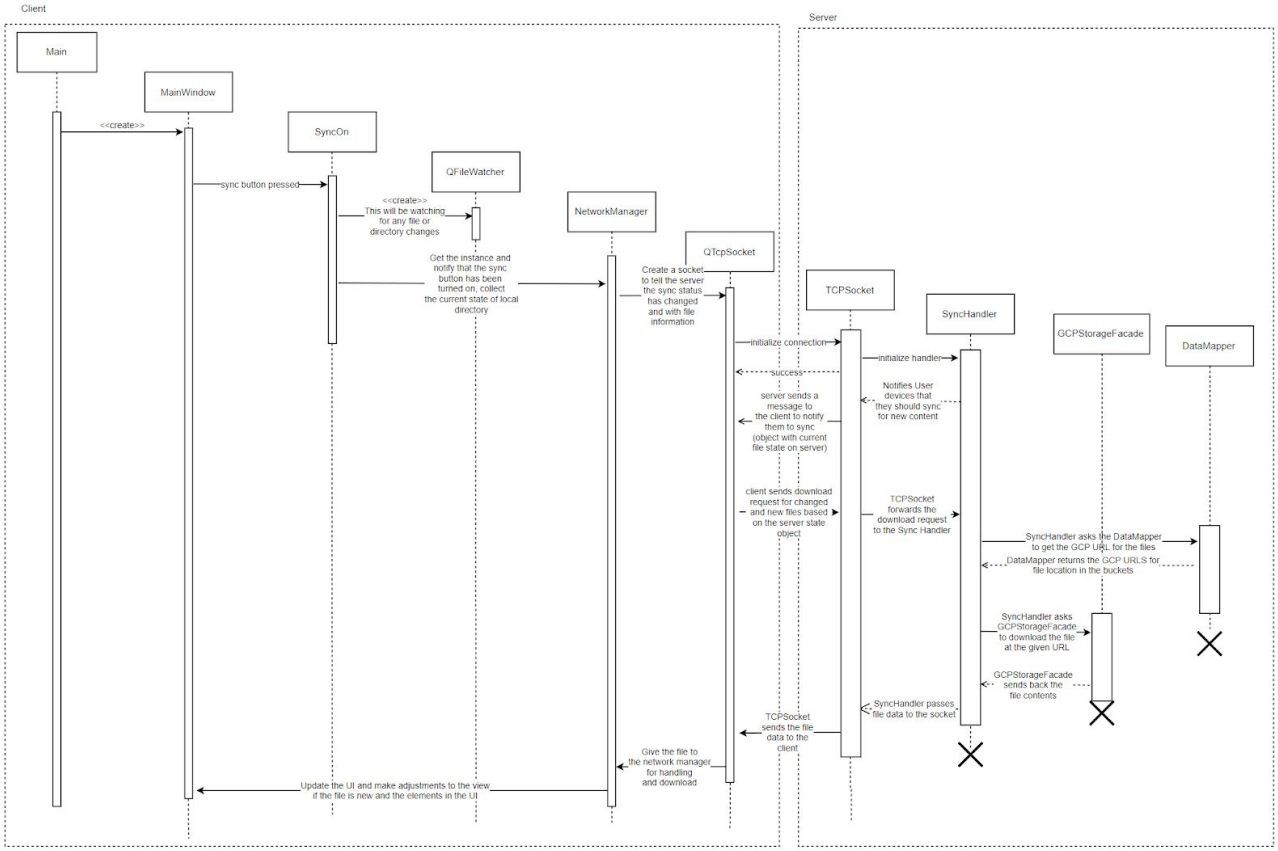
Log In

Description: The user will log in with a Google account and send the client id and client secret the /user endpoint on the server. The server will then pass these values on to the Google OAuth server which should return an access token if it was able to validate the user. The server will create a session key based on the access token. Then, the server will use DataMapper to store the access token and the session key in MongoDB. Once this is stored, the server will send a response to the client with the session key.



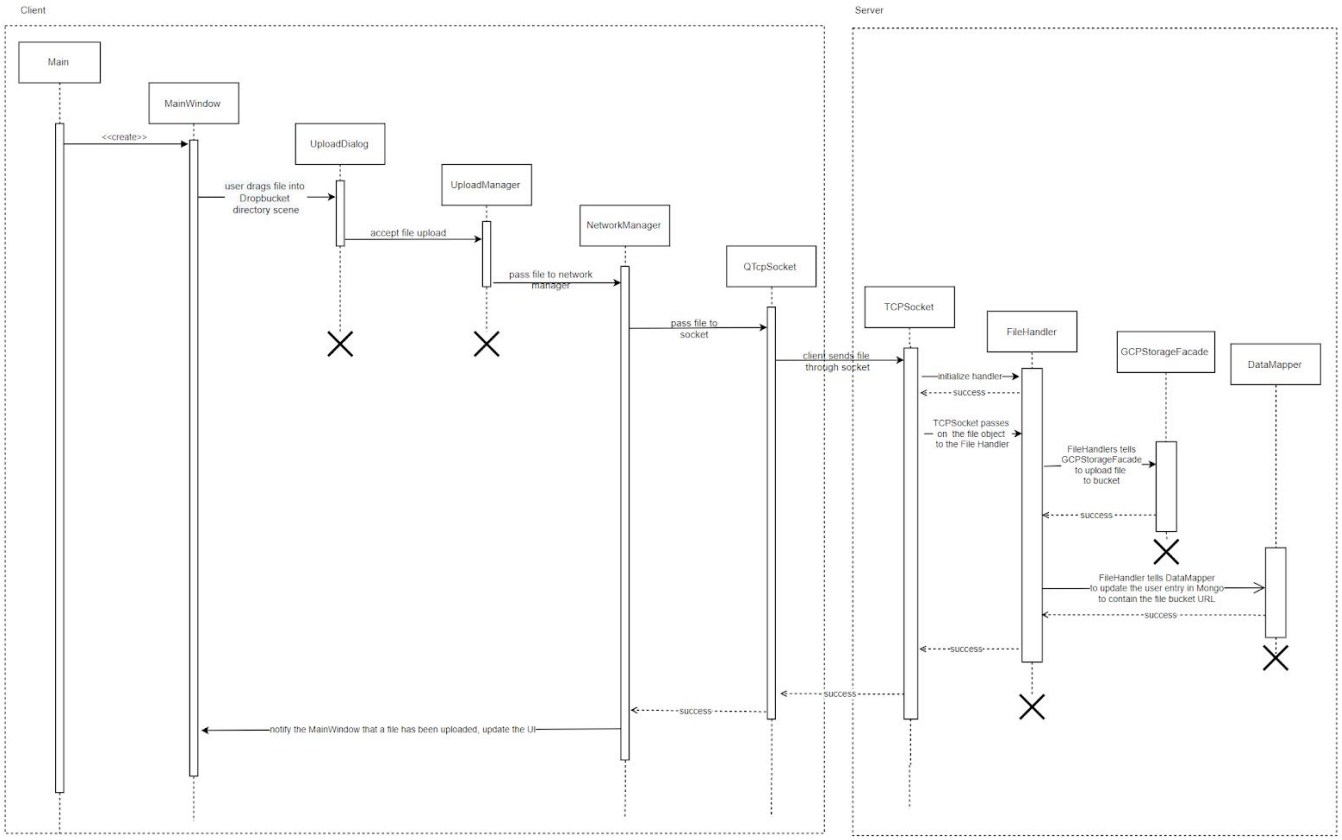
Turn on Sync

Description: This interaction assumes that the sync button is turned off. The user will turn on sync which means that they want to sync their current state with the servers. First, a file watcher is created which invokes a slot when there are any modifications to files or to the directory. If there are any changes, this will communicate to the NetworkManager that files need to be uploaded to the GCP bucket. The NetworkManager is notified that the sync state has been changed. This will upload any files and notify the server via the QTcpSocket of the state change and the files being uploaded. The server will create a SyncHandler which communicates with the DataMapper for the url for the GCP bucket. Once the urls are collected, the GCPStorageFacade to get the files which will then be passed to the TCPSocket to send to the client which will then render out any new files and update the UI to show the file or directories downloaded to the client machine.



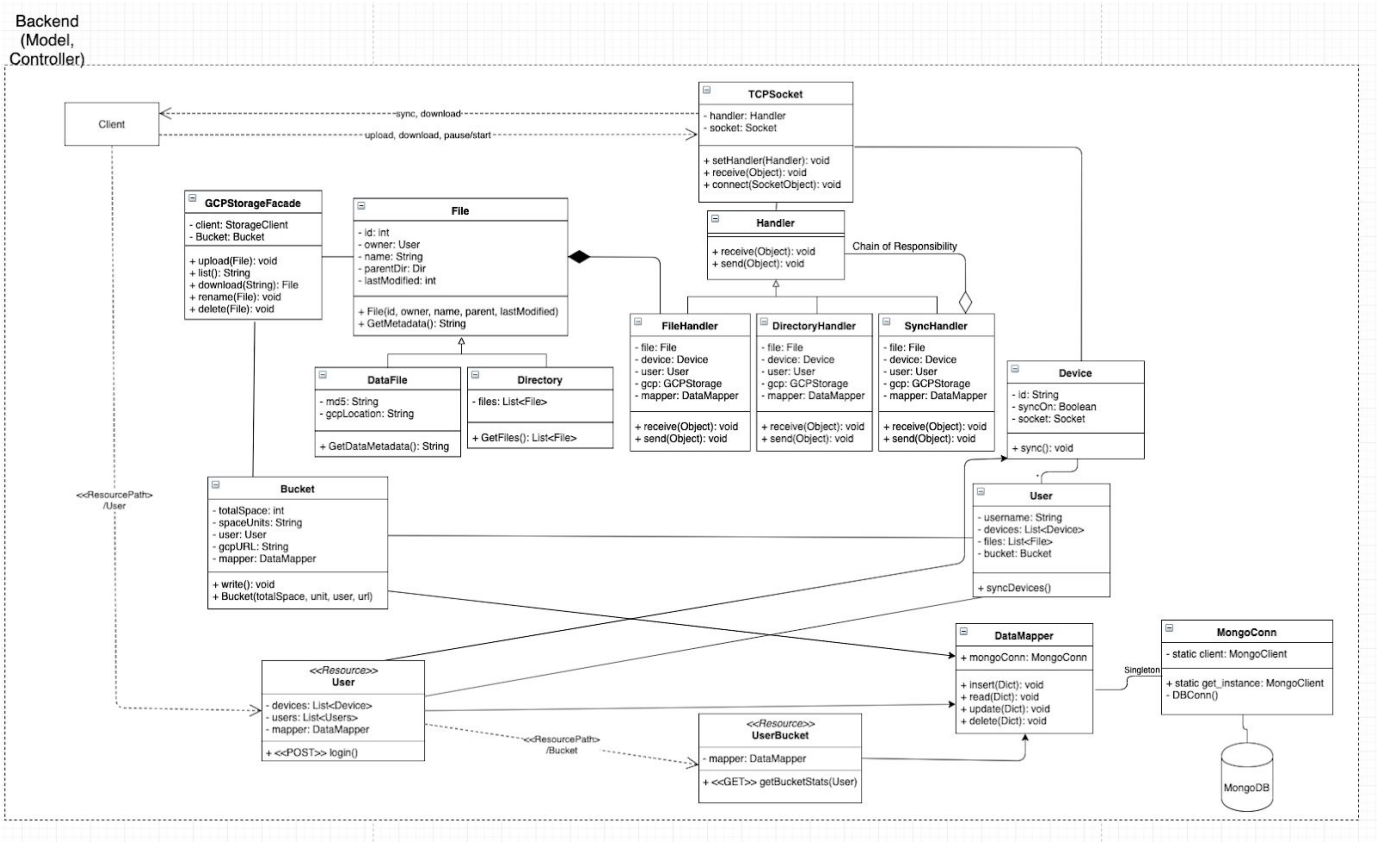
Upload File

Description: The user will upload a file by dragging it into the Dropbucket GUI. This will prompt a dialog to appear that requires the user to accept before moving on. Once accepted, the file will be passed through the network manager to the QTcpSocket object. The QTcpSocket will send the file to the TcpSocket which will initialize a FileHandler that will communicate with GCP using a Facade. Once the file has been uploaded to GCP, the FileHandler will tell the DataMapper to update the user entry in Mongo to reflect the changes.

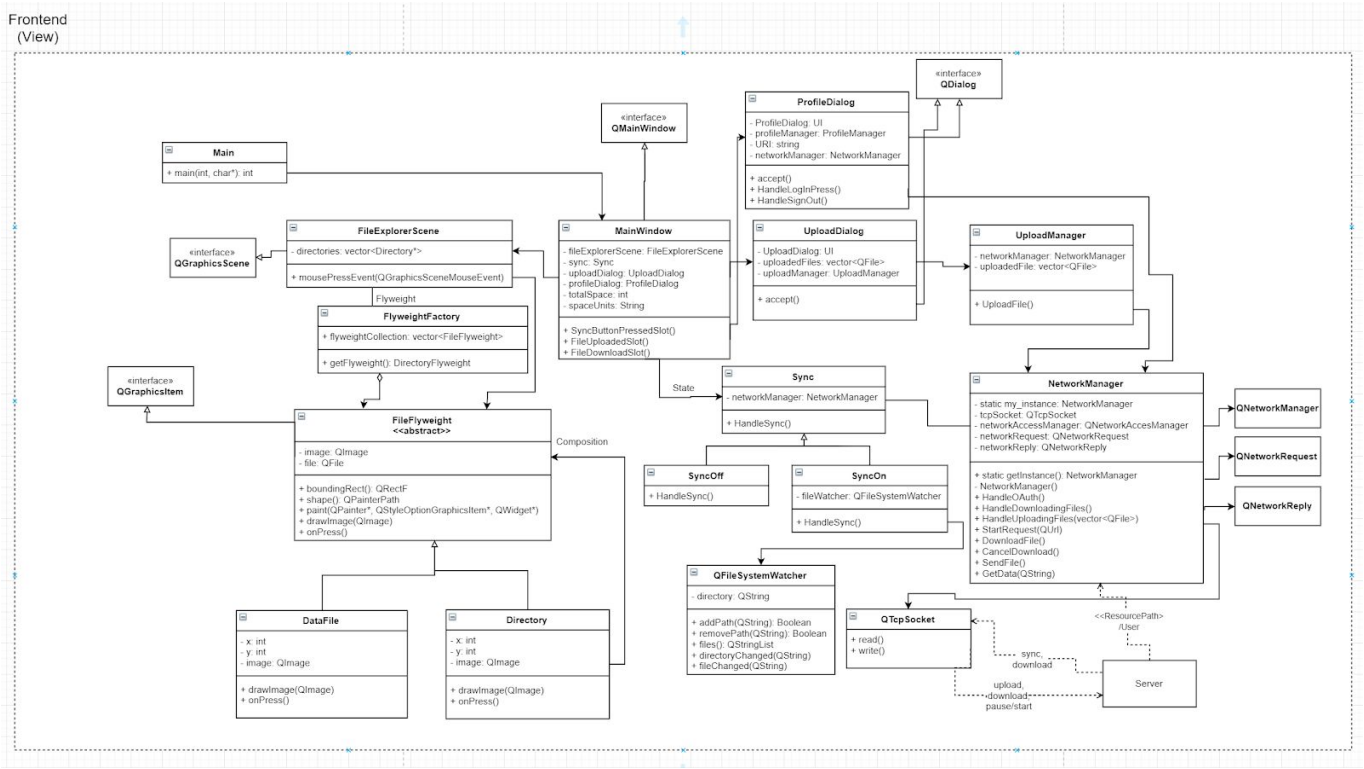


Class Diagram

Backend



Frontend



Pattern Use

Backend Patterns:

So the backend will use a MVC pattern throughout (the view is the client side). The controllers will be our REST API endpoints and handlers while the model will be Objects relating to the database. There will be a DataMapper pattern used to communicate with the Mongo database. We also will use the Chain of Responsibility pattern for handling File, Directory, and Sync requests from the TCP socket. We will use a Facade pattern to communicate with the GCP buckets.

Frontend Patterns:

The frontend will be using flyweight and factory for rendering and creating the file and directory icons which will help in the case of a lot of files and memory usage. Files and directory graphic items will use the composition pattern as directories are composed of files and the graphics items will have shared behavior when clicked and rendered. To differentiate between sync on and off, state will be used so that when it is off the sync handling will follow a Null Object pattern since nothing will occur if the state state is invoked. When the sync is on the on state will be able to download or upload files and sync with the server.