# Project 5

DropBucket

Thomas Alder
Matthew Donovan
Ryan Loi

## Status Summary

### Work Done

*Front End*

So far on the front end, we have implemented loading the root directory into the file explorer scene and rendering out the contents. When a folder is clicked in the file scene, the scene is cleared and the clicked folder's scene is rendered. Currently, we are working on the return button but have been getting segfaults when it is clicked, though the issue should be ironed out soon. The framework has been set up for the syncing so when we are implementing the network part of the client to the server, that is ready to detect changes when it is on. In addition, we have begun the Google OAuth implementation on the Qt side which we will hook up with the server.

*Back End*

On the back end side we have a Django backend set up using Django REST Framework that uses SQLite to store the data. We have models representing User to hold the Google OAuth credentials for a person, Bucket to hold GCP storage bucket-related data, Device to hold info that relates to the configuration of a given user's devices, and a File object that will be used to hold data for each file in the bucket.

We have also been working on a GCP Storage facade that will eventually interact with the Django models. The facade provides a way to create and interact with GCP buckets (one per user) such as: uploading a file, downloading a file, and deleting a file. On the GCP side of things, we needed to set up a service account to allow the facade to perform these bucket operations.

**Work Estimate**

*Front End*

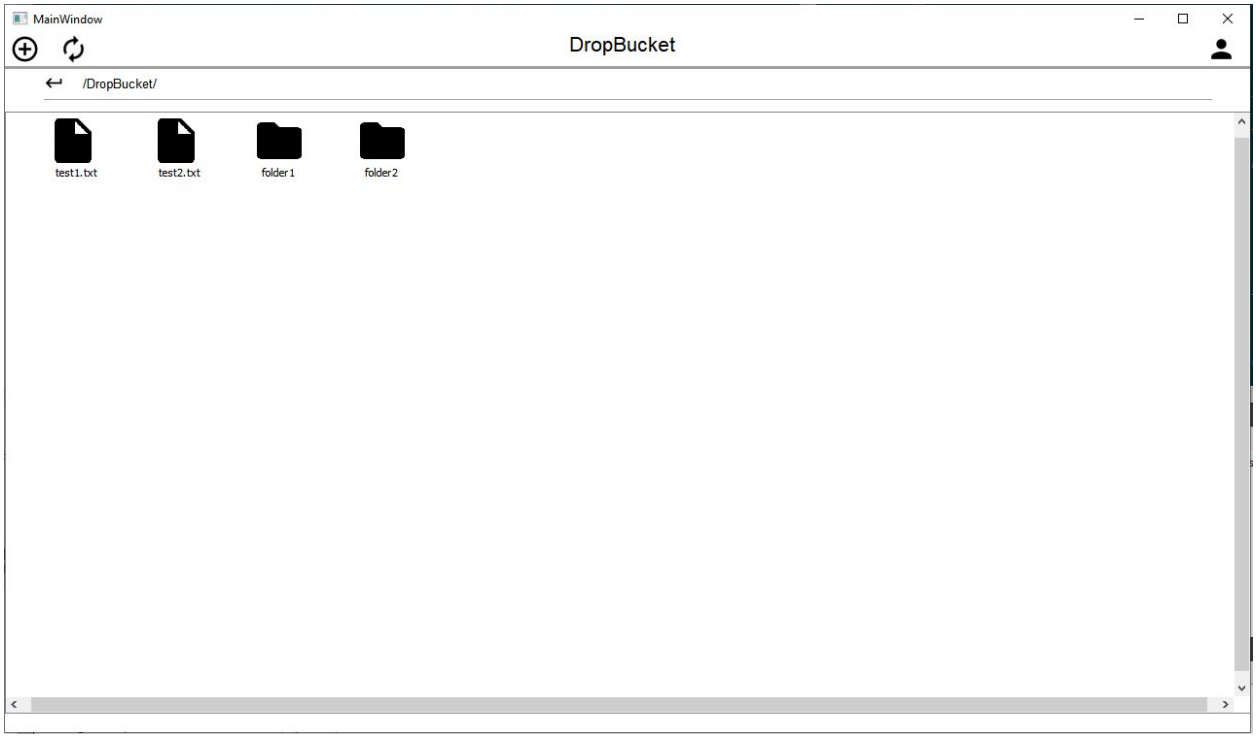
For the front end, implementation for uploading and deleting folders/files needs to be completed in addition to creating a thread which monitors the directories for any changes and depending on the sync status it uploads or downloads files. Once the OAuth is done, sign out can be implemented and the networking between the client and the server can be worked on meaning the initial JSON needs to be sent following sign on outlining the users client set-up so we know their root directory in addition to the current files they have uploaded on the bucket. The JSONs need to be compared to download or upload any files given the current state and from there, files can be uploaded or downloaded depending on the Sync state.

*Back End*

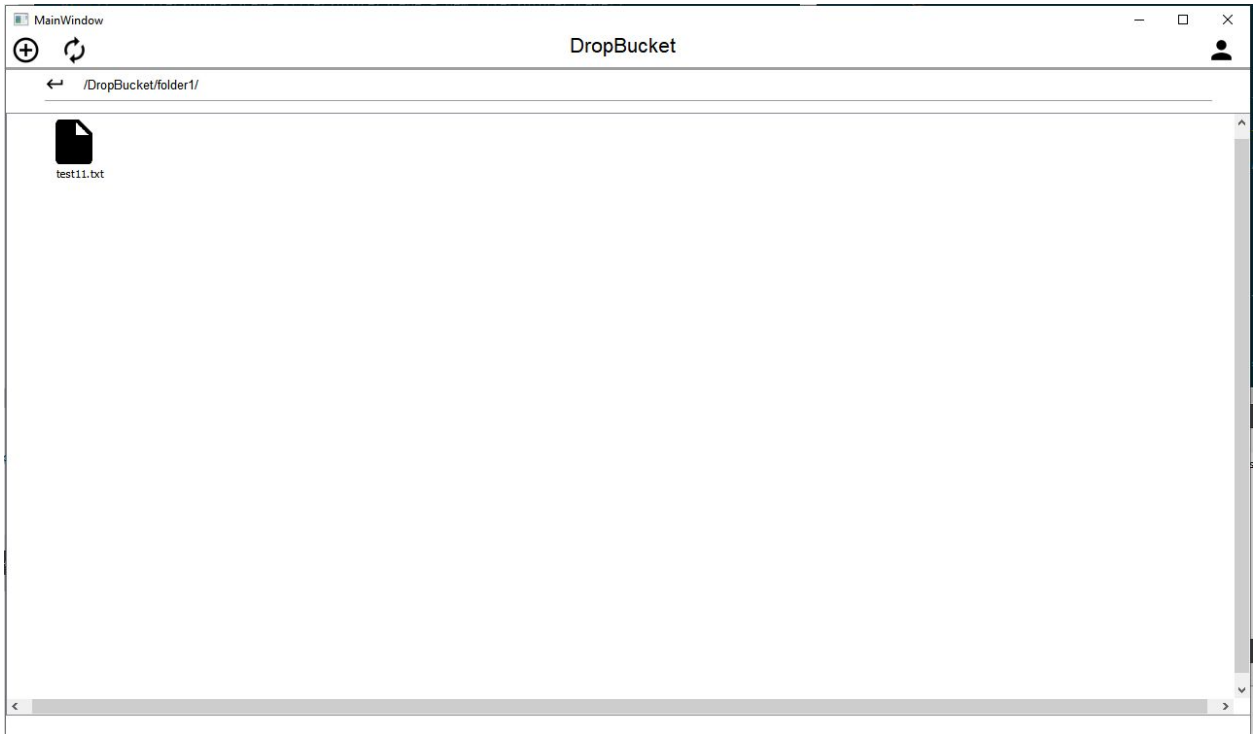
There is still a good amount of work to be done on the backend that relies on the work we are currently doing with Google OAuth. Once we have the OAuth completed, we will begin combining the Django models with the GCP Storage facade to make requests to GCP Storage API that will handle sending and receiving updates when the user add/deletes/changes files in the client. We have changed from using a Mongo database to using Django's default SQLite as it requires no additional set up and allows us to get things functional faster and saves effort on the database aspect. After the database and GCP objects are set up, we will need to start working on the TCP socket connection to the client. This socket will allow two way communication for syncing files to and from the server and will most likely take 1-2 weeks to code.

**Screenshots**

*Root directory loaded*

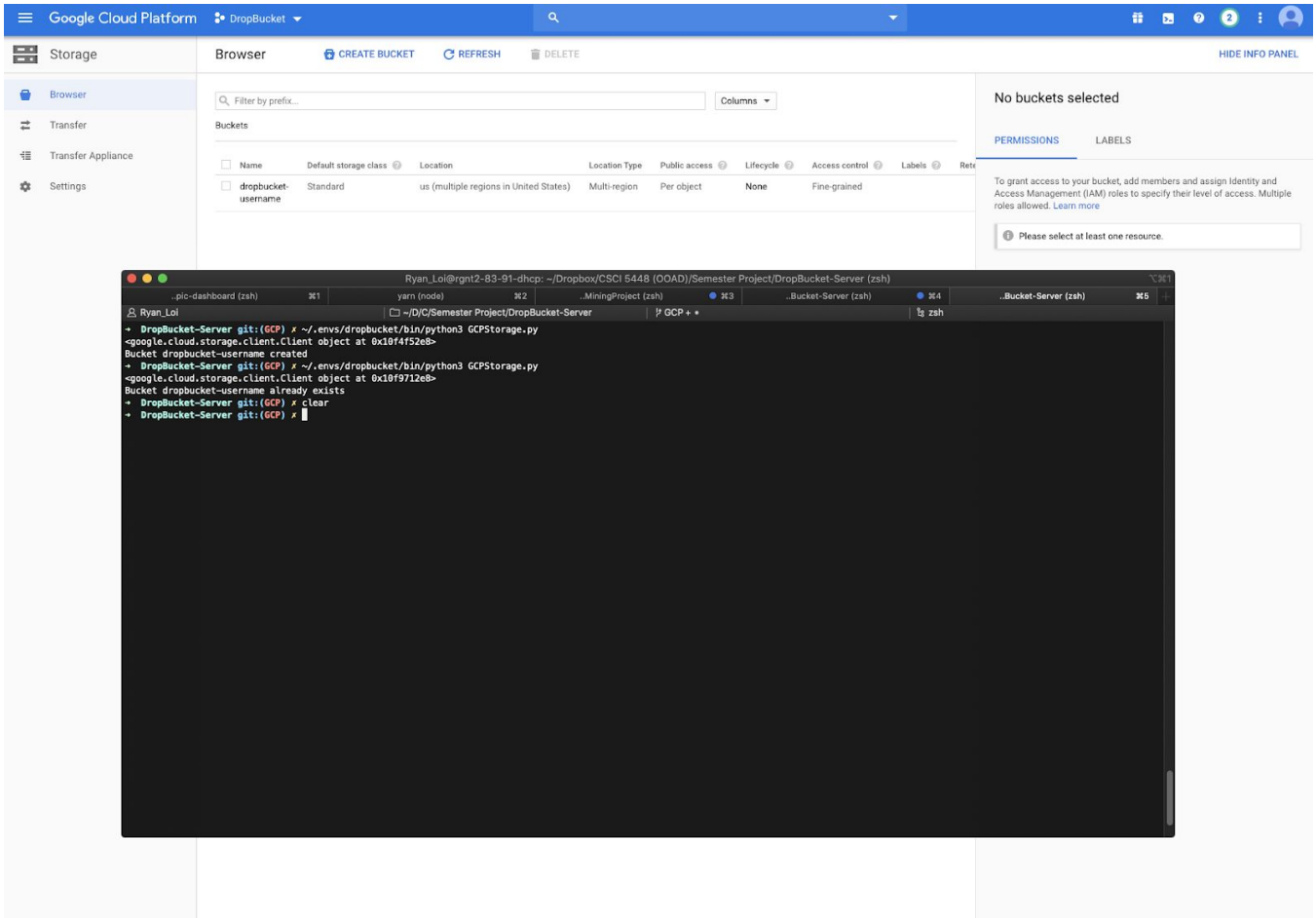



*After clicking on folder 1*



*Login Dialog created*


*GCPStorage Facade creating bucket for user*



**Test Cases**

*Front End*

On the front end, so far we have one test case which ensures the composite structure of Directories is set up correctly when it is generated with a test json file and the root directory is rendered out correctly by checking the size of its item vector.

*Back End*

None yet, only manual testing up to this point.

**Patterns**

Overall, the project between the front end and back end is MVC where the model is the client and the view/controller is the server.

*Front End*

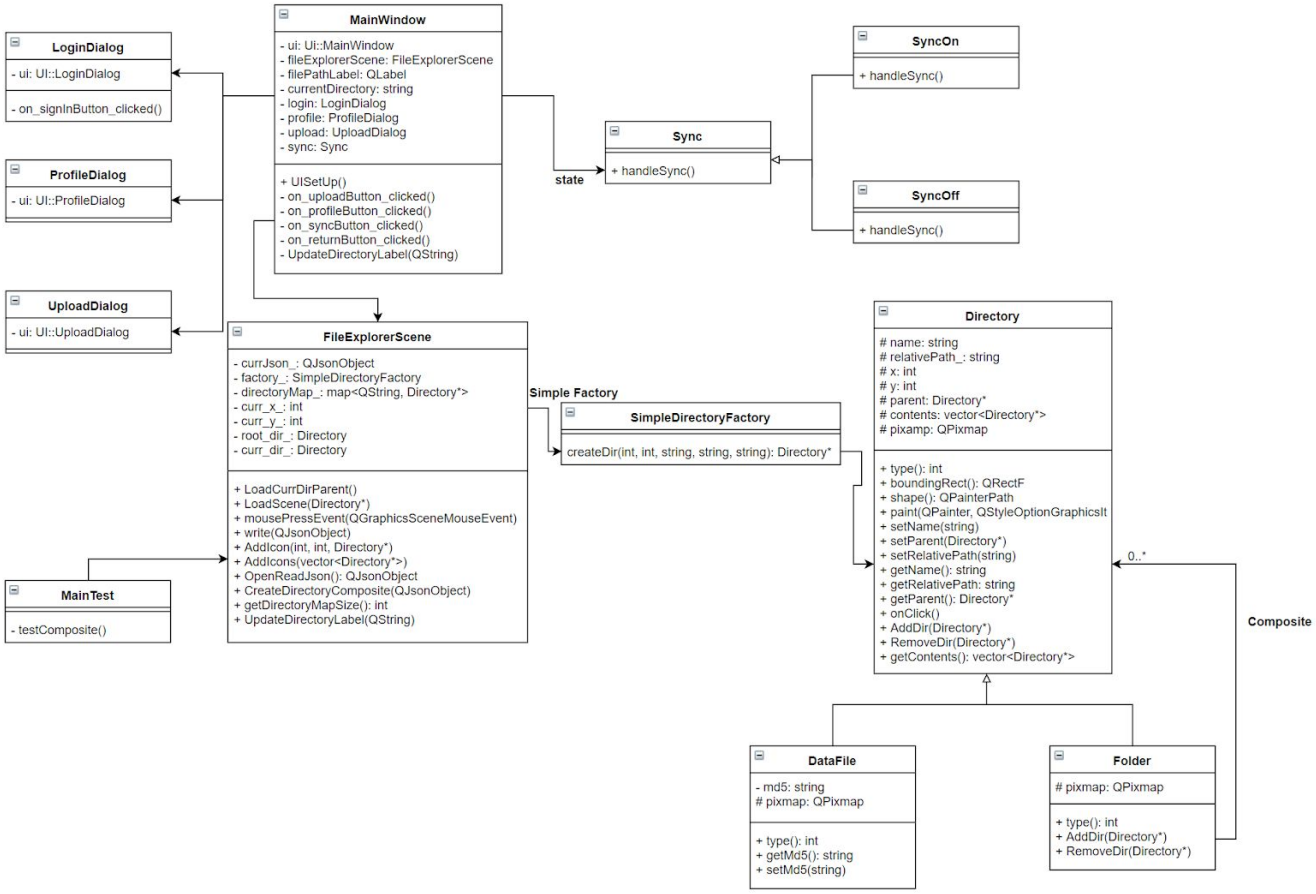
Currently on the front end, we have implemented a Composite structure where the Directory class is an abstract class for concrete classes DataFile and Folder. These serve as the items which are rendered out on to the file explorer scene when the user is navigating around the hierarchy. A simple factory is used to create the Directory subtypes of DataFile and Folder. In addition, we have the framework for State set up so when we are working on the networking that is ready for detecting updates and syncing with the server when necessary. In multiple places, we are using the STL style Iterator implemented in C++ to traverse through the vectors and maps we have set up in the project.

*Backend*

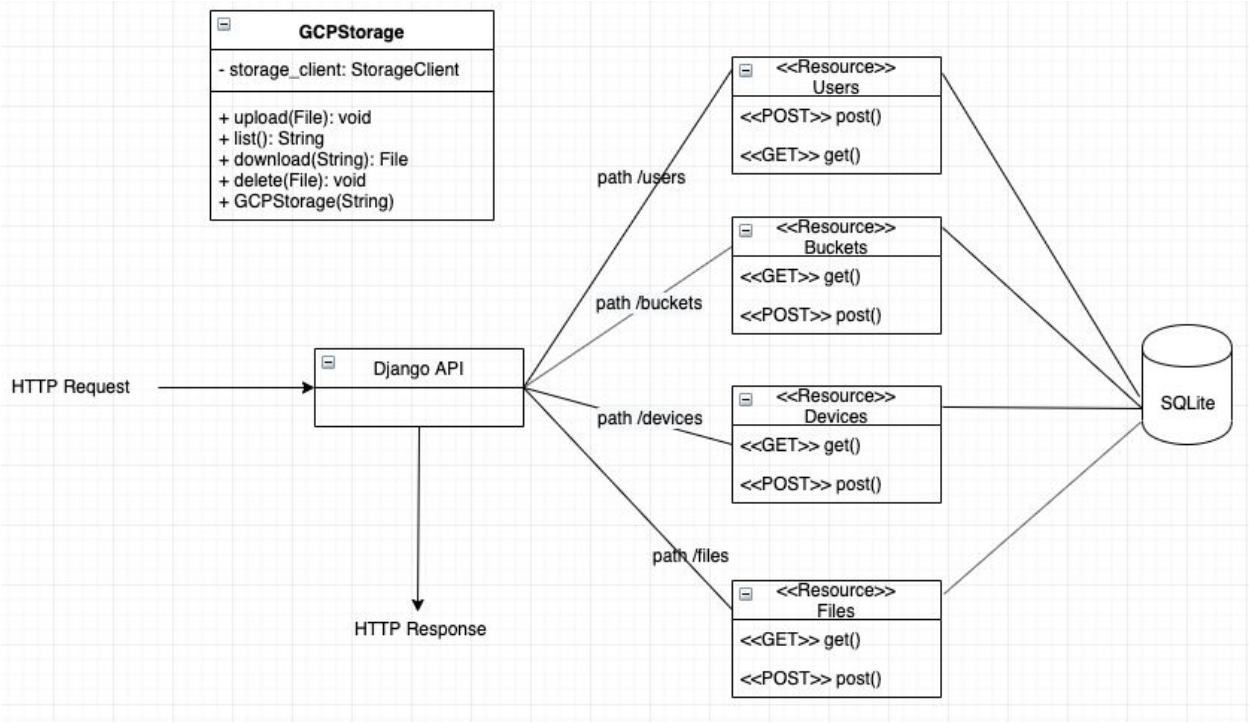
The backend so far has only implemented a Facade pattern for the GCP Storage functions. This gives us a simple interface to upload, download, and delete files in a GCP bucket without needing to worry about how to actually do this with the Google Storage API. This decouples and abstracts the Google Storage API usage away and the Django models that will use this facade will not even know about it.

## Class Diagrams

*Front End*



*Backend*




## Plan for Next Iteration

For the front end, we plan on first finishing the client side operations, e.g. state implementation checking and updating files, adding files, deleting files, log out. Once that is complete, we then need to transition to ensure that the communication and networking between front end and back end is working. This means that first we need a networking implementation on the client side to ensure that we can talk to the server then pass the necessary information between the two ends. Finishing the client side operations will take approximately one week, in which over break and during the week after we can concentrate on the networking.

As mentioned in the work estimate section, we will need to integrate the facade that will send requests to GCP and the Django models. This can be done during the time that the client work is finished, so when both of those pieces are complete we can all swarm on getting the front end and back end connected. This work will include setting up OAuth and a TCP socket connection.