

# Design Document

Captain CyBeard: Neil Before Us

**Ryan Breitenfeldt | Noah Farris**  
**Trevor Surface | Kyle Thomas**

May 4, 2020



Washington State University Tri-Cities  
CptS 423 Software Design Project 2

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Architecture</b>	<b>1</b>
<b>3</b>	<b>Data Dictionary</b>	<b>4</b>
3.1	Main . . . . .	4
3.1.1	Associations . . . . .	4
3.1.2	Attributes . . . . .	4
3.1.3	Methods . . . . .	4
3.2	class Platform_template . . . . .	5
3.2.1	Associations . . . . .	5
3.2.2	Attributes . . . . .	5
3.2.3	Methods . . . . .	5
3.3	Dropbox . . . . .	7
3.3.1	Associations . . . . .	7
3.3.2	Attributes . . . . .	7
3.3.3	Methods . . . . .	8
3.4	AWS . . . . .	10
3.4.1	Associations . . . . .	10
3.4.2	Attributes . . . . .	10
3.4.3	Methods . . . . .	10
3.5	GDriveDownloader . . . . .	11
3.5.1	Associations . . . . .	11
3.5.2	Attributes . . . . .	11
3.5.3	Methods . . . . .	11
3.6	Platform_Selection . . . . .	13
3.6.1	Associations . . . . .	13
3.6.2	Attributes . . . . .	13
3.6.3	Methods . . . . .	13
3.7	Authentication . . . . .	14
3.7.1	Associations . . . . .	14
3.7.2	Attributes . . . . .	14
3.7.3	Methods . . . . .	14
3.8	File_Selection . . . . .	15
3.8.1	Associations . . . . .	15
3.8.2	Attributes . . . . .	15
3.8.3	Methods . . . . .	15
<b>4</b>	<b>User Interface</b>	<b>16</b>
4.1	User Interaction . . . . .	16
4.1.1	Platform Selection Screen . . . . .	16
4.1.2	Authentication Screen . . . . .	17
4.1.3	File Selection Screen . . . . .	17
4.2	Administrator Interaction . . . . .	18
4.2.1	Start Application . . . . .	18
4.2.2	Stop Application . . . . .	19
<b>5</b>	<b>Information Repositories</b>	<b>20</b>
5.1	Cloud Platform Repository . . . . .	20
5.2	Local Storage or Internal Infrastructure . . . . .	20

## List of Figures

1	The overall application architecture. . . . .	1
2	The template class all platforms follow . . . . .	2
3	The AWS Class. . . . .	2
4	The Dropbox Class. . . . .	2
5	The GDriveDownloader Class. . . . .	2
6	The Controller and Views Classes. . . . .	3
7	The workflow for the user . . . . .	16
8	The first screen of the application. . . . .	17
9	The screen to view and download files. . . . .	18

## Revision History

Revision	Date	Author(s)	Description
3.0	05.03.2020	KT	Revisions to reflect reality
2.0	04.11.2020	KT	Revisions from review session
1.0	03.06.2020	RB NF TS KT	First Draft, revisions
0.5	03.04.2020	RB	Google Section
0.4	03.04.2020	NF	AWS Section
0.3	03.04.2020	TS	Dropbox Section
0.2	03.04.2020	KT	Filled in sections other than data dictionary
0.1	02.25.2020	KT	Document Creation

# 1 Introduction

This document will go over the design and architecture for the *Downloader* application that was laid out in **Requirements Specification**[2]. The design document will help the client (Cypherpath) and the software development team (Captain CyBeard) understand the architecture and functionality of the application.

Subsequent sections will go over the general architecture of the application in unified modeling language (UML) followed by a data dictionary. Afterwards the user interface and examples of information repositories needed by the application will be presented.

## 2 Architecture

The application will consist of ten classes in a model, view, controller architecture. The **Main** class will be the controller for the models (the classes that wrap up the cloud platforms) and the view classes will be the three classes that control what the user sees. The *model* classes (**AWS**, **Dropbox**, and **Google**) are templates of the **Template** class. This isn't an actual class that is implemented, however it is the format used when creating a cloud platform model and is provided for future maintainers of the project to more easily add classes.

The following UML diagrams will show the classes and their relationships with each other, along with their attributes and methods.

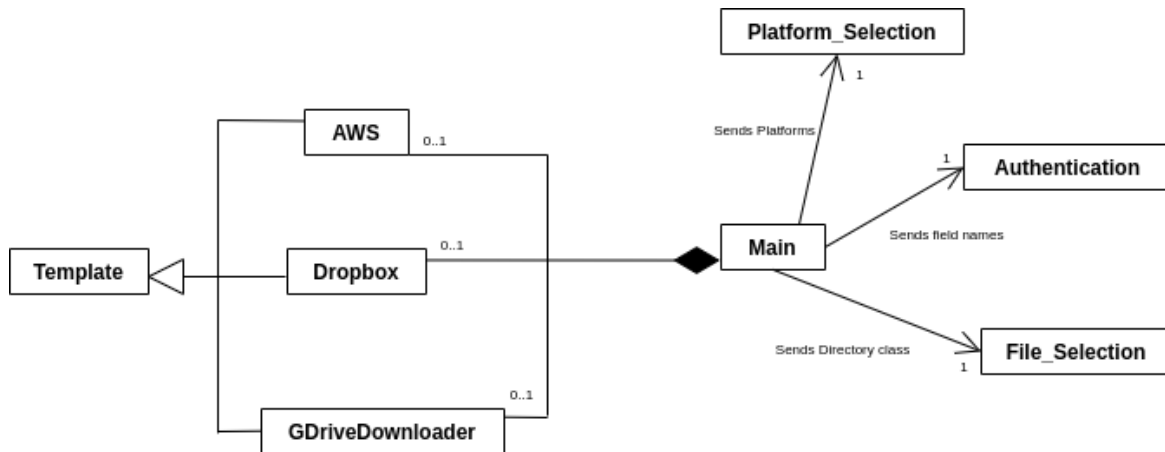


Figure 1: The overall application architecture.

Template
[NAME]_[MODEL]_[PARAM]:String [NAME]_get_files_list_result:String [NAME]_entries_to_download:List
[NAME]_authentication(void) [NAME]_get_files_list(void) [NAME]_format_entries_list(in [NAME]_get_files_list_result:List) [NAME]_select_entries_to_download(in [NAME]_entries_to_download:List) [NAME]_download_selected_entries(void)

Figure 2: The template class all platforms follow

AWS
aws_ec2_client_flow: Object boto3.client() aws_s3_client_flow: Object boto3.client() aws_s3_resource_flow: Object boto3.resource() aws_get_files_list_result: Object JSON aws_entries_to_download_list: Object JSON aws_entries_to_download_list: Object List aws_download_path: String="C:/Downloads"
aws_authentication: void aws_get_files_list: void aws_format_entries_list: void aws_select_entries_to_download_void aws_download_selected_entires: void

Figure 3: The AWS Class.

Dropbox
dropbox_api_key:String dropbox_api_secret:String dropbox_authentication_auth_flow:String dropbox_authentication_authorize_url:String dropbox_authentication_auth_code:String dropbox_authentication_oauth_result:String dbx:String dropbox_get_files_return:String dropbox_get_files_list_result:String dropbox_entries_to_download_list:String dropbox_download_path:String dropbox_dictionary:String
dropbox_authentication() dropbox_get_files_list() dropbox_format_entries_list(in dropbox_get_files_list_result:String) dropbox_select_entries_to_download(in dropbox_entries_to_download:String) dropbox_download_selected_entries()

Figure 4: The Dropbox Class.

GDriveDownloader
self.GDriveDownloader_creds = creds self.GDriveDownloader_SCOPES = ["https://www.googleapis.com/auth/drive"] self.GDriveDownloader_service = service self.GDriveDownloader_file_List = None self.GDriveDownloader_json = None
def GDriveDownloader_authentication() def GDriveDownloader_save_Token() def GDriveDownloader_load_Token() def GDriveDownloader_build_Service() def GDriveDownloader__download_File(in file_Id: file) def GDriveDownloader__get_Files()

Figure 5: The GDriveDownloader Class.

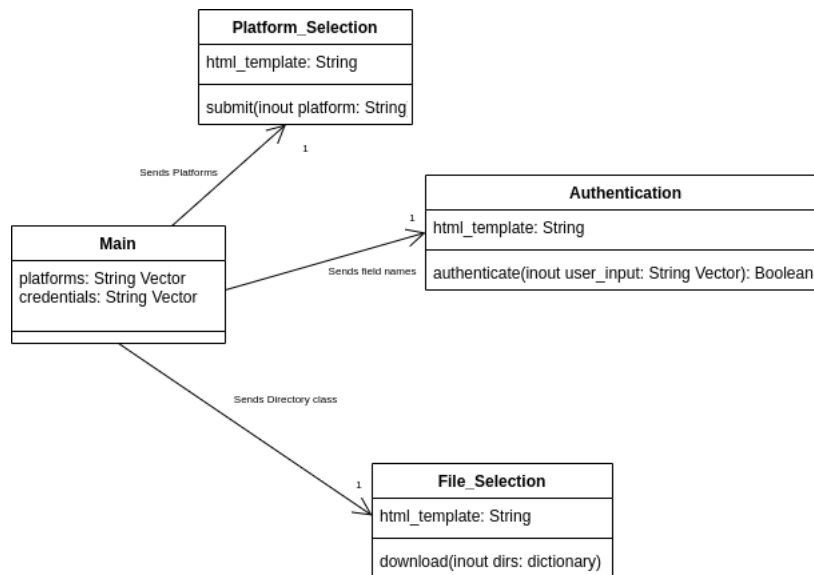


Figure 6: The Controller and Views Classes.

## 3 Data Dictionary

The Data Dictionary sections will describe what each class does, along with a more detailed view of its associations, attributes and methods that the class has.

### 3.1 Main

#### 3.1.1 Associations

##### **AWS**

The 'Main' class contains the AWS class.

##### **Dropbox**

The 'Main' class contains the Dropbox class.

##### **GDriveDownloader**

The 'Main' class contains the GDriveDownloader class.

##### **File\_Selection**

The 'Main' sends Directories back and forth to the File\_Selection class.

##### **Platform\_Selection**

The 'Main' class sends platforms back and forth to the Platform\_Selection class.

##### **Authentication**

The 'Main' class sends field names back and forth to the Authentication class.

#### 3.1.2 Attributes

##### **platforms: String Vector**

The **Main** class contains the attribute **platforms** which is a list of supported cloud platforms that will be passed to the **Platform\_Selection** view so the user can choose a platform.

##### **credentials: String Vector**

The **credentials** attribute stores a list of user credentials to a particular platform. It will contain things like API keys, user names and anything else that a cloud platform will need.

#### 3.1.3 Methods

The Main class does not contain any methods.



## 3.2 class Platform\_template

This class exists to allow future developers to add more cloud platforms to the product. Providing both convention for use, and layout for attribute creation and method creation. Included are basic methods that simplify the authentication. A Method to help format the API return data to comply to the standard specified. A Method to simplify the UI return to parse and complete the download for the user. Other methods are used for helping simplify the class structure for future development.

### 3.2.1 Associations

The association **UI Interface** is used when developing a new cloud platform class to be added into the application. This will also include any necessary inputs for the developer to adjust in order for the class to communicate with the UI.

### 3.2.2 Attributes

**[Platform\_NAME]\_[METHOD]\_[PARAM]:[TYPE]**

This attribute describes the convention for future developers to use when describing attributes to add to new classes. If an attribute is not inherent of a method or [METHOD] as described in the attribute, it is passed in with [NAME]\_[PARAM]. Helping to provide readability to future developers, and indicating how the attributes are being used through the application.

**[\_[Platform\_NAME]]\_user\_download\_path:String**

This attribute needs to be included in every class, and will default to the Users C:/Downloads, unless specifically altered by the user through the application itself.

**[\_[Platform\_NAME]]\_get\_files\_list\_result:List**

This attribute is a recommendation to include in every class as a storage location for the return from the cloud API. This will be utilized in methods included within the template.

**[\_[Platform\_NAME]]\_entries\_to\_download\_list:List**

This attribute is a recommendation to include in every class as storage for the return value of the UI for the cloud service. By using this attribute, the developer can parse the return in the fashion to which they need to allow downloading the user specified data.

**[Platform\_NAME]\_format\_dict:Dictionary**

This attribute is a recommendation to include in every class to store the formatted files into a folder/file dictionary that will allow users to view their files within included directories. Currently the application is not using the method to show folders for users.

**[Platform\_NAME]\_flat\_dict:Dictionary**

This attribute is a recommendation to include in every class to store a 'flattened' Dictionary where only the files and their paths are stored. By using this attribute the user will only see their files without the folders that they are included with.

### 3.2.3 Methods

The three following commands are used for any application that has a supported SDK for OAuth 2 authentication. If this is not present within the platform being added, an authentication method will need to be created separately without the auth\_ methods.

**[Platform\_NAME]\_authentication\_flow(): In:Request**

This function call requires a redirect\_uri that may need to be specified within the app console of the platform being used. This class should return the Flow component of the OAuth2 authentication. This will likely need a request.session to help manage state.

**[Platform\_NAME]\_authentication\_start(): In:Request**

This function will initiate the starting parameters for the OAuth2 authentication to use this capability appropriately, after this method is created, this object will need to be instantiated within the views.py script.

Additionally the `urls.py/urlpatterns` needs to be updated with the following:

```
path('[Platform_NAME]-auth-start', views.[Platform_NAME].[Platform_NAME]_authentication_start)
```

When the user selects the new platform, the Views Index class will need to return a redirect function `redirect('[Platform_NAME]-auth-start')`. Finally this method should instantiate a `redirect_url` to created by the `authentication_flow` method noted previously. The return will need a `django.shortcuts.redirect()` to the `redirect_url` to allow the user to log into the new platform.

#### **[Platform\_NAME]\_authentication\_finish(): In:Request**

This function will finalize the OAuth2 authentication. After this method is created, this object will need to be instantiated within the `views.py` script. Additionally the `urls.py/urlpatterns` needs to be updated with the following:

```
path('[Platform_NAME]-auth-finish', views.[Platform_NAME].[Platform_NAME]_authentication_finish)
```

The `authentication_flow`, may require a `redirect_uri` that includes the full uri to the `[Platform_NAME]-auth-finish`. This function will catch any errors that arise within the OAuth2 authentication flow. This function will also call all necessary function to gather files and folders of the application users platform. The returned object will then need to be formatted into either a dict specified as:

```
{'path':", 'dirs':[], 'files':[]}
```

In which every dir contains the same dict and dict list for subsequent directories. Currently the software requires a `flat_dict`, in which all files are pulled out along with their full path and any required attributes necessary for downloading. The flat dict will be written as:

```
{'file':[]}
```

with all the attributes required for downloading files.

#### **[NAME]\_get\_files\_list()**

This method is used to simplify the API usage for gathering the necessary metadata associated with cloud platform the class is being created for. The return data should be saved local to this class in order to be used by future classes.

#### **[NAME]\_format\_entries\_list():Object JSON**

This method is used to format the returned data provided by the `[Platform_NAME]_get_files_list(self)` method. The format includes a Dictionary defined as

```
{
  "path": "",
  "dirs": [
    {
      "path": "",
      "dirs": [],
      "files": []
    }
  ],
  "files": [
    {
      "path": "",
      "name": ""
    }
  ]
}
```

This Dictionary will be passed to the UI for the user to interact with when selecting what they want to download.

#### **[NAME]\_download\_selected\_entries(in list\_of\_selected\_downloads:Object JSON, [NAME]\_path:String):type**

This method is used to simplify the API usage for downloading. Once a user has selected the items they wish to download, and decided upon the download path. This function will use the cloud platform specified by the class to then download the list of items returned by the UI, into the specified user folder.

### 3.3 Dropbox

This class is used in the Resiliency Platform Tool to assist users in the download of selected Dropbox files. It will connect to the user interface and allow them to log into their Dropbox through a redirect. After logging in, they will have a list view of the contents of their Dropbox. Per Dropbox's api, some files may be locked and not displayed. Using a check box the user will be prompted to select unique files, or to select all. Upon clicking the download button, the files will be sent to the C:/Downloads folder, unless specified by the user to change.

#### 3.3.1 Associations

##### Main

The Dropbox class is contained in 'main', which will pass in authentication information and receive information about files that a user has access to.

##### Dropbox

The Dropbox class calls dropbox using dropbox's Python SDK in order to return information about files and in order to download selected files.

#### 3.3.2 Attributes

##### **dropbox\_api\_key:String**

This attribute will store the value of the Api key generated by the Dropbox application development software in order to interact with dropbox's SDK provided for python. Without the key, the Api calls will fail during the authentication process. This attribute is private and stored within the class itself, generated by an Api key file included within the website.

##### **dropbox\_api\_secret:String**

This attribute will store the value of the Api secret that is generated by the Dropbox application development software. It will be used to interact with the Dropbox SDK for python, allowing easier management of the OAuth2 interface. This attribute is private and stored within the class itself, generated by the Api key file included within the website.

##### **dropbox\_authentication\_auth\_flow:Object**

This attribute is an object that is created by the Dropbox SDK, the formal definition of this object is `dropbox.oauth.DropboxOAuth2FlowNoRedirect()`. This object will be generated, with parameters of both the Api key and the Api secret. Once the object is stored local to the class, the OAuth2 flow can start. For now, the NoRedirect function is being called. When implemented into the website itself, this will be changed to Redirect, allowing the user to log into their Dropbox within the browser.

##### **dropbox\_authentication\_authorize\_url:String**

This attribute stores the url that is generated by the `dropbox_authentication_auth_flow` Object, allowing the user to be redirected to the Dropbox log-in. The result of which will generate a key value to be passed back into the application to complete the validation of the user. Additionally, when the class is integrated into the full website, this will be replaced with the redirect url to allow the same operation for validating the user.

##### **dropbox\_authentication\_auth\_code:String**

This attribute stores the returned authorization code generated from the `dropbox_authentication_authorize_url` link. When prompted the user will enter the string returned by visiting the link. This will be stored locally to allow the class to use the string in the `finish()` method of the Auth Flow Object.

##### **dropbox\_authentication\_oauth\_result:Object**

This attribute will store the object generated by `dropbox_authentication_auth_flow.finish()` call. This attribute is used upon validating the user when creating the main Dropbox interaction object. Allowing the software to interact directly to the users Dropbox storage, which will be used in the below functions to manage Dropbox files, and folders for the user to select and then download.

**dbx:Object**

This attribute is the main Dropbox object. Generated after the authentication, creating access to the users Dropbox. The methods contained within this object will be used in the methods listed below. Which will interact with the Dropbox Api to gather the users files and folders. Then process the download for the objects.

**dropbox\_get\_files\_return:Object**

This attribute stores the result from the Dropbox api method `dropbox.dropbox.Dropbox.files_list_folder()`. Storing the object `dropbox.files.Metadata()` generated by the api call. This data will then be stored individually to allow the user to look through files and select the chosen to be downloaded.

**dropbox\_get\_files\_list\_result:Object List**

This attribute stores the result from the Dropbox api method `dropbox.dropbox.Dropbox.files_list_folder().entries`. Storing the metadata for the various files that are gathered through the Api call. The metadata will include the file name, and extension. The full path to the file, which will be used when the user selects the items in which they want to download, and a parent shared id, which will not be used in this implementation.

**dropbox\_entries\_to\_download\_list:Object List**

This attribute will store the information provided by the user of which files they want to download. The object is the same as the above attributes. Using the `entries.path_lower`, attribute of the generated object, the information will be provided to the download method in order to download the list of files specified by the user.

**dropbox\_download\_path:String="C:/Downloads"**

This attribute will hold the path value used for downloading files. The default for this attribute will be the downloads folder. The user will be given the option to specify the download location. This will update the value to their new destination. If the destination does not exist, the user will be prompted.

**3.3.3 Methods****dropbox\_authentication**

This method implements the authentication pattern user by the Dropbox Python SDK. This method will generate an `authorization_flow` object that will then generate a url link when instantiated. The user will then provide credentials for the link and copy to `authorization_key` to be passed into the program. When integrated into the website, this will cause a redirect to a Dropbox login then load the key without user interaction. After a valid key is provided, given no error is displayed to the user, the key will be stored in a `dbx` attribute, detailed above.

**dropbox\_get\_files\_list**

This method uses the `dbx` attribute generated after authorization of the Dropbox python SDK has completed. Using a method from the `dbx` object `dbx.files_list_folder()` with the arguments, `"/"` to indicate root access, and `recursive=True`, to allow iteration through the entire Dropbox folder system. The call will return an object that will be stored in the `dropbox_get_files_listing`. Which is then parsed, and placed into based off of the `entries` attribute of the object, and stored locally in `dropbox_get_files_list_result`. The method will then iterate through the `entries`, attribute of the object returned, and store them in a List to later be manipulated by the class.

**dropbox\_format\_entries\_list**

This function uses the local `dropbox_get_files_list_result` to sort the data and reorganize it. The ordering will be a hierarchal order, listing folders and the included subfolders and files. This will then be stored in JSON format to be used by the website to display the data in an accordion list which will expand the structures to display subfolders and files included within selected folder. Providing an easier to manage interface for the user.

**dropbox\_select\_entries\_to\_download**

This method will be used to store the indices selected by the user for the files they wish to download. Using key values generated through the UI to then create a new list `Dropbox_entries_to_download_list` in which the selections will be saved local to the class.

**dropbox\_download\_selected\_entries**

This method will use the `dbx.files.download_to_file()`. It will use the locally saved attribute `dropbox_entries_to_download_list` to iterate through, calling the Api using the user defined `dropbox_download_path`. The function will then download all the files within the list, to the location specified. The user will be prompted with an error if a file is not downloadable, or if a download failed for other reasons.

## 3.4 AWS

The class ‘aws’ will be the primary handler of the application’s interaction with Amazon Web Services (AWS), this includes Amazon Elastic Compute Cloud (EC2) and Amazon Simple Storage Service (S3). The application can use the ‘aws’ class to get a list of all available files as well as the ability too download said files, given that the user has access to do both.

### 3.4.1 Associations

#### Main

The AWS class is contained in ‘Files’ view, which will pass in authentication information and receive information about files that a user has access to as well as give them the ability to download files.

#### AWS

The AWS Class makes calls to Amazon Web Services using Amazon’s boto3 Python module.

### 3.4.2 Attributes

#### **aws\_ec2\_client\_flow: Object boto3.client()**

This is an instance of the EC2 client, generated by the authentication() method using boto3.client(). Uses access\_key, access\_key\_id, and the region (by default, the region is set to ‘us-west-2’).

#### **aws\_s3\_client\_flow: Object boto3.client()**

This is an instance of the S3 client, generated by the authentication() method using boto3.client(). Uses access\_key, access\_key\_id, and the region (by default, the region is set to ‘us-west-2’).

#### **aws\_s3\_resource\_flow: Object boto3.resource()**

This is an instance of the S3 resource, generated by the authentication() method using boto3.resource() method. Uses access\_key, access\_key\_id, and the region (by default, the region is set to ‘us-west-2’).

#### **aws\_download\_path: String=“C:/Downloads”**

This attribute will hold the path value used for downloading files. The default for this attribute will be the downloads folder. The user will be given the option to specify the download location. This will update the value to their new destination. If the destination does not exist, the user will be prompted.

### 3.4.3 Methods

#### **aws\_init**

This method passes a AWS access key and AWS access key\_id to Amazons boto3 module, which is used to interact with AWS through python. The key and keyID will be used to create three class attributes that will be used throughout the aws class. These are s3\_client, ec2\_client, s3\_resource.

#### **aws\_get\_image\_list: json**

Using the \_ec2\_client defined in the \_\_init\_\_ method, the ‘get\_image\_list’ method will be used to retrieve all available files that the user has access to. This will use the client.describe\_images method as defined in the boto3 specifications. This method is used by the Files view.

#### **aws\_get\_buckets: json**

Using the \_s3\_client defined in the \_\_init\_\_ method, the ‘get\_buckets’ method will be used to retrieve all available buckets that the user has access to.

#### **aws\_list\_images\_in\_bucket: json**

Uses \_s3\_resource to list all images contained within a bucket, this method will return a JSON dict.

#### **aws\_export\_to\_bucket**

This method is optionally used and gives a developer the ability to export images to a bucket.

#### **aws\_download\_images**

This method will use the \_s3\_resource.download\_file() method within a for loop specified within the Files view to download all selected images.

## 3.5 GDriveDownloader

The GDriveDownloader class is responsible for authenticating to Google, acquiring a list of files from Google Drive, and downloading the file. This class is able to take the authentication token and the built service as an input on creation.

### 3.5.1 Associations

**Main** The GDriveDownloader class is contained in 'main', which will pass in authentication information and receive information about files that a user has access to.

**Google** The GDriveDownloader Class makes calls to Google.

### 3.5.2 Attributes

#### **GDriveDownloader\_creds**

The attribute GDriveDownloader\_creds holds the authentication token.

#### **GDriveDownloader\_SCOPES**

The attribute GDriveDownloader\_SCOPES holds the scope of what the application can access from Google.

#### **GDriveDownloader\_service**

The attribute GDriveDownloader\_service holds the built service.

#### **GDriveDownloader\_file\_List**

The attribute GDriveDownloader\_file\_List holds the list of files from Google Drive.

#### **GDriveDownloader\_json**

The attribute GDriveDownloader\_json holds the JSON needed by the UI.

#### **GDriveDownloader\_files\_to\_download**

The attribute GDriveDownloader\_files\_to\_download holds a list of dictionaries.

### 3.5.3 Methods

#### **GDriveDownloader\_authentication**

This method handles the authentication to Google and gets the authentication token. It saves the token in GDriveDownloader\_creds.

#### **GDriveDownloader\_save\_Token**

This method saves the token to a file.

#### **GDriveDownloader\_load\_Token**

This method loads the token from a file.

#### **GDriveDownloader\_build\_Service**

This method builds the service from the scope and the authentication token.

#### **GDriveDownloader\_\_download\_File**

This method loops over GDriveDownloader\_files\_to\_download and downloads the files to the working dictionary.

#### **GDriveDownloader\_\_get\_Files**

This method queries Google Drive for the files stored in the user's Google account. It will also create the JSON for the UI and store it in GDriveDownloader\_json.

#### **GDriveDownloader\_add\_file\_to\_download**

This method takes a dictionary with keys 'name' and 'id' as input and appends it to over GDriveDownloader\_files\_to\_download.

#### **GDriveDownloaded\_authentication\_flow**

This method creates the authentication flow using the secret from google.

**GDriveDownloaded\_authentication\_start**

This method get the url to redirect to google's login page.

**GDriveDownloaded\_authentication\_finish**

This method gets a json with the access token, turns it into a Credential object, builds the service and gets the files. It then redirects to the files page.



## 3.6 Platform\_Selection

The **Platform\_Selection** class is a view in the MVC design pattern that will present the user with a drop-down menu that contains the available platforms for downloading.

### 3.6.1 Associations

#### Main

The Platform\_Selection class is contained in the Main class.

### 3.6.2 Attributes

#### platforms: String Vector

The **platforms** attribute is a list of the strings containing the cloud platforms that are supported by the application for the user to download.

### 3.6.3 Methods

#### submit(OUT platform: string)

The method **submit** will pass back the **Main** class (the controller) a string containing the platform that the user selected. The **Main** class will use this information for the next view classes.

## 3.7 Authentication

The **Authentication** class is a view in the MVC design pattern that will retrieve the user's credentials for a cloud platform.

### 3.7.1 Associations

#### Main

The Authentication class is contained in the Main class.

### 3.7.2 Attributes

#### fields: String Vector

The **fields** attribute is a list of field names that the application will create and prompt the user for. For example, there could be a *user name* field and a *password* field or a *client\_secret* token or any other variable number of fields needed that the **Authentication** class will need to present to the user.

### 3.7.3 Methods

#### authenticate(IN: user\_input: String Vector): Boolean

The method **authenticate** takes in the filled in fields from the user and sends those back to the **Main** class. The **Main** class will reply with *True* if the authentication was a success, allowing the application to move onto the next view. If the credentials did not work for authentication then *False* will be returned and the **Authenticate** view will re-prompt the user for their credentials.

## 3.8 File\_Selection

The class **File\_Selection** is a view in the MVC design pattern that will display the available files to download to the user and allow the user to select the files.

### 3.8.1 Associations

#### Main

The File\_Selection class is contained in the Main class.

### 3.8.2 Attributes

#### available\_files: Directory

The attribute **available\_files** contains the files that on the cloud platform to download.

### 3.8.3 Methods

#### download(INOUT: dirs: Directory)

The **download** method will send a message back to the **Main** class (the controller) containing a **Directory** class with just the files that the user has selected to download.

## 4 User Interface

The User interface section will over the actions that a user will be able to take, explaining all menus, buttons, pull down menus, selections, etc. This section will also describe actions that system administrators will need to take to start and stop the application.

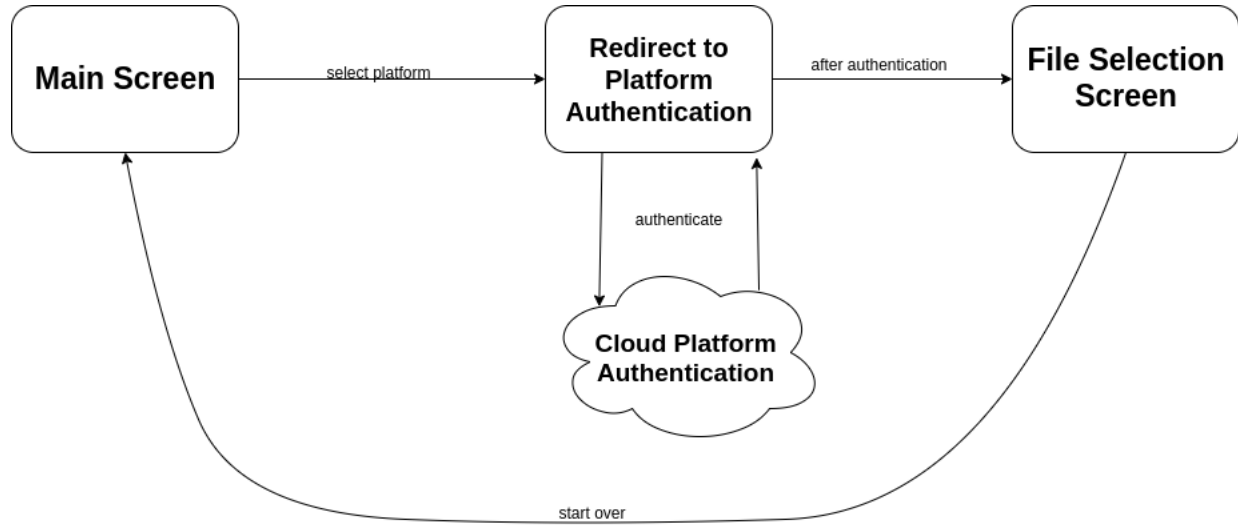


Figure 7: The workflow for the user

### 4.1 User Interaction

The user will interact with the application through the web interface. They will have several screens (views) to navigate through. Those will include the main screen to select the platform to download from (AWS, Dropbox, Google Drive, etc) and then will be presented with a view to enter their credentials. Once their credentials are entered they will be presented with the final view which contains their directory structure and contents and they will be able to multi-select which of those contents they would like to download.

For more information, readers can refer to the [Prototype Screenshots for the Downloader Application](#)[3] which will cover the work flow for the user and screenshots of the preliminary screens.

#### 4.1.1 Platform Selection Screen

The first page of the application that the user will come across is a page to select which platform they will be viewing and downloading files from. This page will contain a *pull down* menu containing each of the cloud platforms that the application supports. Once the user has selected the one they want, they will press the *submit* button which will bring them to the next page.

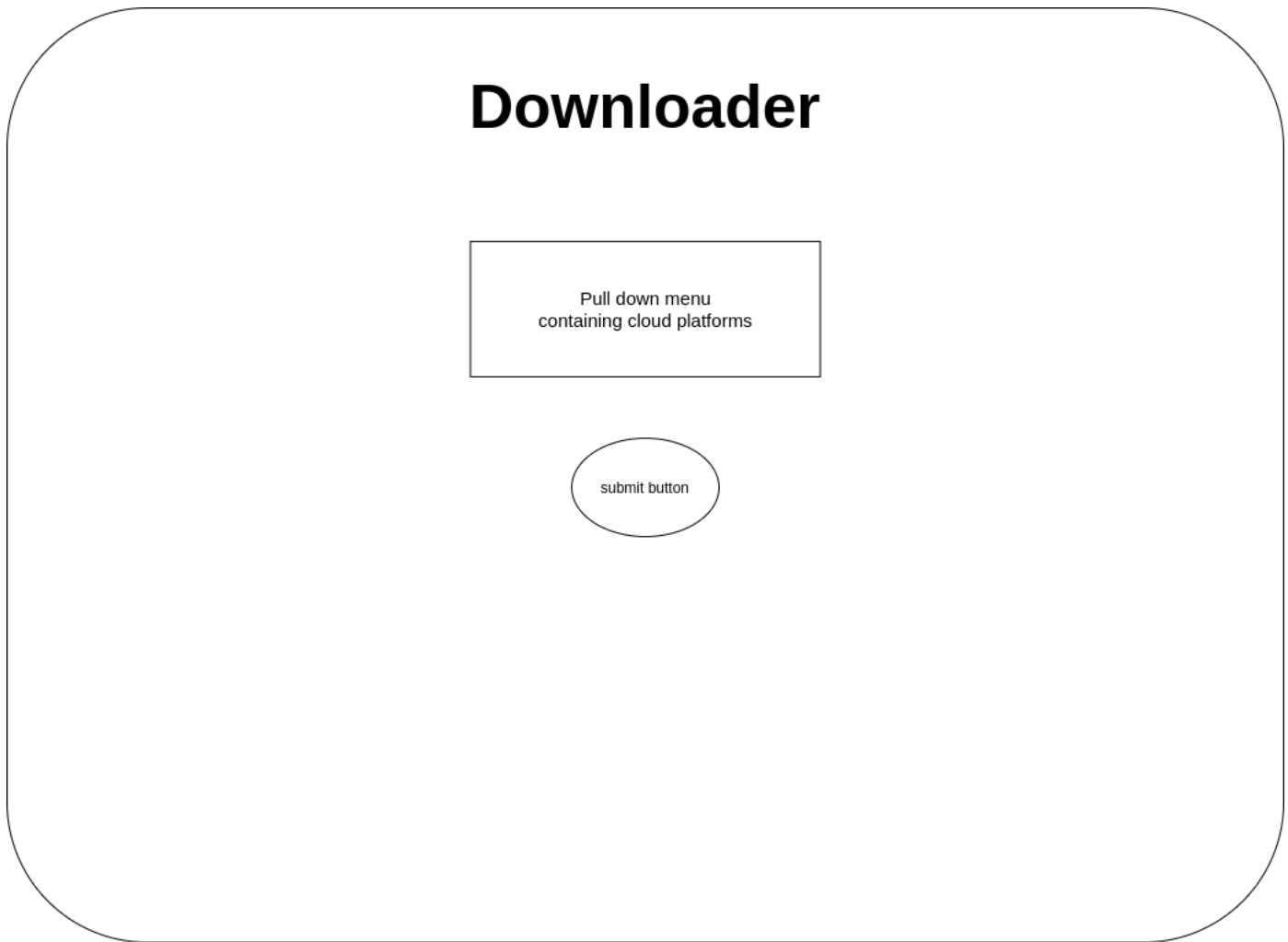


Figure 8: The first screen of the application.

#### 4.1.2 Authentication Screen

After the user has selected their platform from the *platform selection screen*, the application will determine what kind of authorization information it needs from the user and prompt the user to enter this information. This will include a user name, and either a password, access token or both. Once the user enters their credentials for their cloud account they will press the *ok* button to allow the application to authenticate to the cloud platform and read their files. They will be redirected to the final screen at this point.

#### 4.1.3 File Selection Screen

On the final screen of the application, after the user has selected and authenticated to a cloud platform they will be presented with the files and directory structure that they have on that platform. The user will be able to navigate their directory structure from this screen as well as multi-select which files and directories they would like to download.

Once the user is ready to download their files and directories they will click on the *download* button which will initiate the downloading of their files to where the user's web browser directs downloads to. Once the download is complete the user can exit the browser window, select the *start over* button to go back to the *Platform Selection Screen* or select a different set of files to download.

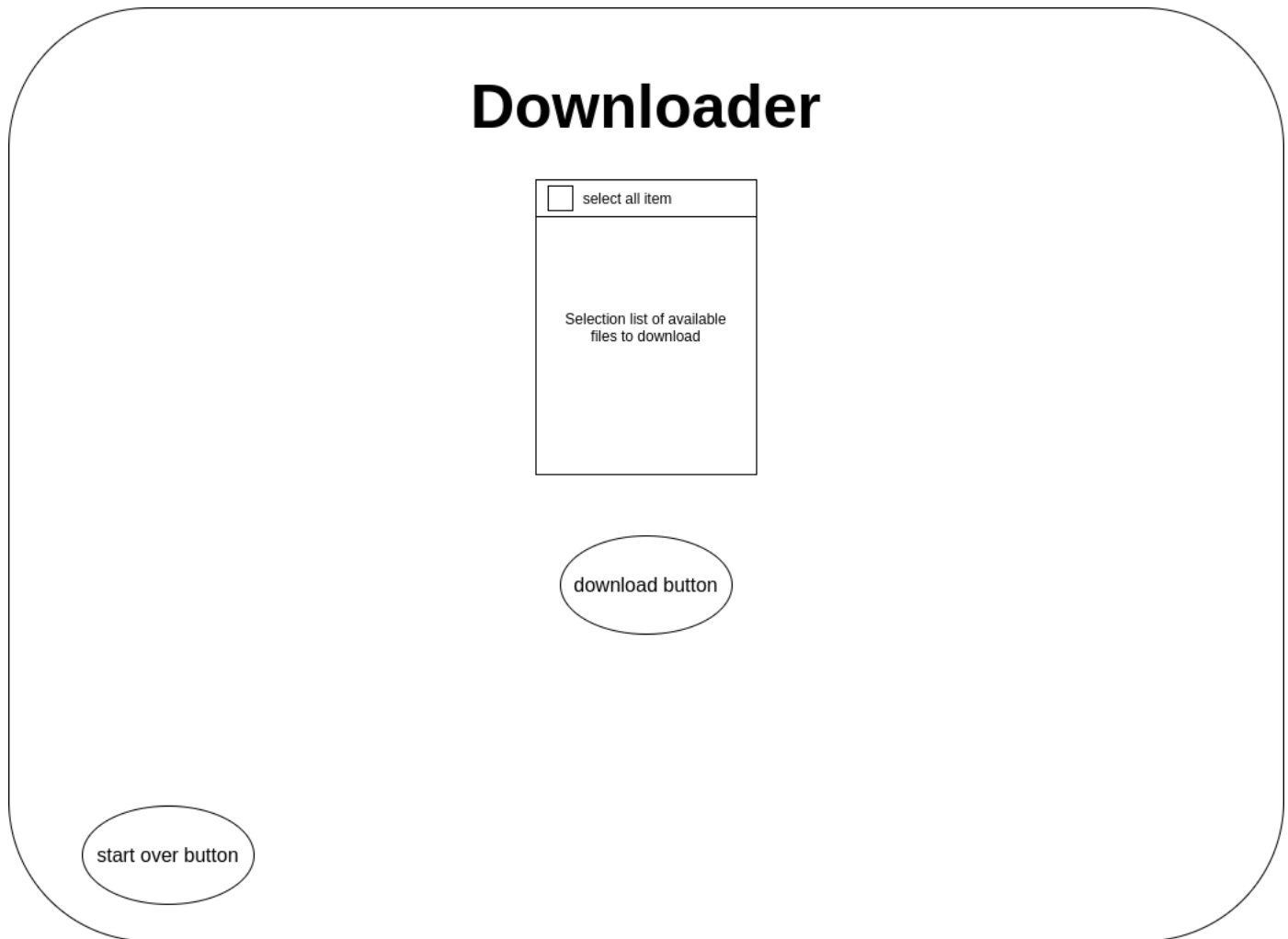


Figure 9: The screen to view and download files.

## 4.2 Administrator Interaction

The system administrator will interact with the application by using the built-in Django commands on the server that the application will run on. They will mainly only be starting and stopping the web application through this interface. Django does have a built-in web administration page but this application will not need to take advantage of this functionality.

### 4.2.1 Start Application

To start the application the system administrator will use the Django created *manage.py* file to invoke the application on its default port of *8000*.

```
$ python manage.py runserver
```

If the administrator would like to expose the webserver beyond *localhost* and let other devices connect or use a different port then they will pass an argument with the sources and new port. For example, to let any source connect to the server from port *8080* the administrator would type:

```
$ python manage.py runserver 0:8080
```

The “0” before the *colon* is shorthand for *0.0.0.0* which will allow any source to connect.

#### **4.2.2 Stop Application**

Since the web server will be a foreground process in the terminal, the web administrator will press *ctrl + c* to kill the process or any of the other ten thousand ways to terminate a process on a \*nix system.

## 5 Information Repositories

The application uses several different information repositories. There will be one for each platform supported as outlined in the **Requirements Specification** [2]. For example, AWS storage will be an information repository as well as Dropbox. The user's local hard drive will also be an information repository during development and then once the client takes delivery their internal storage will become the information repository.

### 5.1 Cloud Platform Repository

The Cloud Repositories are the storage on the cloud for each platform that the user has files stored on. These files can be any type but will typically be an operating system ISO for the client's use case. These repositories can contain any arbitrary number of files and folders stored within this repository, the users will be able to navigate these through the application and select which ones to download. This information repository will be read only for the application and the user, they will not be deleting or adding contents to the cloud platform.

### 5.2 Local Storage or Internal Infrastructure

The second type of information repository that the application will contain is the location that the application will download the user selected files from. This will be the user's local storage during the development cycle and in production the client's internal infrastructure will be the repository for these files. This information repository will not be read from, it will only be written too. If the user would like to alter these files then they will need to use another application since the *Downloader* application does not include the user's hard drive in it's scope.



## References

- [1] Ryan Breitenfeldt, Noah Farris, Trevor Surface, Kyle Thomas. *Project Plan*. [*Project Plan for Downloader Application*] 2019.
- [2] Ryan Breitenfeldt, Noah Farris, Trevor Surface, Kyle Thomas. *Requirements Specification*. [*Requirements Specification for Downloader Application*] 2019.
- [3] Ryan Breitenfeldt, Noah Farris, Trevor Surface, Kyle Thomas. *Prototype*. [*Prototype Screenshots for Downloader Application*] 2019.
- [4] Cypherpath.com. (2019). Cypherpath, Inc. [online] Available at: <https://www.cypherpath.com/> [Accessed 21 Oct. 2019].
- [5] Amazon Web Services, Inc. (2019). Amazon Web Services (AWS) - Cloud Computing Services. [online] Available at: <https://aws.amazon.com/> [Accessed 10 Oct. 2019].
- [6] Python.org. (2019). Welcome to Python.org. [online] Available at: <https://www.python.org/> [Accessed 6 Nov. 2019].
- [7] Django project.com. (2019). The Web framework for perfectionists with deadlines | Django. [online] Available at: <https://www.djangoproject.com/> [Accessed 6 Nov. 2019].