Design Document

Captain CyBeard: Neil Before Us

Ryan Breitenfeldt | Noah Farris
Trevor Surface | Kyle Thomas

Febraury 25, 2020

Washington State University Tri-Cities
CptS 421 Software Design Project 1

Contents

List of Figures

Revision History

Revision	Date	Author(s)	Description
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 ***NUMBER OF CLASSES*** in a model, view, controller architecture. The following UML diagrams will show the classes and their relationships with each other, along with their attributes and methods.

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 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.

3.1.1 Associations

Is contained in UI

Makes calls to Dropbox

3.1.2 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 mamangement 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 loging. 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 reutrned 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 stroed 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 attibute 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 teh 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.

3.1.3 Methods

dropbox_authentication:void
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:void**
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 and object that will be stored in the dropbox_get_files_listing. Which is then parced, 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, atribute of the object returned, and store them in a List to later be manipulated by the class. **dropbox_format_entries_list:void**
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 inlcuded 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 structues to display subfolders and files inlcuded within selected folder. Providing an easier to manage interface for the user. **dropbox_select_entries_to_download:void**
This method will be used to store the endicies 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:void**
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.

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.

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.

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.

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] Djangoproject.com. (2019). The Web framework for perfectionists with deadlines | Django. [online] Available at: https://www.djangoproject.com/ [Accessed 6 Nov. 2019].