Cloud Platform Downloader

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

Project Plan

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

List of Figures

Revision History

Revision	Date	Author(s)	Description
2.0	12.08.2019	KT	Added new gantt chart
1.1	11.12.2019	KT	Performed Edits
1.0	09.27.2019	RB NF TS KT	Completed Document
0.5	09.27.2019	RB	Filled in Estimate section
0.4	09.26.2019	KT	Filled in Approach section
0.3	09.24.2019	RB NF TS KT	Filled in scope and added diagram
0.2	09.19.2019	KT	Filled in Introduction Section
0.1	09.12.2019	KT	Document Creation

1 Introduction

This document is a project plan for developing a Django Web Application that allows Cypherpath users to enter a URL for online Virtual Machines and select which Virtual Machines will be downloaded onto Cypherpath's servers. The purpose of the project plan is to provide a roadmap for Cypherpath and the software development team of the development process and help keep track of the progress.

Cypherpath is a company that provides a product called *Resiliency Platform Tool* that stores virtual networks, machines, configurations and more enabling their customers to quickly recover from cyber attacks such as ransomware. Currently, customers need to manually download their virtual disk images from the various cloud platforms they are subscribed to (such as VMware or Amazon Web Services) and then upload them to Cypherpath's platform. This application will allow customers to have their virtual disk images transferred directly into the Resiliency Platform.

Subsequent sections of this project plan will cover the scope of the project, the software engineering approach that will be used for the project and an estimate for how long the project will take broken by task in the form of a Gantt Chart.

2 Scope

The project is to develop a Python-Django web application that allows a user that is logged into the application to enter a URL that points to one of several possible cloud based VM platforms and be presented with the authentication for that platform. After the user enters their credentials for the VM platform they will be presented with the virtual disk images they have on their account that are available to download. These virtual disk images are Cypherpath's customers virtualized appliances that are used for their business tasks.

The application will present relevant information to the user such as the directory structure, names of the VM images, and a way to select which files and folders to download to their local machine.

The first platform to focus on for interacting with will be VMware. Time permitting, other platforms such as Amazon Web Services (AWS), Citrix, Google Drive and Dropbox will be added. The application will have a modular design with both cloud platforms and authentication mechanisms so more can be added in the future.

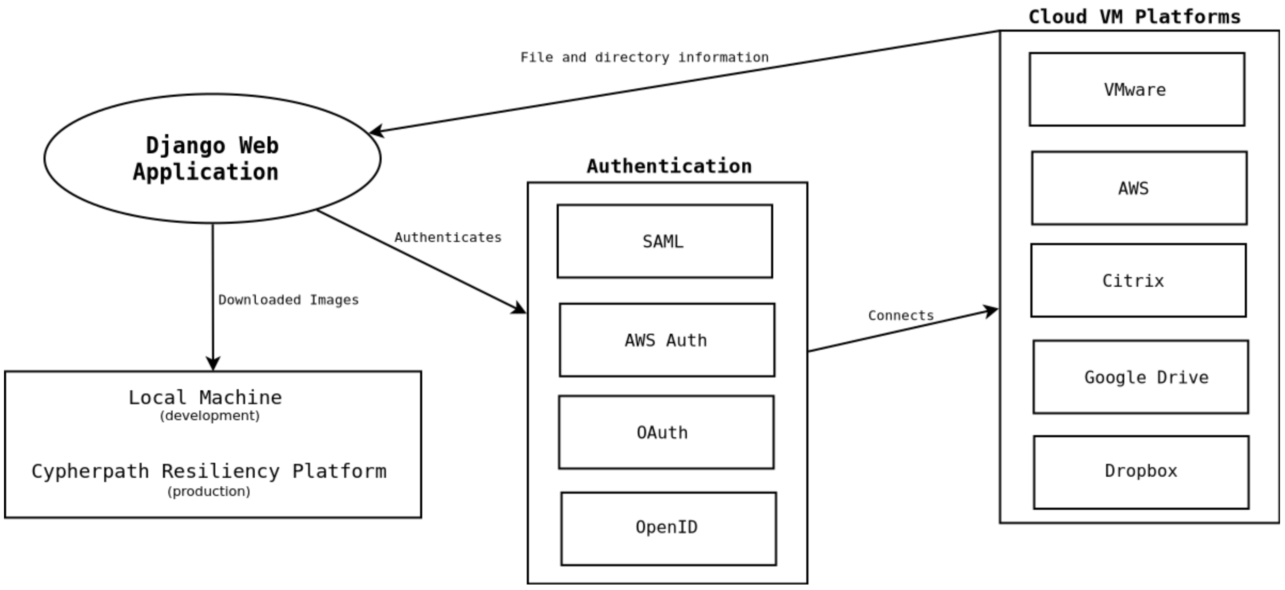


Figure 1: The application environment

3 Approach

The software engineering approach that will be used for this project is the **Agile Scrum** approach. This iterative approach will ensure that the software development team will remain on track and that Cypherpath will receive the maximum amount of value for their time.

The project will be implemented as a Python-Django web application, Python version 3+ and Django version 2+ will be used. The Python-Django framework provides the tools and environment needed to develop and test a web application such as a lightweight web server and SQLite. For version control the software development team will be using git with a remote repository hosted on Github. Once the project is completed, Cypherpath will host the web application within their infrastructure.

4 Estimate

The project is estimated to take two academic semesters in order to complete collecting requirements, designing, implementation and testing. The estimated date to complete the project is Wednesday April 4th, 2020. The project is estimated to take 375 hours to complete.

The following page will contain a breakdown of the tasks and estimated time to complete them for the project.

Untitled Gantt Project

May 4, 2020

http://

Project manager	
Project dates	Aug 27, 2019 - May 5, 2020
Completion	0%
Tasks	60
Resources	0

Tasks

Name	Lead	Begin date	End date	Days	Work Hours
1 Project plan		8/29/19	9/27/19	30	22
1.1 understand problem domain	All	8/29/19	8/29/19	4	2
1.2 Determine scope	Trevor	9/2/19	9/4/19	3	3
1.3 Identify Tasks	Ryan	9/5/19	9/6/19	4	2
1.4 Estimate time to perform tasks	Ryan	9/9/19	9/12/19	4	4
1.5 Schedule Tasks	Ryan	9/13/19	9/16/19	4	2
1.6 Write project plan	Kyle	9/17/19	9/23/19	7	5
1.7 Review Project plan	All	9/24/19	9/26/19	3	1
1.8 Present Plan	Noah	9/27/19	9/27/19	1	1
2 Requirments		8/27/19	12/11/19	107	27
2.1 requirment meetings	All	8/27/19	12/4/19	100	15
2.2 determine fesablity of requirments	Trevor	9/6/19	9/6/19	3	3
2.3 determine software needs	Trevor	9/9/19	9/9/19	1	2
2.4 write requirements document	Kyle	12/2/19	12/6/19	5	5
2.5 review requirements document	All	12/9/19	12/10/19	2	2
2.6 present requirements	Noah	12/11/19	12/11/19	1	1
3 Design		2/5/20	3/6/20	11	9
3.1 plan design		2/5/20	2/24/20	50	66
3.1.1 design UI	Kyle	2/5/20	2/11/20	6	6
3.1.2 design AWS interface		2/7/20	2/17/20	14	20
3.1.2.1 autentication	Noah	2/7/20	2/14/20	7	10
3.1.2.2 get files	Noah	2/10/20	2/17/20	7	10
3.1.3 design Google interface		2/13/20	2/17/20	8	10
3.1.3.1 autentication	Ryan	2/13/20	2/17/20	4	5
3.1.3.2 get files	Ryan	2/13/20	2/17/20	4	5
3.1.4 design Dropbox interface		2/13/20	2/17/20	8	10
3.1.4.1 autentication	Trevor	2/13/20	2/17/20	4	5

Tasks

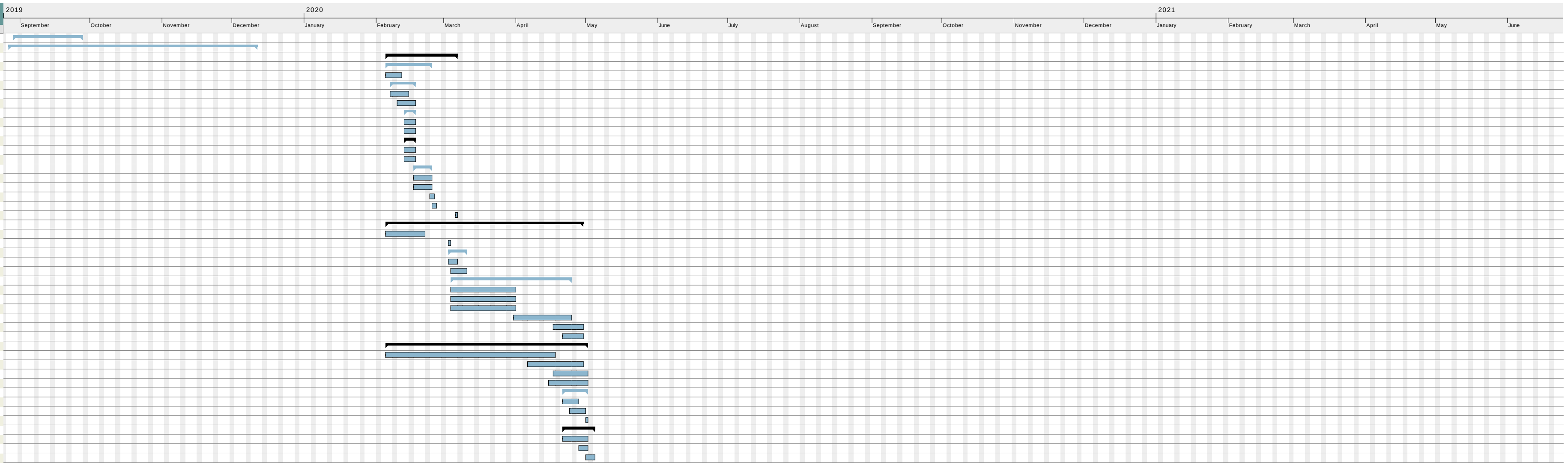
Name	Lead	Begin date	End date	Days	Work Hours
3.1.4.2 get files	Trevor	2/13/20	2/17/20	4	5
3.1.5 design VMware interface		2/17/20	2/24/20	14	20
3.1.5.1 authentication	all	2/17/20	2/24/20	7	10
3.1.4.2 get files	all	2/17/20	2/24/20	7	10
3.2 write document	Kyle	2/24/20	2/25/20	3	6
3.3 Review design document	All	2/25/20	2/26/20	2	2
3.4 present design document	Noah	3/6/20	3/6/20	1	1
4 implementation		2/5/20	4/29/20	94	231
4.1 learn Django and APIs	All	2/5/20	2/21/20	20	60
4.2 initional setup django	Ryan	3/3/20	3/3/20	1	1
4.3 make Django front end		3/3/20	3/10/20	9	25
4.3.1 login page	Kyle	3/3/20	3/6/20	3	10
4.3.2 downloads page	Kyle	3/4/20	3/10/20	6	15
4.4 django backend		3/4/20	4/24/20	60	140
4.4.1 aws	Noah	3/4/20	3/31/20	27	30
4.4.2 google docs	Ryan	3/4/20	3/31/20	27	30
4.4.3 dropbox	Trevor	3/4/20	3/31/20	27	30
4.4.4 File maneger	All	3/31/20	4/24/20	25	20
4.5 code review	All	4/17/20	4/29/20	2	3
4.6 code fixing	All	4/21/20	4/29/20	8	2
5 Testing		2/5/20	5/1/20	96	30
5.1 design tests	Noah	2/5/20	4/17/20	73	10
5.2 test code	Trevor	4/6/20	4/29/20	23	10
5.3 write up documentation	Kyle	4/17/20	5/1/20	14	5
5.4 Bug fixing	All	4/15/20	5/1/20	16	5
6 User's guide		4/21/20	5/1/20	8	9

Tasks

Name	Lead	Begin date	End date	Days	Work Hours
6.1 write user guide	Kyle	4/21/20	4/27/20	6	6
6.2 review user guide	All	4/24/20	4/30/20	6	2
6.3 present user guide	Noah	5/1/20	5/1/20	1	1
7 project summery		4/21/20	5/4/20	9	11
7.1 Write project summery	Kyle	4/21/20	5/1/20	11	8
7.2 review poject summery	All	4/28/20	5/1/20	3	2
7.3 present	Noah	5/1/20	5/4/20	1	1

Gantt Chart

Name	Lead	Begin date	End date	Days	Work Hours
1 Project plan		8/29/19	9/27/19	30	22
2 Requirments		8/27/19	12/11/19	107	27
3 Design		2/5/20	3/6/20	11	9
3.1 plan design		2/5/20	2/24/20	50	66
3.1.1 design UI	Kyle	2/5/20	2/11/20	6	6
3.1.2 design AWS interface		2/7/20	2/17/20	14	20
3.1.2.1 autentication	Noah	2/7/20	2/14/20	7	10
3.1.2.2 get files	Noah	2/10/20	2/17/20	7	10
3.1.3 design Google interfa…		2/13/20	2/17/20	8	10
3.1.3.1 autentication	Ryan	2/13/20	2/17/20	4	5
3.1.3.2 get files	Ryan	2/13/20	2/17/20	4	5
3.1.4 design Dropbox inter…		2/13/20	2/17/20	8	10
3.1.4.1 autentication	Tre…	2/13/20	2/17/20	4	5
3.1.4.2 get files	Tre…	2/13/20	2/17/20	4	5
3.1.5 design VMware interf…		2/17/20	2/24/20	14	20
3.1.5.1 authentication	all	2/17/20	2/24/20	7	10
3.1.4.2 get files	all	2/17/20	2/24/20	7	10
3.2 write document	Kyle	2/24/20	2/25/20	3	6
3.3 Review design document	All	2/25/20	2/26/20	2	2
3.4 present design document	Noah	3/6/20	3/6/20	1	1
4 implementation		2/5/20	4/29/20	94	231
4.1 learn Django and APIs	All	2/5/20	2/21/20	20	60
4.2 initional setup django	Ryan	3/3/20	3/3/20	1	1
4.3 make Django front end		3/3/20	3/10/20	9	25
4.3.1 login page	Kyle	3/3/20	3/6/20	3	10
4.3.2 downloads page	Kyle	3/4/20	3/10/20	6	15
4.4 django backend		3/4/20	4/24/20	60	140
4.4.1 aws	Noah	3/4/20	3/31/20	27	30
4.4.2 google docs	Ryan	3/4/20	3/31/20	27	30
4.4.3 dropbox	Tre…	3/4/20	3/31/20	27	30
4.4.4 File maneger	All	3/31/20	4/24/20	25	20
4.5 code review	All	4/17/20	4/29/20	2	3
4.6 code fixing	All	4/21/20	4/29/20	8	2
5 Testing		2/5/20	5/1/20	96	30
5.1 design tests	Noah	2/5/20	4/17/20	73	10
5.2 test code	Tre…	4/6/20	4/29/20	23	10
5.3 write up documentation	Kyle	4/17/20	5/1/20	14	5
5.4 Bug fixing	All	4/15/20	5/1/20	16	5
6 User's guide		4/21/20	5/1/20	8	9
6.1 write user guide	Kyle	4/21/20	4/27/20	6	6
6.2 review user guide	All	4/24/20	4/30/20	6	2
6.3 present user guide	Noah	5/1/20	5/1/20	1	1
7 project summery		4/21/20	5/4/20	9	11
7.1 Write project summery	Kyle	4/21/20	5/1/20	11	8
7.2 review poject summery	All	4/28/20	5/1/20	3	2
7.3 present	Noah	5/1/20	5/4/20	1	1



Resources Chart

Name	Default role	2019 September	October	November	December	2020 January	February	March	April	May	June	July	August	September	October	November	December	2021 January	February	March	April	May	June

Prototype Screenshots for Downloader Application

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

List of Figures

1 Introduction

The following figures are screenshots for the *Downloader* Prototype. The prototype was created with HTML, CSS and Javascript and opened with a web browser (a web server was not used to serve the files). The *Downloader* application is an application where users will enter a URL that goes to their external cloud service and then the application will have the users authenticate to their external cloud service. After authentication, the user can navigate their directories on the cloud platform and select which files to download. The screenshots show this process visually.

Figures 3 and 4 must be two separate steps due to the limitations of vanilla Javascript, however when the application is actually implemented with Django these two inputs from the user will be in the same window.

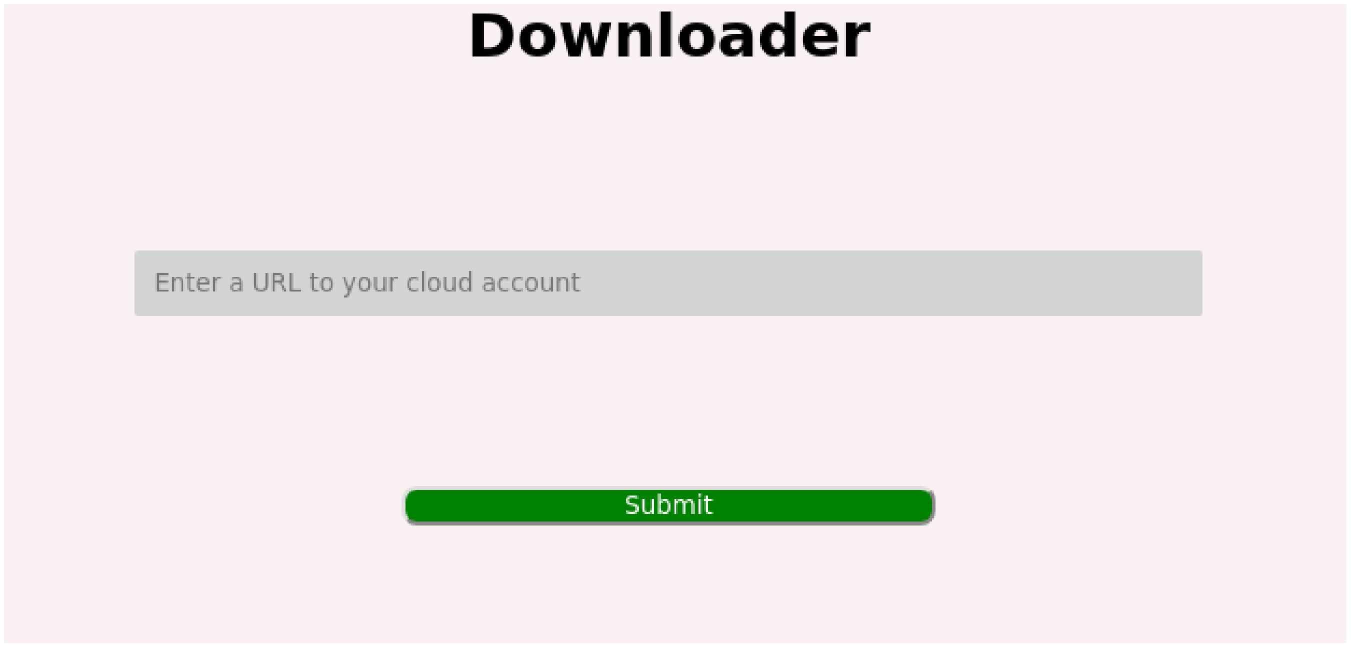


Figure 1: First page, ready for a url to a cloud platform.

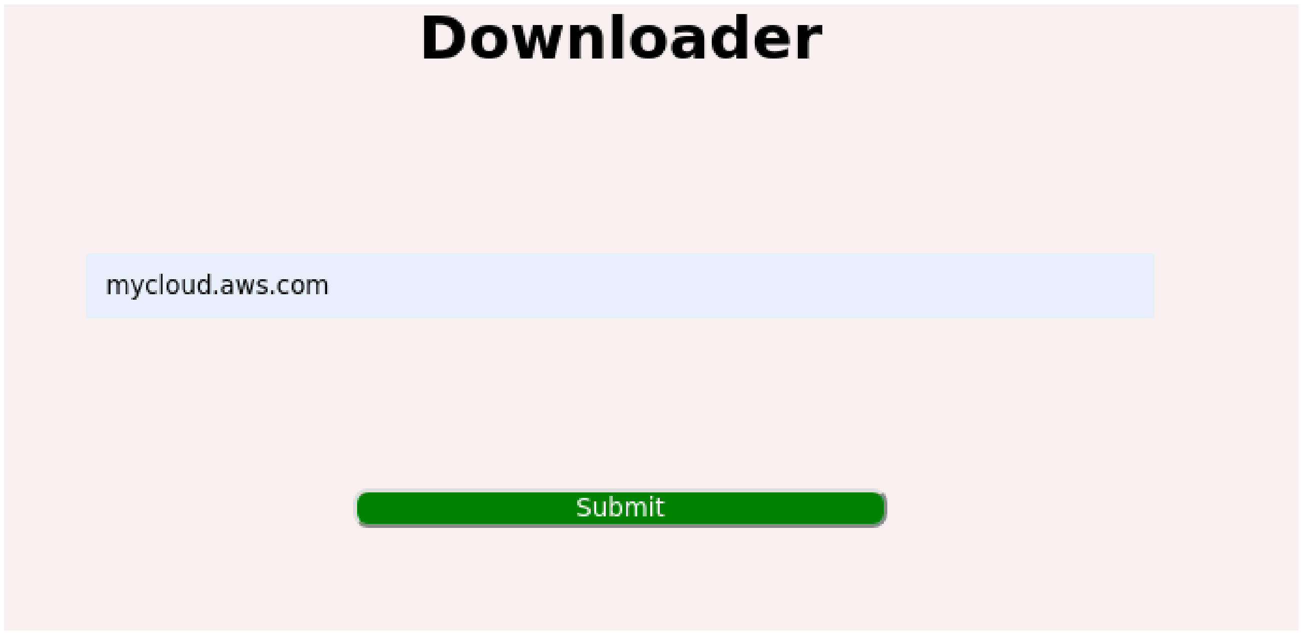


Figure 2: URL entered that goes to cloud account.

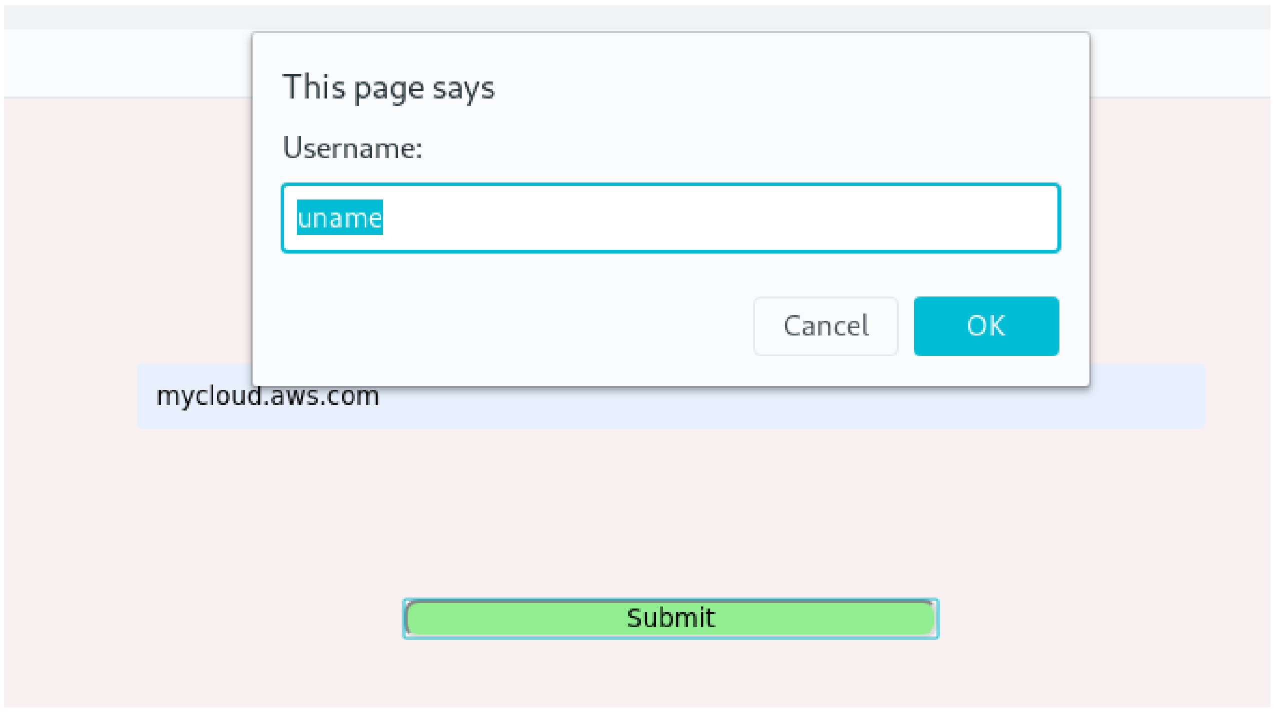


Figure 3: Username to cloud platform entered.

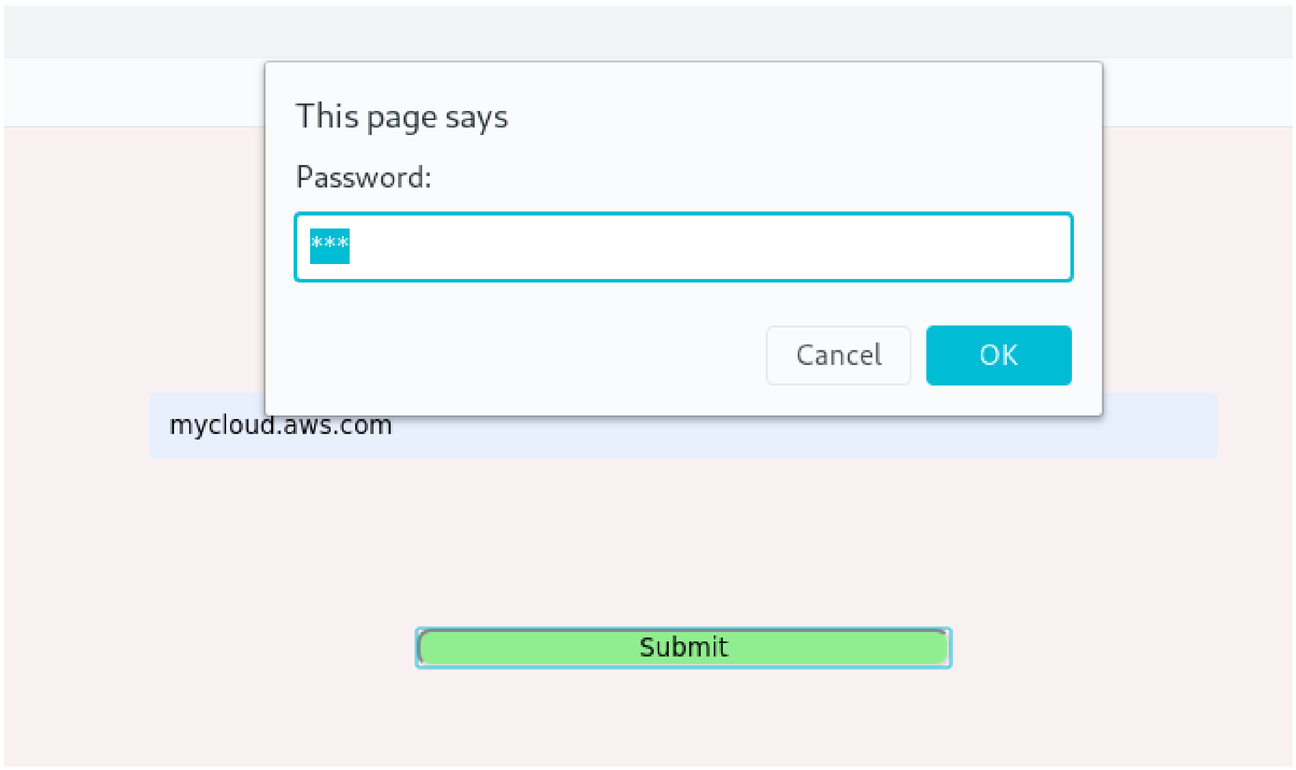


Figure 4: Password to cloud platform entered. Note: Javascript doesn't allow for a popup with two text boxes in the same window or two windows at the same time. The actual implementation will include both the username and password entry boxes in the same popup window.

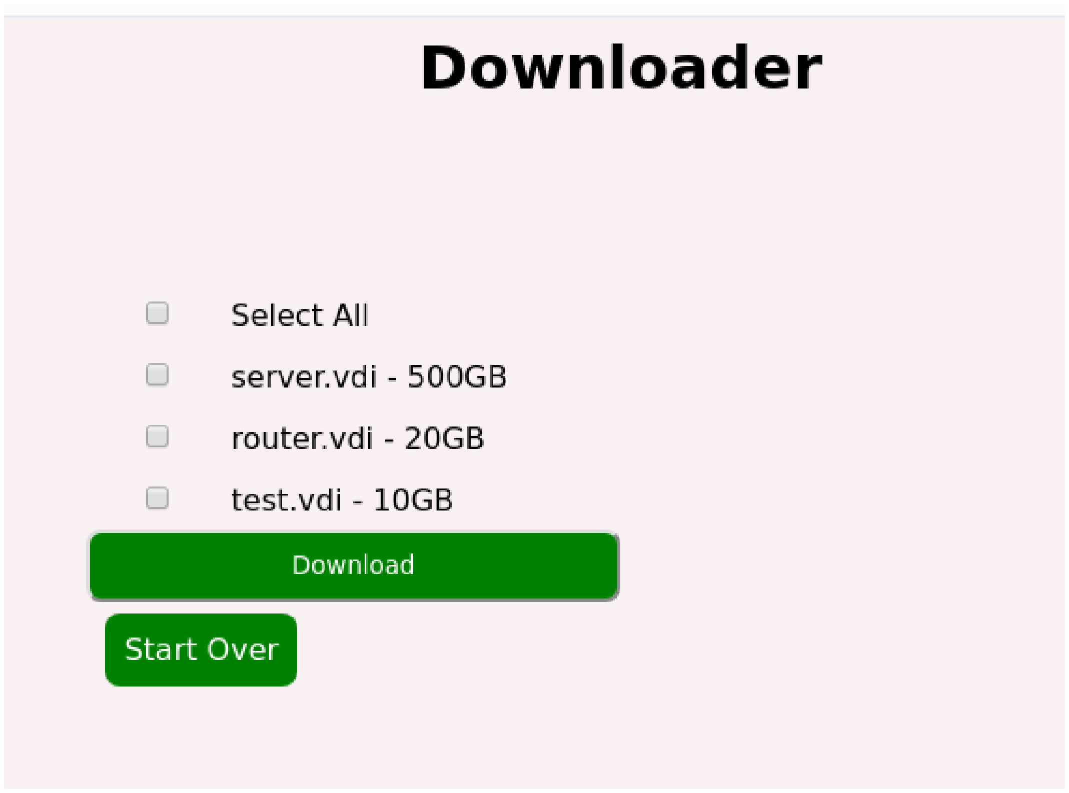


Figure 5: Files available for downloading in the root directory of the cloud storage.

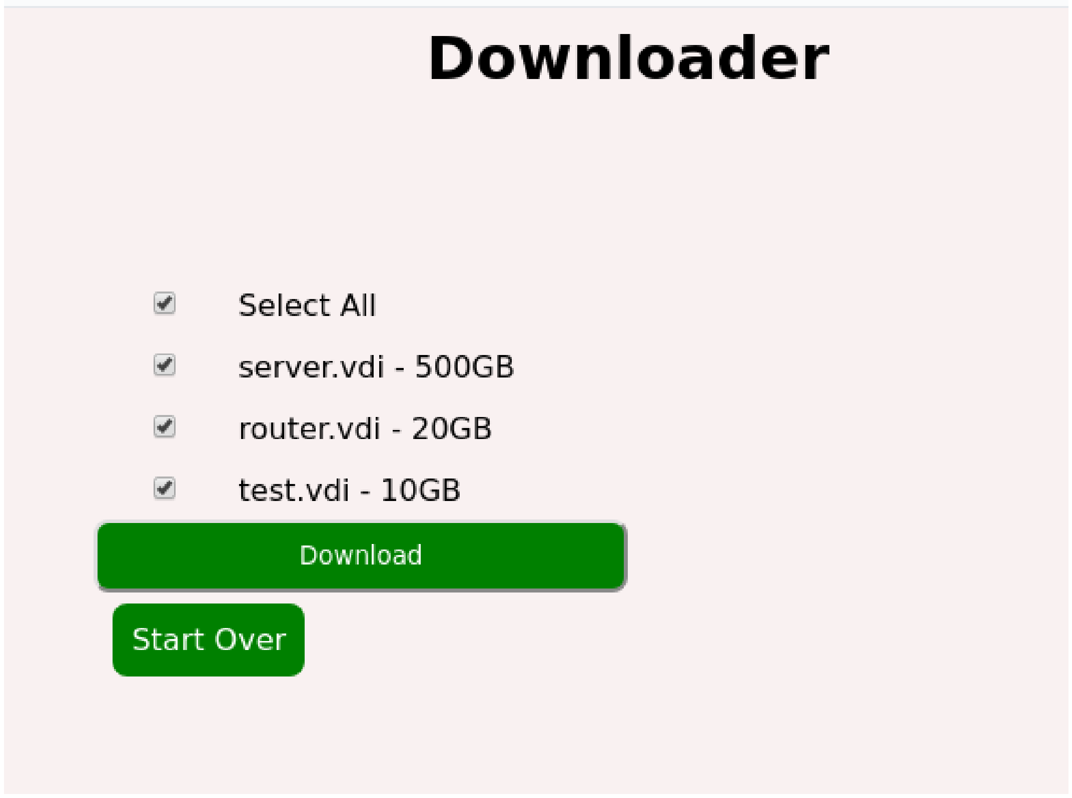


Figure 6: All files selected for downloading.

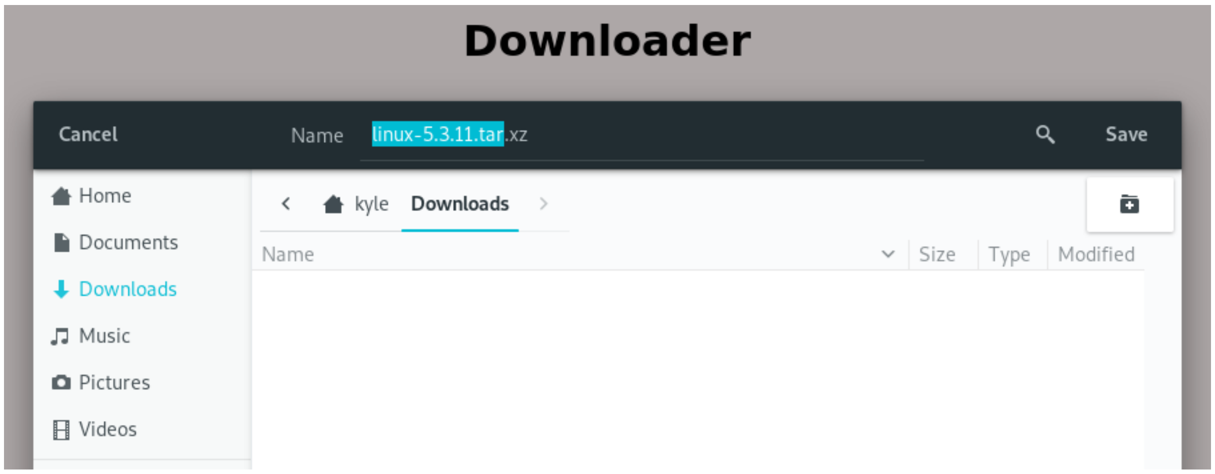


Figure 7: Download button pressed, browser prompts where to save files.

Requirements Specification

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

List of Figures

Revision History

Revision	Date	Author(s)	Description
3.0	05.03.2020	KT	Updated requirements to reflect the application and customer changes.
2.2	12.07.2019	KT	Added in Dr. Corrigan's corrections
2.1	12.04.2019	TS	Provided more grammatical checking along with a Reference Recommendation
2.0	12.04.2019	KT	Added priority section & final edits
1.2	12.03.2019	KT	Grammar edits and wording
1.1	11.16.2019	NF	Grammar edits and wording
1.0	11.15.2019	RB NF TS KT	First Final Doc
0.9	11.14.2019	KT	Spelling & grammar edits
0.8	11.12.2019	TS	Added more components to subsections for length
0.7	10.29.2019	NF	Edits to whole doc
0.6	10.29.2019	KT	Edits
0.5	10.28.2019	NF	Background
0.4	10.28.2019	RB	Introduction
0.3	10.15.2019	TS	Completed Overview
0.2	10.10.2019	RB NF TS KT	Filled in Environment & Operation sections
0.1	09.27.2019	KT	Document Creation

1 Introduction

This requirements specification document will go over the requirements for the Virtual Machine file Downloader for Cypherpath's Resiliency Platform [2] that was outlined in the `Project Plan` [1]. This *Downloader* application will allow users to transfer files directly from various cloud platforms into the Resiliency Platform created by Cypherpath that gives their customers cyber resiliency. The document will cover how the application will provide a solution for users to easily upload their Virtual Machine files to the Resiliency Platform. Along with covering the solution, the requirements specification will also provide details on how users will interact with the application through a web interface. This document will be understood and agreed upon by both the customer (Cypherpath) and the software development team (Captain CyBeard).

Subsequent sections will include background information on Cypherpath and their Resiliency Platform, followed by a high level overview of the application. After this, the document will get into the details of the application such as the environment the application will execute in, and how the application will be operated by users and web administrators.

The Cypherpath Resiliency Platform tool is a product that provides their customers the ability to upload pre-existing virtual machines, network configurations and more, into a self-contained digital environment. This enables their customers to quickly recover from events such as ransomware attacks or human errors. Right now, customers have to manually download their virtual disk images from the various cloud platforms they are hosted on, and then upload those to the Resiliency Platform. Cypherpath would like to have a web application that will enable users to have their virtual disk images go directly from the cloud platform into the Resiliency Platform, cutting out the multi-step process of downloading and then uploading large virtual disk images.

2 Background

Cypherpath's Resiliency Platform started out as a platform where the U.S. Military could emulate the networks of countries and simulate cyber attacks between the countries in order to learn about the strength and weaknesses of the countries simulated. After proving its usefulness in this market, Cypherpath entered the commercial market and provides its platform as a service to companies and local governments alike to provide resiliency to these entity's network infrastructures. Resiliency in the cyber sense means that systems and networks are secure, recoverable and continuous. Customers of Cypherpath enjoy a readiness of their technological resources, and can rapidly adjust and adapt those resources when failures, damages or attacks occur in order to keep their business running smoothly.

This project will become a component of the Resiliency Platform by enabling Cypherpath's customers to more easily migrate their virtual assets into the platform. Currently customers have to download their virtual assets onto their local machine and then upload them into the Resiliency Platform. This project will simplify this process by having the Resiliency Platform download the virtual asset directly into its infrastructure. The software development team will not be integrating the application with the Resiliency Platform. The product delivered to Cypherpath will be a standalone web application that downloads digital assets from various cloud platforms. Upon completion of the project, Cypherpath will integrate the web application into their Resiliency Platform by changing the user interface and theme of the application to match theirs. They only need the backend Python-Django part of the application.

3 Overview

The purpose of this application is to allow users to download virtual disk images from online cloud platforms, to the Resiliency Platform (the application delivered will only download the files to local storage). Currently, Cypherpath customers manually visit each of their cloud platforms and download their files, then upload them into the Resiliency Platform. This application will act as middleware and direct the download into the Resiliency Platform. The application itself will operate as a web server using Python3.8+ [4] and Django2.2+ [5]. Users will start the application by navigating to the URL pointing to the application. Once at the main page the user will select a cloud platform from a pull down menu and then be directed to authenticate to the selected platform. After authentication, the application will use the API of the cloud platform and show the user their files stored on the platform. The user will be able to navigate their directory structure and select which files and folders to load into the platform (again, during development the application will download the files to local storage instead of interacting with the Resiliency Platform).

The application will be written with Python3.8+ and Django2.2+, including the requests libraries for API calls, Authentication libraries like SAML and standard libraries. To install the application, Cypherpath will integrate the Django web application into their environment along with installing (if not installed already) the libraries for interacting with the external cloud platforms and authentication API's. Users of the application will not need to install any software since most operating systems come with a web browser which will interact with the application on the user's end.

4 Environment

This section will describe the environment the application will reside in. This will include the web application, authentication API's, cloud storage API's and the local storage of the machine running the application. During development this will be the users machine, but Cypherpath will be using their storage scheme for production.

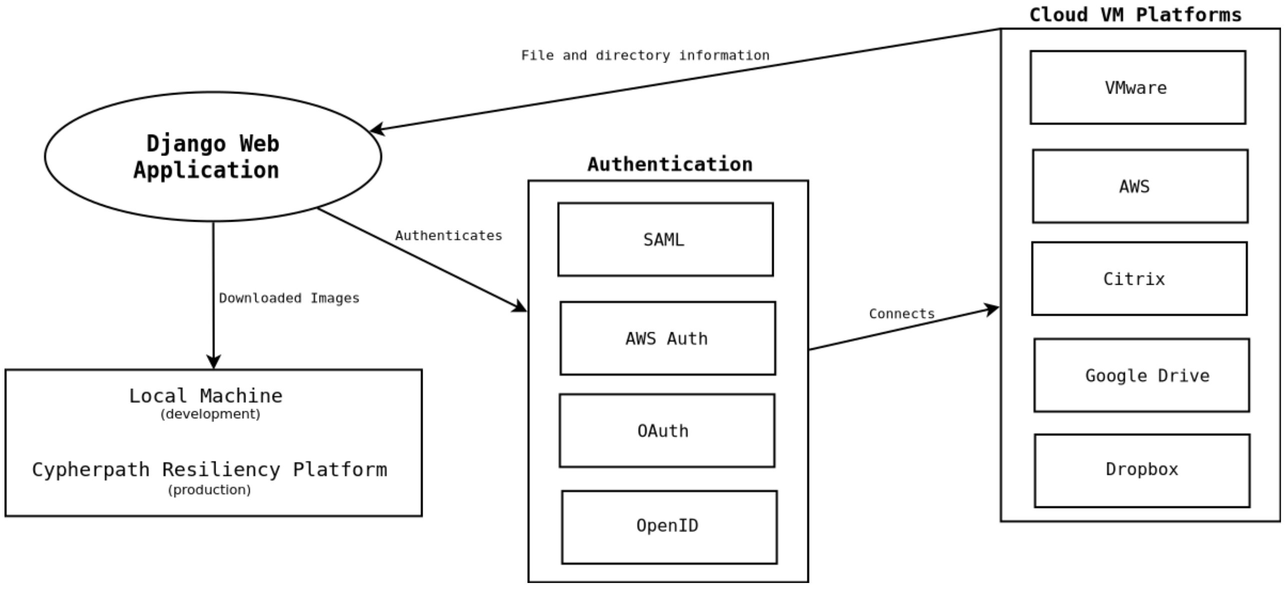


Figure 1: A visual representation of the applications environment.

4.1 Web Application

The web server for the application during development will be Django's built in development server and when the project is completed, Cypherpath will integrate the web application into their existing ecosystem using their webserver. The Django application will host the functionality of the application, including HTML templates for the web pages, where users provide Virtual Machine URLs and browse their directories. Additionally, the Django framework will allow the usage of Python classes that will enable the back-end operation of the application.

4.2 Authentication API's

The application will authenticate to the cloud platforms with various authentication methods and API's. The application will support the authentication for various platforms. Some will use OAuth2 or variations of it and redirect to the cloud provider's authentication, others will use a custom login page to obtain the user's credentials (like AWS). The application will be designed in a modular fashion so that more authentication methods can be easily added later.

4.3 Cloud Storage API's

Once the application is authenticated to a cloud storage platform it will gather the list of files on the platform for showing to the user and allowing them to select which of their files they would like to download. The application will support interacting with Amazon Web Services [3], Google Drive and Dropbox. The API's that are necessary for the communication, will request the directory structure of their cloud storage and provide it to the user so they can select which files from the cloud platform, they would like to download. Like the authentication modules, the modules for interacting with the cloud API's will need to be designed so that adding support for other cloud platforms will be possible.

4.4 Local Storage

Once the user has selected which files to download from their cloud platform, the application will download those files to the user's local storage. The application will download all of the user's selected files from the cloud platform to their local storage. The local storage location will be determined by the web server, the application will download the files from the cloud platform to a location specified by the application backend. Nothing else needs to be done with the files because after the application is finished and Cypherpath has integrated the web application into their platform, they will direct the downloads where they need them.

5 Operation

In the following sections the operation of the application will be described, including starting the application (invocation) from the user's perspective and the web server administrator, the commands the application uses and finally, how users and administrators will terminate the application.

5.1 Invocation

During development the application will be started using Django's built-in webserver and the user will not need to be authenticated to use the application. To invoke the application the user will simply enter the URL for the application (such as http://localhost:8080). They will then be presented with the main screen of the application. The screen will have a single pull down menu for a user to select a cloud platform. Additionally a submit button will be provided for them to continue to step through the application.

To start the development server, the developer/mock web administrator will use the following commands in the Django project directory containing the *manage.py* file.

```
$ python manage.py migrate  # only necessary to run the first time
$ python manage.py runserver
```

The user will now be able to use their web browser to visit the main page.

The user interface and invocation of the application are simple. Later when Cypherpath integrates the application into their Resiliency Platform, invocation by the user will be from within Cypherpath's platform. Cypherpath does not have preferences on the user interface since they will be adding their own style, layout, themes, etc. to the user interface. The server side of the application will be invoked from Cypherpath's web server that serves their platform.

5.2 User Actions

This section covers how the user will interact with the application and how entering the application, displaying the file structure, downloading virtual disk images and error catching will be implemented.

5.2.1 Select Platform

On the main page of the application the user will be presented with a pull down menu where they will select a cloud platform. The menu will be centrally located with a design to help users clearly see where to select the platform. Once a platform is selected and submitted the application will invoke the authentication methods for that cloud platform. This may be redirecting the user the cloud platform's authentication or prompting the user for their credentials and passing those directly to the cloud platform. Once authenticated the application will present the user with their files and folders that are on that platform.

5.2.2 Display File Structure and Download Selected Files

Once the user has selected a cloud platform and is authenticated, they will be presented with their files that are stored on that platform. They can use this information to verify all files were retrieved from the cloud once downloaded and select which files to download. After selecting the files the user wants, they will click the download button, which initiates the download of their selected files from the cloud platform. This will redirect the user to a page showing them just the files that they selected. During development, the application server will download the files to a location on server's local drive. When Cypherpath assimilates the application into their platform they will direct the download where they need it. This will be accomplished by Cypherpath changing the location that the application server stores the downloads.

These pages that present the user with their files and downloads will also contain a "start over" button, which will bring them back to the main page.

5.2.3 Error Catching

Errors will be detected through the various aspects of the program. If there are not any cloud platforms install on the application, the pull down menu will gracefully display so. If the user does not provide valid credentials to the external cloud platform they will be prompted to re-enter their credentials. The application will not enforce a limit on the number of authentication attempts the user can make, however the external authentication API or the cloud platform may have limits that it will enforce. Additionally if the download fails and the web browser's download manager is unable to recover or restart the download then the user will have to press the download button again assuming they haven't left the page.

5.3 Termination

Since the user is not authenticated to the web application, to terminate their interaction with the application they will simply close the browser tab or window that has the application open. Use of the authentication token will be deleted as well per use, there will be no cookies to remember users every time they log into the accounts. After termination of the application the user will have to go through the same steps when they invoke the application since sessions and states are not remembered by the application.

During development the Django Web Server that is used will be terminated by the developer entering *ctrl + c* on their keyboard to end the process. Once Cypherpath has integrated the application into their ecosystem they will terminate the web application their way.

6 Priority of Cloud Platforms

Cypherpath has expressed that Amazon Web Services is their priority to have supported with the application. After those modules are working, Captain Cybeard will then add support for Google Drive, Dropbox, and Citrix in that order of priority. The requirement is that at least Amazon Web Services be supported.

At first, VMware's platforms were the priority, however with the acquisition of Cypherpath they no longer prioritized VMware since setting up a VSphere cloud instance to support was no longer feasible.

References

[1] Ryan Breitenfeldt, Noah Farris, Trevor Surface, Kyle Thomas. *Project Plan.* [*Project Plan for Downloader Application*] 2019.

[2] Cypherpath.com. (2019). Cypherpath, Inc. [online] Available at: https://www.cypherpath.com/ [Accessed 21 Oct. 2019].

[3] Amazon Web Services, Inc. (2019). Amazon Web Services (AWS) - Cloud Computing Services. [online] Available at: https://aws.amazon.com/ [Accessed 10 Oct. 2019].

[4] Python.org. (2019). Welcome to Python.org. [online] Available at: https://www.python.org/ [Accessed 6 Nov. 2019].

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

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

List of Figures

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.

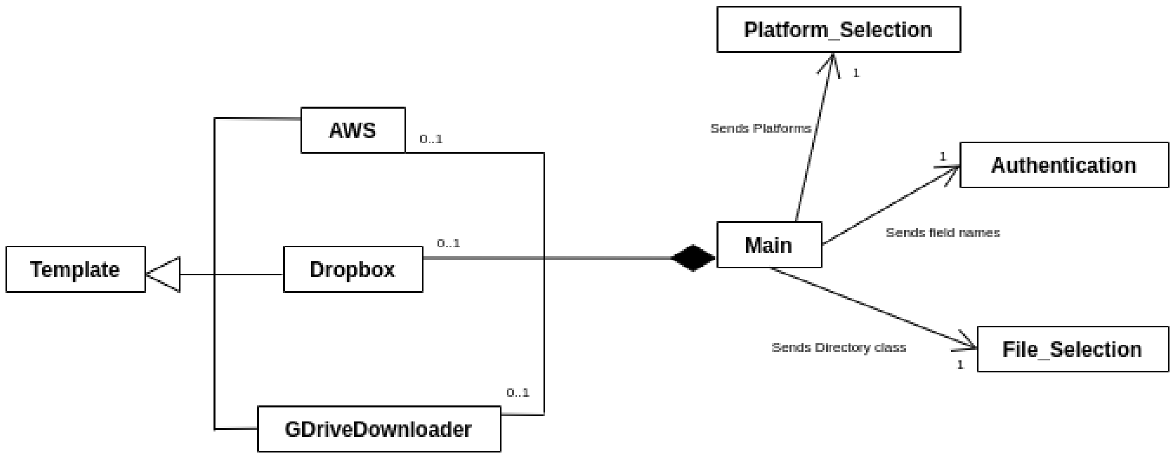


Figure 1: The overall application architecture.

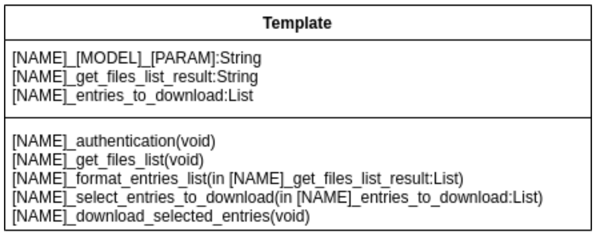

Template
[NAME]_[MODEL]_[PARAM]:String
[NAME]_get_files_list_result:String
[NAME]_entries_to_download:List
[NAME]_authentication(void)
[NAME]_get_files_list(void)
[NAME]_format_entries_list(in [NAME]_get_files_list_result:List)
[NAME]_select_entries_to_download(in [NAME]_entries_to_download:List)
[NAME]_download_selected_entries(void)

Figure 2: The template class all platforms follow

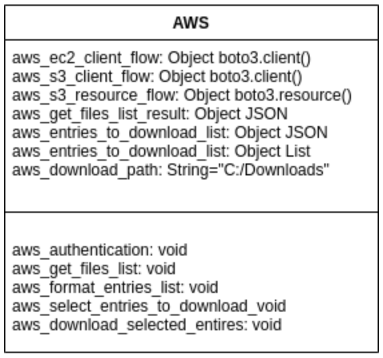

AWS
aws_ec2_client_flow: Object boto3.client()
aws_s3_client_flow: Object boto3.client()
aws_s3_resource_flow: Object boto3.resource()
aws_get_files_list_result: Object JSON
aws_entries_to_download_list: Object JSON
aws_entries_to_download_list: Object List
aws_download_path: String="C:/Downloads"
aws_authentication: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

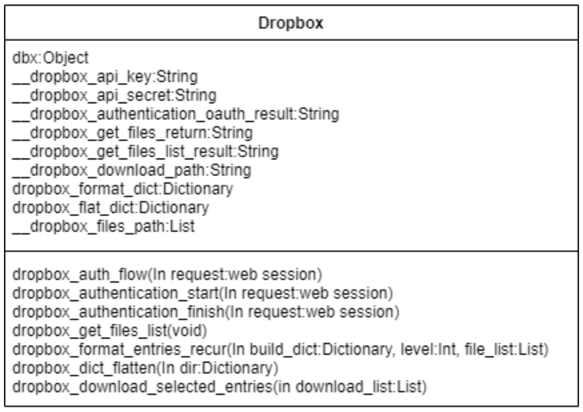

Dropbox
dbx:Object
__dropbox_api_key:String
__dropbox_api_secret:String
__dropbox_authentication_oauth_result:String
__dropbox_get_files_return:String
__dropbox_get_files_list_result:String
__dropbox_download_path:String
dropbox_format_dict:Dictionary
dropbox_flat_dict:Dictionary
__dropbox_files_path:List
dropbox_auth_flow(In request:web session)
dropbox_authentication_start(In request:web session)
dropbox_authentication_finish(In request:web session)
dropbox_get_files_list(void)
dropbox_format_entries_recur(In build_dict:Dictionary, level:Int, file_list:List)
dropbox_dict_flatten(In dir:Dictionary)
dropbox_download_selected_entries(in download_list:List)

Figure 4: The Dropbox Class.

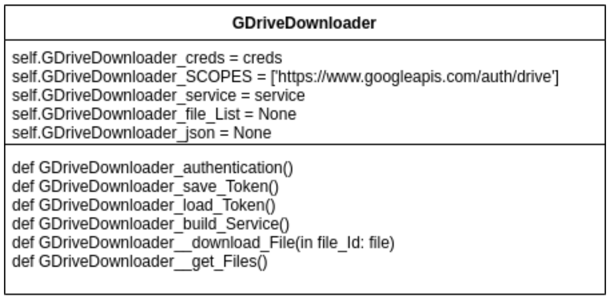

GDriveDownloader
self.GDriveDownloader_creds = creds
self.GDriveDownloader_SCOPES = ['https://www.googleapis.com/auth/drive']
self.GDriveDownloader_service = service
self.GDriveDownloader_file_List = None
self.GDriveDownloader_json = None
def GDriveDownloader_authentication()
def GDriveDownloader_save_Token()
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.

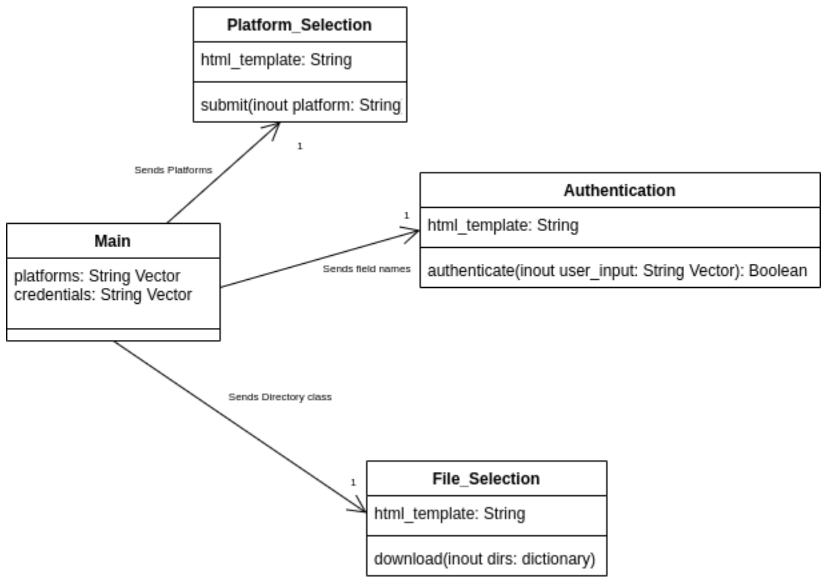


Figure 6: The Controller and Views Classes.

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 thins 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 is 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 formated 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 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 veiws.py script.

Additionally the urls.py/urlpattern 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 plaform, 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 veiws.py script. Additionally the urls.py/urlpattern 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 formated 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 transfer 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 uploaded to the host server to a specified location provided by the owner of the server.

For future developers, in order to properly use this class, several changes will need to occur. An app must be created through the Dropbox.com/developer app-console. When creating the application the developer will be given an API key and API secret to be used, there are other configurations that will need to be made that are explained further in this document.

3.3.1 Associations

Is contained in UI

Makes calls to Dropbox

3.3.2 Attributes

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_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.

To generate the API key, the developers must create an associated app within the dropbox.com/developer website, The generated API key must be stored within an API_keys.py file within the platforms directory. The string itself must also be stored under the variable name Dropbox_Api_key, or the current attribute assignment will fail.

__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.

To generate the API secret, the developers must create an associated app within the Dropbox.com/develops website, The developers, must reveal the secret show within the application console of the developer website for Dropbox. When used the string must be saved to the API_keys.py file, within the platforms directory. The variable storage name must be Dropbox_Api_secret, otherwise the current functionality of the application will fail. Reference API_keys.py for examples.

__dropbox_authentication_oauth_result:Object
This attribute will store the object generated by dropbox.oauth.DropboxOAuth2Flow.finish() call. This attribute is used after 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. This object contains an .access_token attribute that will be passed into the dbx attribute to instantiate the Dropbox user Object.

__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_download_path:String="/Downloads"
This attribute will hold the path value used for downloading files. For this application, the download functionality will not download to the users main OS. The files will be downloaded directly to the server that is hosting the website, as per the requirement specification.

dropbox_fromat_dict:Object Dict Value: {'path':", 'dirs':[], 'files':[]}
This attribute is a specified dict that is used for the Django Views object within the Django platform. This attribute can be interchanged with the operation of the view object, if the developer wanted to display the directories and subsequent included files. This is the default created dict when the files are retrieved from the Dropbox user account.

dropbox_flat_dict:Object Dict Value: {'files':[]}
This attribute is a specified dict that is used for the Django Views object within the Django platform. This attribute is the currently set view default, it shows only the files available to the user, excluding the directories included within.

__dropbox_files_paths:Object List
This attribute stores the paths of the files returned from the dropbox.get_file_list_folder(). This attribute is used for the paths when passing the necessary argument to the dropbox.files_download_to_file function.

3.3.3 Methods

dropbox_auth_flow: In:request
This method implements the authentication pattern used by the Dropbox Python SDK. This function uses the dropbox.oauth.DropboxOAuth2Flow() function to return an OAuth object. To use the command, a redirect_uri needs to be specified. This requires the developer to specify the location in the Django application where the dropbox-auth-finish/ path is located. Additionally within the dropbox.com/developer app console the developer needs to add the redirect URI to the console in order for the application to work properly. The final required component is a 'dropbox-auth-csrf-token' to be called within the object. This is used to mitigate attempts at cross site scripting.

dropbox_authentication_start: In:web_app_session
This method redirects the user to the Dropbox authentication website. The authorization_url is generated by the previous method returning the OAuth2 object, and using the command .start() and passing in the web_app_session as well. After the authorize URL is generated, the method will then redirect the user to the required page.

dropbox_authentication_finish: In:request
This method finalizes the authentication necessary to gain access to the user's Dropbox. Calling the dropbox_auth_flow method passing in request, and calling the .finsh() method, passing in request.GET to the method will generate the oauth result which contains the access_token needed to instantiate the Dropbox user object. The method will be wrapped in a try block in case http errors occur during login. Include bad login state CSRF errors from a cross site scripting attempt, Invalid credentials and a catch all for remaining errors. After the authentication has returned successfully, the dropbox object is then created. After instantiating the Dropbox user object, the files will be collected from their account, finally both the format method and the flatten method will be called to pass the required dict to the Views page. Finally the user will be redirected to the '/files' page where they will be able to select the files they wish to download.

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 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_recur: In: build_dict, level, file_list
This method uses a recursive algorithm to generate a multi leveled dict that contains the users directories, and included files, and full paths needed to display the information. With the current implementation the detected difference between a file and a folder is the application users propensity to add a '.' within a filename, and not within their directories. However if this were to occur there would be an invalid file type during download. This was not specified to change via the requirement specification. Via the recursive algorithm, a file_list will be generated that contains all the currently discovered files, the level specifies how deep within the folder tree the current folder and file sets are located. When a folder is discovered, the method is called again passing in the new directory and finding all files associated with the folder.

dropbox_dict_flatten_recur: In:dir
This method uses a recursive algorithm to flatten the previous method dict. This is used for the current implementation of the project. If a developer wishes to use the directory structure to help specify files, this function is unneeded. The recursive algorithm appends to a list the files received for the various folders, when a folder is detected, the method is called again and the files are appended to the list.

dropbox_download_selected_entries: In:download_list
This method will use the dbx.files_download_to_file() and dbx.files_download_zip_to_file(). It will use the locally saved attribute dropbox_entries_to_download_list to iterate through, calling the API using the developer defined dropbox_download_path. The function will then download all the files within the list, to the location specified on the website host server. 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 overGDriveDownloader_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 overGDriveDownloader_files_to_download.

GDriveDownloaded_authentication_flow
This method creates the autentication flow using the seceret 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 redircts 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.

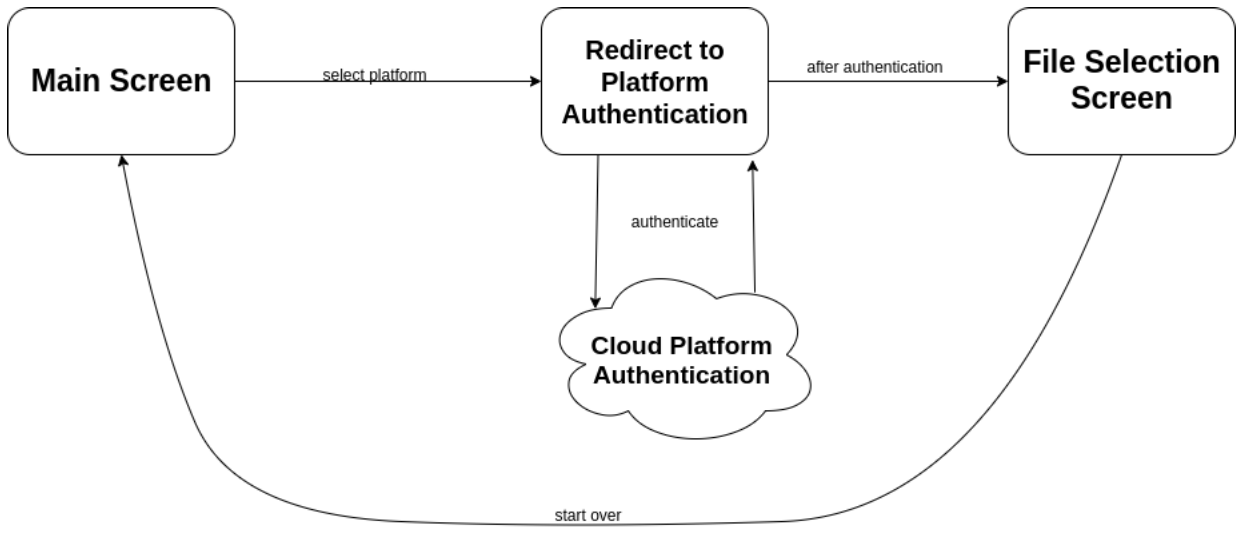


Figure 7: The workflow for the user

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.

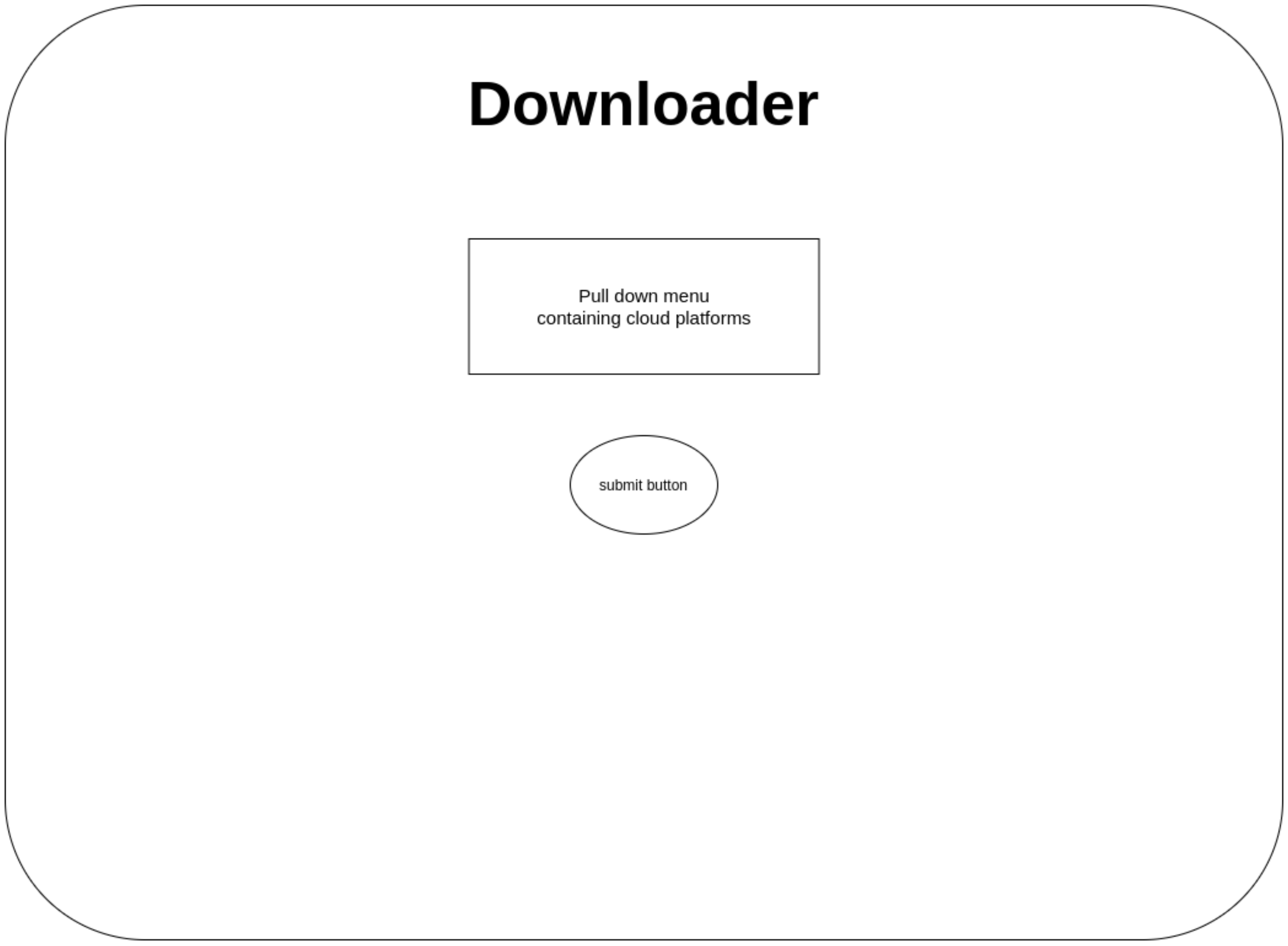


Figure 8: The first screen of the application.

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.

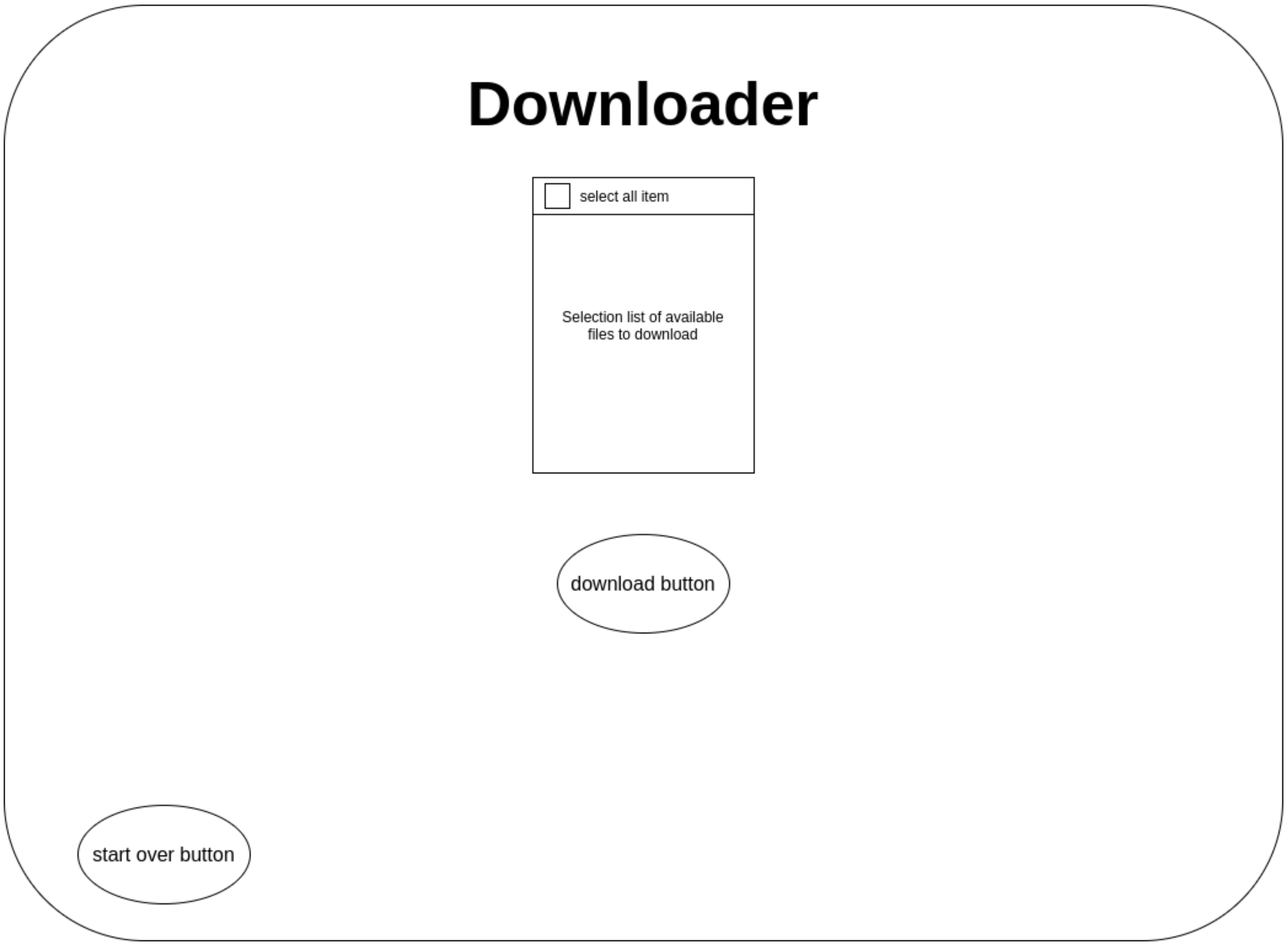


Figure 9: The screen to view and download files.

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

Implementation

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

project_root/requirements.txt

```
asn1crypto==0.24.0
attrs==17.4.0
Automat==0.6.0
blinker==1.4
boto3==1.13.0
botocore==1.16.0
cachetools==4.0.0
certifi==2019.11.28
cffi==1.14.0
chardet==3.0.4
click==7.1.1
colorama==0.3.7
configobj==5.0.6
constantly==15.1.0
cryptography==2.4
Django==2.2.11
docutils==0.15.2
dropbox==9.4.0
Flask==1.1.2
google-api-python-client==1.7.11
google-auth==1.11.2
google-auth-httplib2==0.0.3
google-auth-oauthlib==0.4.1
httplib2==0.17.0
hyperlink==17.3.1
idna==2.9
incremental==16.10.1
itsdangerous==1.1.0
Jinja2==2.11.2
jmespath==0.9.5
jsonpatch==1.16
jsonpointer==1.10
jsonschema==2.6.0
keyring==10.6.0
keyrings.alt==3.0
MarkupSafe==1.1.1
netifaces==0.10.4
oauth2client==4.1.3
oauthlib==3.1.0
pyasn1==0.4.8
pyasn1-modules==0.2.8
pycparser==2.20
pycrypto==2.6.1
PyJWT==1.5.3
pyOpenSSL==17.5.0
pyserial==3.4
python-dateutil==2.8.1
python-debian==0.1.32
pytz==2019.3
```

```
pyxdg==0.26
requests==2.23.0
requests-oauthlib==1.3.0
requests-unixsocket==0.1.5
rsa==4.0
s3transfer==0.3.3
SecretStorage==2.3.1
service-identity==16.0.0
six==1.14.0
sqlparse==0.3.1
ssh-import-id==5.7
tqdm==4.45.0
Twisted==17.9.0
uritemplate==3.0.1
urllib3==1.25.8
Werkzeug==1.0.1
zope.interface==5.1.0
```

project_root/README.md

```
# Cloud Backup Application
Language:       Python 3.8 Django 2.2

Dev Env:         Linux x64

Authors:          Ryan Breitenfeldt,
                        Noah Farris,
                        Trevor Surface,
                        Kyle Thomas

Class:              CptS 421/423 Fall 2019/Spring 2020

University:    Washington State University Tri-CIties

### Development

#### Making a Python Virtual Environment:

`$ python -m venv <name of environment>` (I used cloud-backup-env)

Then add the folder it created to your .gitignore file

`$ source <name of env folder>/bin/activate`

You are now in the virtual environment anything you install will not be system wide

The first time you start:

`$ pip install -r requirements.txt`

To leave the environment:

`$ deactivate`

To add dependencies that you installed please do:

`$ pip freeze > requirements.txt`

Then commit the new requirements.txt to the repo.

#### Running the webserver:
Viewing on the same pc

`$ python manage.py migrate`

`$ python manage.py runserver`

Then visit: https://localhost:8000/cloud
```

Viewable from any pc on the network:

‘$ python manage.py runserver 0:8000‘

#### Adding a Cloud Platform
Add the file with the new cloud platform class into ‘cloud_download/platforms‘

Then import the file in ‘cloud_download/platforms/__init__.py‘

Finally, add the platform inside ‘cloud_download/views.py‘

project_root/cloud_backup/urls.py

```
"""cloud_backup URL Configuration

The ‘urlpatterns‘ list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/2.2/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path('', include('cloud_download.urls')),
    path('cloud/', include('cloud_download.urls')),
    path('admin/', admin.site.urls),
]
```

project_root/cloud_download/urls.py

```
#!/usr/bin/env python3

""" URL directs for cloud backup application
Application:    Cloud Backup
File:           /cloud_backup/cloud_download/urls.py
Description:    url paths
Language:       Python 3.8 Django 2.2
Dev Env:        Linux x64

Authors:        Ryan Breitenfeldt
                Noah Farris
                Trevor Surface
                Kyle Thomas
Class:          CptS 421/423 Fall '19 Spring '20
University:     Washington State University Tri-CIties
"""

from django.urls import path
from . import views

__authors__ = ['Ryan Breitenfeldt', 'Noah Farris', 'Trevor Surface', 'Kyle Thomas']


urlpatterns = [
    '''
    For Admins:
        The following paths need to be updated when using OAuth2.0 authentication. Depending upon th
        created two new 'paths' need to be created. A '[Platform_name]-auth-start', and a '[Platform
        same fashion below. otherwise the OAuth2.0 authentication will not work properly.
    '''
    path('', views.Index.as_view(), name='index'),
    path('dropbox-auth-start/', views.dropbox.dropbox_authentication_start),
    path('dropbox-auth-finish/', views.dropbox.dropbox_authentication_finish),
    path('google-auth-start/', views.google.GDriveDownloaded_authentication_start),
    path('google-auth-finish/', views.google.GDriveDownloaded_authentication_finish),
    path('files/', views.Files.as_view(), name='files'),
    path('aws_login/', views.Aws_Login.as_view(), name='aws_login'),
]
```

project_root/cloud_download/views.py

```
#!/usr/bin/env python3

""" URL views for cloud backup application
Application:     Cloud Backup
File:                 /cloud_backup/cloud_download/views.py
Description:    Django Views / web pages
Language:       Python 3.8 Django 2.2
Dev Env:         Linux x64

Authors:          Ryan Breitenfeldt
                        Noah Farris
                        Trevor Surface
                        Kyle Thomas
Class:              CptS 421/423 Fall '19 Spring '20
University:    Washington State University Tri-Cities
"""
import ast
from django.shortcuts import render, redirect
from django.views import View
from . import platforms
from .forms import AWS_AuthForm
from django import forms
import json
from .models import aws_data
import ast
from django.contrib import messages

__authors__ = ['Ryan Breitenfeldt', 'Noah Farris', 'Trevor Surface', 'Kyle Thomas']

'''
For Admins:
    When adding a new platform to the software if the platform SDK supports OAuth2.0 authentication
    Then the new platform class will need to be instantiated bellow along with the previous OAuth2.0
    class objects.
'''
global cloud
cloud = ""
dropbox = platforms.dropbox_script.DropBox()
google = platforms.gDriveDownloader.GDriveDownloader()


class Index(View):
    '''
    For Admins:
        When adding a new platform, the attribute platform will need to be updated with the new plat
        Additionally the post method of this class will need to be updated with elif similar to the
        of the post method. To finalize the post method, there also needs to be a redirect to '[plat
        To enable this as well the urls.py needs to be updated.
    '''
    index_template = 'cloud_download/index.html'
    platforms = ['google', 'dropbox', 'aws']

    def get(self, request):
        context = {'platforms': self.platforms}
        return render(request, self.index_template, context)

    def post(self, request):
        platform = request.POST['platform']
        global cloud
        if platform == 'google':
            print(platform)
            cloud = 'google'
            return redirect('google-auth-start/')
        elif platform == 'dropbox':
            cloud = 'dropbox'
            return redirect('dropbox-auth-start/')
        elif platform == 'aws':
            cloud = 'aws'
            return redirect('aws_login/')
        else:
            print("Unsupported platform")
            return redirect('index/')

        return redirect('aws_login/')

class Files(View):
    '''
    For Admins:
        The current functionality of the software requires a single level dict with the parameters {
        of dicts containing {'path':'', 'files':''} where 'files' is the name of the file without fo
        are implemented with a dict that allows mulilevel folder and file view, if the developer wou
        programming is required.

        The get method of this class will need to be updated with another elif to provide the necces
    '''

    template = 'cloud_download/files.html'
    success_template = 'cloud_download/success.html'
    context = {}
    def get(self, request):
        if cloud == 'dropbox':
            context = dropbox.dropbox_flat_dict
        elif cloud == 'google':
            context = google.GDriveDownloader_json
        elif cloud == 'aws':
            obj = aws_data.objects.first()
            key_id_object = aws_data._meta.get_field("aws_key_id")
            key_object = aws_data._meta.get_field("aws_key")
            aws_key_id = key_id_object.value_from_object(obj)
            aws_key= key_object.value_from_object(obj)
            aws = platforms.aws.aws(aws_key_id, aws_key)
            return render(request, self.template, {'files': aws.list_images_in_bucket()})

        return render(request, self.template, context)
    def post(self, request):
        '''
        For admins
            This method needs to be updated whe a new platform is added regardless of Oauth2 authent
            This is what calls the download procedures defined within the new platform class. Any ad
            information needed for downloading will need to be added to the context list and the inf
            transfered. Add an elif case for the new platform with a similar for loop consitant with
            if statements, then call the download funciton neccessary for the platform.
        '''

        context = {}
        user_selection = request.POST.getlist('box')
        files_to_download = []

        if cloud == 'aws':
            obj = aws_data.objects.first()
            key_id_object = aws_data._meta.get_field("aws_key_id")
            key_object = aws_data._meta.get_field("aws_key")
            aws_key_id = key_id_object.value_from_object(obj)
            aws_key= key_object.value_from_object(obj)
            aws = platforms.aws.aws(aws_key_id, aws_key)
            for file in user_selection:
                if file == 'on':
                    print('False')
                else:
                    file_name = ast.literal_eval(file)
                    aws.download_image('testbucket1293248523850923853', file_name['name'])
                    json_acceptable_string = file.replace("'", "\"")
                    files_to_download.append(json.loads(json_acceptable_string))
            context['files'] = files_to_download
            return render(request, self.success_template, context)

        elif cloud == 'dropbox':
            for file in user_selection:
                if file == 'on':
                    print('False')
                else:
                    file_name = ast.literal_eval(file)
                    files_to_download.append(file_name)
            context['files'] = files_to_download
            dropbox.dropbox_download_selected_entries(context['files'])
            return render(request, self.success_template, context)

        elif cloud == 'google':
            for file in user_selection:
                if file == 'on':
                    print('False')
                else:
                    files_to_download.append(ast.literal_eval(file))
            context['files'] = files_to_download
            if cloud == 'google':
                google.GDriveDownloader_files_to_download = files_to_download
                google.GDriveDownloader__download_File()
            return render(request, self.success_template, context)

'''
For Admins:
    The following class was implemented per the standard of AWS. If the new platform does not suppor
    then additional views will need to be created to allow the login parameters required by the plat
'''

class Aws_Login(View):
    template_name = 'cloud_download/aws_login.html'

    def get(self, request):
        form = AWS_AuthForm(request.POST)
        return render(request, self.template_name, {'form': form})

    def post(self, request):
        aws_data.objects.all().delete()
        form = AWS_AuthForm(request.POST)
        if form.is_valid():
            form.save()
            aws_key_id = form.cleaned_data.get('aws_key_id')
            aws_key = form.cleaned_data.get('aws_key')
            try:
                platforms.aws.aws(aws_key_id, aws_key).get_image_list()
                return redirect("/cloud/files/")
            except:
                messages.error(request, 'Amazon Web Services Key or Key ID is incorrect!')

        else:
            form = AWS_AuthForm()

        return render(request, self.template_name, {'form': form})

def index_redirect(request):
    return redirect('index/')
```

project_root/cloud_download/models.py

```
from django.db import models

class aws_data(models.Model):
    aws_key_id = models.CharField(max_length=500)
    aws_key = models.CharField(max_length=500)
```

project_root/cloud_download/forms.py

```
from django import forms
from .models import aws_data
from . import platforms

class AWS_AuthForm(forms.ModelForm):
    aws_key_id = forms.CharField(widget=forms.TextInput(attrs={'placeholder': 'AWS Key ID'}), label=
    aws_key = forms.CharField(widget=forms.TextInput(attrs={'placeholder': 'AWS Key'}), label='Key',
    class Meta:
        model = aws_data
        managed = False
        fields = ('aws_key_id', 'aws_key',)

    def clean_message(self):
            aws_key_id = self.cleaned_data.get('aws_key_id')
            aws_key = self.cleaned_data.get('aws_key')
            try:
                platforms.aws.aws(aws_key_id, aws_key).get_image_list()
            except:
```

project_root/cloud_download/apps.py

```
#!/usr/bin/env python3

""" App declaration for cloud backup application
Application:     Cloud Backup
File:                 /cloud_backup/cloud_download/apps.py
Description:    Django App Config
Language:       Python 3.8 Django 2.2
Dev Env:         Linux x64

Authors:          Ryan Breitenfeldt
                        Noah Farris
                        Trevor Surface
                        Kyle Thomas
Class:              CptS 421/423 Fall '19 Spring '20
University:    Washington State University Tri-CIties
"""

from django.apps import AppConfig

__authors__ = ['Ryan Breitenfeldt', 'Noah Farris', 'Trevor Surface', 'Kyle Thomas']


class CloudDownloadConfig(AppConfig):
    name = 'cloud_download'
```

project_root/cloud_download/static/cloud_download/aws.css

```
@import url(https://fonts.googleapis.com/css?family=Open+Sans);

body{
    font-family: 'Open Sans', sans-serif;
    background:#363636;
    margin: 0 auto 0 auto;
    width:100%;
    text-align:center;
    margin: 20px 0px 20px 0px;
  }

  p{
    font-size:12px;
    text-decoration: none;
    color:#ffffff;
  }

  h1{
    font-size:1.5em;
    color:#ddd;
    font: 'Open Sans', Arial, sans-serif;
  }

  h2{
    font-size:1.0em;
    color:#ddd;
    font: 'Open Sans', Arial, sans-serif;

  }
  .box{
    background:#484848;
    width:300px;
    border-radius:6px;
    margin: 0 auto 0 auto;
    padding:0px 0px 70px 0px;
    border: #2980b9 4px solid;
  }

  .aws_key_id{
    background:#ecf0f1;
    border: #ccc 1px solid;
    border-bottom: #ccc 2px solid;
    padding: 8px;
    width:250px;
    color:#AAAAAA;
    margin-top:10px;
    font-size:1em;
    border-radius:4px;
  }
```

```
.aws_key{
  border-radius:4px;
  background:#ecf0f1;
  border: #ccc 1px solid;
  padding: 8px;
  width:250px;
  font-size:1em;
}

.btn{
  background:#484848;
  width:125px;
  padding-top:5px;
  padding-bottom:5px;
  color:#ddd;
  border-radius:3px;
  border: #008CBA 2px solid;
  margin-top:20px;
  margin-bottom:20px;
  float:left;
  margin-left:80px;
  font-weight:800;
  font-size:0.8em;
}

.btn:hover{
  background: rgb(25, 63, 70);
}
```

project_root/cloud_download/static/cloud_download/aws_login_style.css

```
body{
    font-family: 'Open Sans', sans-serif;
    background:#363636;
    margin: 0 auto 0 auto;
    width:100%;
    text-align:center;
    margin: 20px 0px 20px 0px;
  }

  p{
    font-size:12px;
    text-decoration: none;
    color:#ffffff;
  }

  h1{
    font-size:1.5em;
    color:#ddd;
  }

  h2{
    font-size:1.0em;
    color:#ddd;
  }
  .box{
    background:white;
    width:300px;
    border-radius:6px;
    margin: 0 auto 0 auto;
    padding:0px 0px 70px 0px;
    border: #2980b9 4px solid;
  }

  .aws_key_id{
    background:#ecf0f1;
    border: #ccc 1px solid;
    border-bottom: #ccc 2px solid;
    padding: 8px;
    width:250px;
    color:#AAAAAA;
    margin-top:10px;
    font-size:1em;
    border-radius:4px;
  }

  .aws_key{
    border-radius:4px;
    background:#ecf0f1;
    border: #ccc 1px solid;
    padding: 8px;
    width:250px;
    font-size:1em;
  }

  .btn{
    background:#2ecc71;
    width:125px;
    padding-top:5px;
    padding-bottom:5px;
    color:white;
    border-radius:4px;
    border: #27ae60 1px solid;

    margin-top:20px;
    margin-bottom:20px;
    float:left;
    margin-left:80px;
    font-weight:800;
    font-size:0.8em;
  }

  .btn:hover{
    background:#2CC06B;
  }

  #btn2{
    float:left;
    background:#3498db;
    width:125px;  padding-top:5px;
    padding-bottom:5px;
    color:white;
    border-radius:4px;
    border: #2980b9 1px solid;

    margin-top:20px;
    margin-bottom:20px;
    margin-left:10px;
    font-weight:800;
    font-size:0.8em;
  }

  #btn2:hover{
  background:#3594D2;
  }
```

project_root/cloud_download/static/cloud_download/style.css

```
@import url(https://fonts.googleapis.com/css?family=Open+Sans);

/*Page styles*/
html { height: 100%; }

h1 {
    color: #ddd;
    text-align: center;
    font: 38px 'Open Sans', Arial, sans-serif;
}

h2 {
    color: #ddd;
    text-align: center;
    font: 15px 'Open Sans', Arial, sans-serif;
}

h3 {
  color: #ddd;
  text-align: center;
  font: 18px 'Open Sans', Arial, sans-serif;
}
body {
  height: 100%;
  margin: 0;
  background: #363636;
  align-items: center;
}
.outer {
    position: absolute;
    left: 30%;
}

.outer2 {
    position: absolute;
    left: 44%;
}

.outer3 {
  position: absolute;
  left: 27%;
}
.button {
    border: none;
    color: white;
    padding: 15px 45px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
  font: 16px 'Open Sans', Arial, sans-serif;
  margin: 8px 4px;
  cursor: pointer;
  vertical-align: top;
  border-radius: 3px
}

.button1 {
  background-color: #484848;
  color: #ddd;
  border: 2px solid #008CBA;
}
.button_index {
  border: none;
  color: white;
  padding: 15px 50px;
  text-align: center;
  text-decoration: none;
  display: inline-block;
  font: 30px 'Open Sans', Arial, sans-serif;
  margin: 8px 4px;
  cursor: pointer;
  vertical-align: top;
  border-radius: 3px
}
.button2 {
  background-color: #484848;
  color: #ddd;
  border: 2px solid #008CBA;
}
.button1:hover{
  background: rgb(25, 63, 70);
  }
.button2:hover{
  background: rgb(25, 63, 70);
  }

  .btn{
    background:#484848;
    width:125px;
    padding-top:5px;
    padding-bottom:5px;
    color:#ddd;
    border-radius:3px;
    border: #008CBA 2px solid;
    margin-top:20px;
    margin-bottom:20px;
    float:left;
    margin-left:180px;
    font-weight:800;
    font-size:0.8em;
    }

    .btn:hover{
      background: rgb(25, 63, 70);
    }

.header {
    position: relative;
    top: 0;
}

.inner {
    position: relative;
}
.boxes {
  margin: auto;
  padding: 50px;
  background: #484848;
  border-radius: 7px;
}
.buttonHold {
    text-align: center;
}
/*Checkboxes styles*/
input[type="checkbox"] { display: none; }

input[type="checkbox"] + label {
  display: block;
  position: relative;
  padding-left: 35px;
  margin-bottom: 20px;
  font: 14px/20px 'Open Sans', Arial, sans-serif;
  color: #ddd;
  cursor: pointer;
  -webkit-user-select: none;
  -moz-user-select: none;
  -ms-user-select: none;
}

input[type="checkbox"] + label:last-child { margin-bottom: 0; }

input[type="checkbox"] + label:before {
  content: '';
  display: block;
  width: 20px;
  height: 20px;
  border: 1px solid #6cc0e5;
  position: absolute;
  left: 0;
  top: 0;
  opacity: .6;
  -webkit-transition: all .12s, border-color .08s;
  transition: all .12s, border-color .08s;
}

input[type="checkbox"]:checked + label:before {
  width: 10px;
  top: -5px;
  left: 5px;
  border-radius: 0;
  opacity: 1;
  border-top-color: transparent;
  border-left-color: transparent;
  -webkit-transform: rotate(45deg);
  transform: rotate(45deg);
}

ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
}

li {
  font: 200 20px/1.5 Helvetica, Verdana, sans-serif;
  border-bottom: 1px solid #008CBA;
  text-align: center;
}

li:last-child {
  border: none;
}
.table {
display: table;   /* Allow the centering to work */
margin: 0 auto;
}
li a {
  text-decoration: none;
  color: #ddd;
  display: block;
  width: 600px;
  text-align: center;
  -webkit-transition: font-size 0.3s ease, background-color 0.3s ease;
  -moz-transition: font-size 0.3s ease, background-color 0.3s ease;
  -o-transition: font-size 0.3s ease, background-color 0.3s ease;
  -ms-transition: font-size 0.3s ease, background-color 0.3s ease;
  transition: font-size 0.3s ease, background-color 0.3s ease;
}

li a:hover {
  font-size: 30px;
}
```

project_root/cloud_download/templates/cloud_download/aws_login.html

```
<!--
file page template for cloud backup application

Application:     Cloud Backup
File:                 /cloud_backup/cloud_download/templates/cloud_download/aws_login.html
Description:    html template for files
Language:       Python 3.8 Django 2.2
Dev Env:         Linux x64

Authors:                Ryan Breitenfeldt
                        Noah Farris
                        Trevor Surface
                        Kyle Thomas
Class:              CptS 421/423 Fall '19 Spring '20
University:    Washington State University Tri-CIties
-->
<html>
    <head>
        {% load static %}
        <link rel="stylesheet" type="text/css" href="{% static 'cloud_download/aws.css' %}">
        <meta charset="UTF-8">
    </head>

    <form method="post">
        <div class="box">
        <h1>Amazon Web Services</h1>
        <h2>Login</h1>

        {% csrf_token %}
        {% for field in form.hidden_fields %}
            {{ field }}
        {% endfor %}

        {% for field in form.visible_fields %}
            {{ field }}
        {% endfor %}
        <input type="submit" value="Submit" div class="btn">
        </div> <!-- End Box -->

    </form>
    {% if messages %}
    <ul class="messages">
        {% for message in messages %}
        <script>alert("{{ message }}")</script>
        {% endfor %}
    </ul>
    {% endif %}
</html>
```

project_root/cloud_download/templates/cloud_download/files.html

```
<!--
file page template for cloud backup application

Application:     Cloud Backup
File:                 /cloud_backup/cloud_download/templates/cloud_download/files.html
Description:    html template for files
Language:       Python 3.8 Django 2.2
Dev Env:         Linux x64

Authors:          Ryan Breitenfeldt
                        Noah Farris
                        Trevor Surface
                        Kyle Thomas
Class:              CptS 421/423 Fall '19 Spring '20
University:    Washington State University Tri-Cities
-->
<html>
    <head>
        {% load static %}
        <link rel="stylesheet" type="text/css" href="{% static 'cloud_download/style.css' %}">
    </head>
        <div class = "outer">
            <div class = "header"><h1>Cloud Download Application</h1></div>
            <div class = "inner">
            <form id="aclass" action="" method="post">
                {% csrf_token %}
                <div class="boxes">
                <input type="checkbox" id = "sel_all" name = "box" onclick="toggle(this)">
                <label for="sel_all"> Select All </label>
                {% for file in files %}
                <input type="checkbox" id = "{{ file }}" value="{{ file }}" name="box">
                <label for="{{ file }}">{{ file.name }} </label>
                {% empty %}
                    <label name="box" for="no_files">No files found... </label>
                {% endfor %}
                </div>
                <input class = "button button1" type="submit" value="Download">
                <a class="button button2" id="back" href="/cloud">Start Over</a>
            </form>
            </div>
        </div>

    <script>
        function toggle(source) {
          checkboxes = document.getElementsByName('box');
          for(var i=0, n=checkboxes.length;i<n;i++) {
            checkboxes[i].checked = source.checked;
          }
        }
    </script>
</html>
```

project_root/cloud_download/templates/cloud_download/googleIndex.html

```
<!--https://developers.google.com/identity/sign-in/web/server-side-flow -->
<!-- The top of file index.html -->
<html itemscope itemtype="http://schema.org/Article">
<head>
  <!-- BEGIN Pre-requisites -->
  <script src="//ajax.googleapis.com/ajax/libs/jquery/1.8.2/jquery.min.js">
  </script>
  <script src="https://apis.google.com/js/client:platform.js?onload=start" async defer>
  </script>
  <!-- END Pre-requisites -->

  <!-- Continuing the <head> section -->
    <script>
        function start() {
          gapi.load('auth2', function() {
            auth2 = gapi.auth2.init({
              client_id: '204730000731-c0gs1os80ucalj6mto9c1etmaee70is7.apps.googleusercontent.com',

            });
          });
        }
      </script>
    </head>
    <body>
      <!-- ... -->
<!-- Add where you want your sign-in button to render -->
<!-- Use an image that follows the branding guidelines in a real app -->
<button id="signinButton">Sign in with Google</button>
<script>
  $('#signinButton').click(function() {
    // signInCallback defined in step 6.
    auth2.grantOfflineAccess().then(signInCallback);
  });
</script>
<!-- Last part of BODY element in file index.html -->
<script>
    function signInCallback(authResult) {
      if (authResult['code']) {

        // Hide the sign-in button now that the user is authorized, for example:
        $('#signinButton').attr('style', 'display: none');

        // Send the code to the server
        $.ajax({
          type: 'POST',
          url: 'http://localhost:8000',
          // Always include an `X-Requested-With` header in every AJAX request,
          // to protect against CSRF attacks.
          headers: {
```

```
        'X-Requested-With': 'XMLHttpRequest'
      },
      contentType: 'application/octet-stream; charset=utf-8',
      success: function(result) {
        // Handle or verify the server response.
      },
      processData: false,
      data: authResult['code']
    });
  } else {
    // There was an error.
  }
}
</script>
</body>
</html>
```

project_root/cloud_download/templates/cloud_download/index.html

```
<!--
cloud selection page template for cloud backup application

Application:     Cloud Backup
File:                /cloud_backup/cloud_download/templates/cloud_download/index.html
Description:    html template for index page
Language:       Python 3.8 Django 2.2
Dev Env:         Linux x64

Authors:          Ryan Breitenfeldt
                        Noah Farris
                        Trevor Surface
                        Kyle Thomas
Class:              CptS 421/423 Fall ’19 Spring ’20
University:    Washington State University Tri-Cities
-->
<html>
    <head>
        {% load static %}
        <link rel="stylesheet" type="text/css" href="{% static ’cloud_download/style.css’ %}">
        <meta charset="UTF-8">
    </head>

    <body>
        <h1>Cloud Backup Application</h1>
        <h2>Please select a cloud application and click ’Submit’.</h2>
        <div class = "outer2">
        <div class = "inner">
          <div class = "button-group">
            <form action="" method="post">
              {% csrf_token %}
              <div class>
              <select class = "button_index button1" id="platform" name="platform">
                {% for platform in platforms %}
                <option value="{{platform}}">{{platform}}</option>
                {% empty %}
                <option value="empty">error, no platforms found</option>
                {% endfor %}
              </select>
             </div>
             <div>
              <input class = "button button1" type="submit" value="Submit">
            </div>
            </form>
          </div>
      </div>
    </div>
    </body>
</html>
```

project_root/cloud_download/templates/cloud_download/success.html

```
<!--
file page template for cloud backup application

Application:     Cloud Backup
File:                 /cloud_backup/cloud_download/templates/cloud_download/files.html
Description:    html template for files
Language:       Python 3.8 Django 2.2
Dev Env:         Linux x64

Authors:          Ryan Breitenfeldt
                        Noah Farris
                        Trevor Surface
                        Kyle Thomas
Class:              CptS 421/423 Fall ’19 Spring ’20
University:    Washington State University Tri-Cities
-->
<html>
    <head>
        {% load static %}
        <link rel="stylesheet" type="text/css" href="{% static ’cloud_download/style.css’ %}">
    </head>
    <body>
        <h1>Cloud Backup Application</h1>
        <h3>Downloaded Files:</h3>
        <div class="outer3">
            <div class = "table">
            <ul>
                {% for file in files %}
                <li><a>{{ file.name }}</a></li>
                {% empty %}
                <li><a> No files selected for downloading </a></li>
                {% endfor %}
            </ul>
            </div>
            <a class="button btn" id="back" href="/cloud">Start Over</a>
        </div>
    </body>

</html>
```

project_root/cloud_download/platforms/__init__.py

```
#!/usr/bin/env python3

""" Cloud Platforms Module

Application:     Cloud Backup
File:                 /cloud_backup/cloud_download/platforms/__init__.py
Description:    Platform module initializer
Language:       Python 3.8 Django 2.2
Dev Env:         Linux x64

Authors:          Ryan Breitenfeldt
                        Noah Farris
                        Trevor Surface
                        Kyle Thomas
Class:              CptS 421/423 Fall '19 Spring '20
University:    Washington State University Tri-CIties
"""

from . import dropbox_script, gDriveDownloader, Api_keys, aws


__authors__ = ['Ryan Breitenfeldt', 'Noah Farris', 'Trevor Surface', 'Kyle Thomas']
```

project_root/cloud_download/platforms/aws.py

```
import boto3
import json
import os

class aws():
    '''
        self._ec2_client: This is the EC2 Client
        self._s3_client: This is the S3 Client
        self._s3_resource: This is the S3 resource.

        Parameters: access_key_id, access_key, region (default: 'us-west-2')
    '''
    def __init__(self, access_key_id, access_key, region = 'us-west-2'):
        self._ec2_client = boto3.client('ec2', region_name = region,
                        aws_access_key_id=access_key_id,
                        aws_secret_access_key=access_key)
        self._s3_client = boto3.client('s3',region_name=region,
                        aws_access_key_id=access_key_id,
                        aws_secret_access_key=access_key)
        self._s3_resource = boto3.resource('s3',region_name=region,
                        aws_access_key_id=access_key_id,
                        aws_secret_access_key=access_key)

    '''
        Return list of available images as a JSON response.
    '''
    def get_image_list(self):
        images = self._ec2_client.describe_images(Owners=['self'])
        return images

    '''
        Returns a list of all buckets that an AWS Account has access to.
    '''
    def get_buckets(self):
        response = self._s3_client.list_buckets()
        buckets_json = response['Buckets']
        buckets_list = []
        for i in buckets_json:
            buckets_list.append(i['Name'])
        return buckets_list

    '''
        Lists all images within a bucket and returns this as a list of dicts.

        Parameters: bucket_name
    '''
    def list_images_in_bucket(self, bucket_name):
        bucket = self._s3_resource.Bucket(bucket_name)
        files_list = []
        for my_bucket_object in bucket.objects.all():
            files_list.append({'name': my_bucket_object.key})

        data_set = files_list
        return data_set
    '''
        Exports an image to a bucket, you can access a list of images but if they aren't in a bucket
        you cannot download them.

        Parameters: image_id, role_name, bucket_name, format (default = 'VMDK')
    '''
    def export_to_bucket(self, image_id, role_name, bucket_name, format='VMDK'):
        response = self._ec2_client.export_image(
            Description='string',
            DiskImageFormat=format,
            DryRun=False,
            ImageId=image_id,
            RoleName=role_name,
            S3ExportLocation={
                'S3Bucket': bucket_name
            }
        )
        return response

    '''
        Given a bucket and a file name this will download that file.

        Parameters: bucket_name, file_name
    '''
    def download_image(self, bucket_name, file_name):
        bucket = self._s3_resource.Bucket(bucket_name)
        bucket.download_file(file_name, "/Users/noahfarris/Desktop/downloads" +"/" + os.path.basenam
```

project_root/cloud_download/platforms/class_template.py

```
'''
For Admins:
    This Template class is to help developers add additional cloud platforms to the software
'''

'''
class [Platform_NAME](object):
    def __init__(self):
        ## For attributes use convention [NAME]_[FUNCTION]_[PARAM]
        ## The below attributes are strongly recomended to have within new platform classes
        __[Platform_NAME]__user_download_path       #Consider this to be a private function
        __[Platform_NAME]_get_files_list_result     #Conisder this to be a private function
        __[Platform_NAME]_entries_to_download_list  #Consider this to be a private function
        [Platform_NAME]_format_dict
        [Platform_NAME]_flat_dict

    ##  For Admins - the following three functions are designed based off of the OAuth2 requriements
    ##  If the platform that is being added does not have OAuth2 authentication the following three
    ##  are unneccessay.

    def [Platform_NAME]_authentication_flow(self, request):
        # This function call requires a redirect_uri that may need to need to be specified within th
        # of the platform being used. This class should return the Flow component of the OAuth2 auth
        # will likey need a request.session to help manage state.

    def [Platform_NAMe]_authentication_start(self):
        # This function will initiate the starting parameters for the OAuth2 authentication to use t
        # capability appropriately, after this method is created, this object will need to be instat
        # within the veiws.py script. Additionally the urls.py/urlpattern needs to be updated with t
            path('[Platform_NAME]-auth-start', views.[Platform_NAME].[Platform_NAME]_authentication_
        # When the user selects the new plaform, the Views Index class will need to return a redirec
        # redirect('[Platform_NAME]-auth-start')
        # Finally this method should instatiate a redirect_url to created by the authentication_flow
        # The return will need a django.shortcuts.redirect() to the redirect_url to allow the user t

    def [Platform_NAME]_authentication_finish(self):
        # This fucntion will finalize the OAuth2 authentication. After this method is created, this
        # within the veiws.py script. Additionally the urls.py/urlpattern needs to be updated with t
            path('[Platform_NAME]-auth-finish', views.[Platform_NAME].[Platform_NAME]_authentication
        # The authentication_flow, may require a redirect_uri that includes the full uri to the [Pla
        # This function will catch any errors that arise within the OAuth2 authentication flow.
        # This function will also call all neccessary function to gather files and folders of the ap
        # The returned object will then need to be formated into either a dict specified as:
            {'path':'', 'dirs':[], 'files':[]} in which every dir contains the same dict and dict li
        # Currently the software requires a flat_dict, in which all files are pulled out along with
        # attributes neccessary for downloading. The flact dict will be written as
            {'file':[]} with all the attributes required for downloading files.

    def [Platform_NAME]_get_files_list(self):
        # This is a generic template for gathering files, use selected platforms api to gather the a
        # Often this may have a list of file entires as a return

    def [Platform_NAME]_format_entries_list(self):
        # This function will take the return of [NAME]_get_files_list
        # And provide the neccessary information to pass to the views context to display the applica

    def [Platform_NAME]_download_selected_entries(self, path):
        # This function will download the list returned by [NAME]_select_entries_to_download
        # Using the [NAME]_download_path to specify the location on the server where the files and f
        # This function should also include a warning that if a file is alread known on a system, it
'''
```

project_root/cloud_download/platforms/dropbox_script.py

```
#!/usr/bin/env python3

''' Dropbox Platform
Application:     Cloud Backup
File:                 /cloud_backup/cloud_download/platforms/dropbox_script.py
Description:    Dropbox cloud platform
Language:       Python 3.8 Django 2.2
Dev Env:         Linux x64

Authors:          Trevor Surface
Class:              CptS 421/423 Fall '19 Spring '20
University:    Washington State University Tri-CIties
'''
import os
import dropbox
import json
from django.shortcuts import redirect
from . import Api_keys
import ast
__author__ = "Trevor Surface"

'''
dropbox.oauth.DropboxOAuth2Flow(key, secret, redirect_uri, session, token_type)
dropbox.oauth.DropboxOAuth2Flow(key, secret, redirect_uri, session, token_type).start()
    returns: redirect URL STR
dropbox.oauth.DropboxOAuth2Flow(key, secret, redirect_uri, session, token_type).finish(GET_parameter
    returns: OAuth2FlowResult
    raises: dropbox.oauth.BadRequestException
            dropbox.oauth.BadStateException
            dropbox.oauth.CsrfException
            dropbox.oauth.NotApprovedException
            dropbox.oauth.ProviderException
dropbox.dropbox.Dropbox(access_token)
dropbox.dropbox.Dropbox(access_token).list_folder_result(path)
  returns: dropbox.files.Metadata()
      includes:   files.Metadata.name
                  files.Metadata.path_lower
                  files.Metadata.path_display
                  files.Metadata.parent_shared_folder_id
  raises: dropbox.exceptions.ApiError()
      includes:   request_id
                  error
                  user_message_text
dropbox.dropbox.Dropbox(access_token).files_list_folder_continue(cursor)
  retuns: dropbox.files.Metadata()
  raises: dropbox.exceptions.ApiError()
dropbox.dropbox.Dropbox(access_token).files_download_to_file(download_path, path)
   returns: dropbox.files.FileMetadata()
       includes:   files.FileMetadata.id
```

```
                   files.FileMetadata.client_modified
                   files.FileMetadata.server_modified
                   files.FileMetadata.rev
                   files.FileMetadata.size
                   files.FileMetadata.media_info
                   files.FileMetadata.symlink_info
                   files.FileMetadata.sharing_info
                   files.FileMetadata.is_downloadable
                   files.FileMetadata.export_info
                   files.FileMetadata.property_groups
                   files.FileMetadata.has_explicit_shared_members
                   files.FileMetadata.content_hash
dropbox.dropbox.Dropbox(access_token).files_download_zip_to_file(download_path, path)
    returns: dropbox.files.DownloadZipResult
    raises: dropbox.exceptions.ApiError()
        includes:   dropbox.files.DownloadZipError
'''

class DropBox(object):
    def __init__(self):
        '''
        __init__
        Attributes to modify:
        self.__dropbox_api_key, self.__dropbox_api_secret
            Admin needs to create an app via the dropbox app console: https://dropbox.com/developer
            Once the app has been created, Admin will get both a KEY and a SECRET, both of which
            should be added to an API_Keys.py file, with parameter names Dropbox_Api_key and
            Dropbox_Api_secret. Make sure to include the file in the .gitignore.
        self.__dropbox_download_path
            Admin needs to specify a location on the server that will store the downloaded files,
            additionally, for larger applications, the user may be asked to specify a location.
        '''
        self.dbx = None
        '''
            This attribute will contain the dropbox object after a users has been authenticated
        '''
        self.__dropbox_api_key = Api_keys.Dropbox_Api_key
        '''
        See __init__, Attributes to modify
        '''
        self.__dropbox_api_secret = Api_keys.Dropbox_Api_secret
        '''
        See __init__, Attributes to modify
        '''
        self.__dropbox_authentication_oauth_result = ""
        '''
            This attribute will contain the oath information, which includes the access token
            See the dropbox function definition at the top of the screen
        '''
        self.__dropbox_get_files_return = ""
        '''
```

```
            This attribute will contain the result returned by the dropbox.get_file_list_folder()
            api call, defined in the above api definitions
        '''
        self.__dropbox_get_files_list_result = []
        '''
            This attribute will contain a list of results with information from the previous attribu
            accessing the .entries parameter of the return
        '''
        self.__dropbox_download_path = os.environ['HOME'] + "/Downloads"
        '''
            See __init__, Attributes to modify
        '''
        self.dropbox_format_dict = {'path': '', 'dirs':[], 'files':[]}
        '''
            This attribute holds a dict for each folder, containing a variable list of folders in ea
            and the included files within a given folder
        '''
        self.dropbox_flat_dict = {'files':[]}
        '''
            This attribute holds a flattened dict that will only show users files, not including the
            of the neccessary files.
        '''
        self.__dropbox_files_paths = []
        '''
            This attribute stores the paths of the files returned from the dropbox.get_file_list_fol
        '''

    def dropbox_auth_flow(self, request):
        '''
            This function generates the flow object that needs to be passed through
            the authentication_start() method and the authentication_finish() method

        For Admin's:
            The redirect_uri parameter needs to be set in both the urls.py/urlpatterns as
            a path, which calls the dropbox_authetication_finish function. Additionally
            in the app console for dropbox (https://dropbox.com/developer) the same redirect_uri
            needs to be specified for the generated application.
        '''
        redirect_uri = "http://localhost:8000/cloud/dropbox-auth-finish"
        return dropbox.oauth.DropboxOAuth2Flow(self.__dropbox_api_key, self.__dropbox_api_secret, re

    def dropbox_authentication_start(self, web_app_session):
        '''
            This function generates the redirect for the application users to login to their
            dropbox accounts
        '''
        authorize_url = self.dropbox_auth_flow(web_app_session.session).start()
        return redirect(authorize_url)

    def dropbox_authentication_finish(self, request):
        '''
```

```
            This function finalizes the authentication with the application user

        For Admins:
            The exceptions for Bad State, needs to be updated to the path that initialized the
            authentication. This needs to be updated in the urls.py/urlpatterns, with a path that
            calls the dropbox_authentication_start method.

            Additionally the Not Approved Exception return will need to be updated so that the appli
            user is redirected to the home page of the webiste.
        '''
        try:
            self.__dropbox_authentication_oauth_result = self.dropbox_auth_flow(request.session).fin
        except dropbox.oauth.BadRequestException as e:
            raise e
        except dropbox.oauth.BadStateException as e:
            return redirect("http://localhost:8000/cloud/dropbox-auth-start/")
        except dropbox.oauth.CsrfException as e:
            return HttpResponseForbidden
        except dropbox.oauth.NotApprovedException as e:
            flash('Not approved?  Why not?')
            return redirect("http://localhost:8000/cloud")
        except dropbox.oauth.ProviderException as e:
            logger.log("Auth error: %s" % (e,))
            raise e
        self.dbx = dropbox.dropbox.Dropbox(self.__dropbox_authentication_oauth_result.access_token)
        ''' The above function call uses the oauth_return to generate the dropbox object'''
        self.dropbox_get_files_list()
        ''' The above function uses the dropbox api's to gather the list of folders and files the au
            has in their dropbox account, for further details see the function definition below'''
        self.dropbox_format_entries_recur(self.dropbox_format_dict, 1, self.__dropbox_files_paths)
        ''' The above funtion generates a dict, or json, depending upon the use of the #json.dumps c
            function defined below. The admin can change the views page to either display the json o
            #self.dropbox_format_dict = json.dumps(self.dropbox_format_dict)
        self.dropbox_dict_flatten_recur(self.dropbox_format_dict)
        ''' The above function flattens the dict to only show the files that are available on the ap
            dropbox account, this can be removed if the Admin would like to show the directories WAR
            is only set up to show a flat_dict object, and will need further editing to enable the f
        return redirect('/cloud/files')
        ''' The final part of the function will redirect the user to the files view.py page, where t
            be displayed for them to choose which to download to the server'''

    def dropbox_get_files_list(self):
        '''
            This function calls the files_list_to_folder() dropbox function, gathering the list of a
            files and folders. The application will then generate two lists in order to create a pro
            in the cloud/files/ view.
        '''
        try:
            self.__dropbox_get_files_list_return = self.dbx.files_list_folder("", recursive=True)
        except dropbox.exception.ApiError as e:
            raise e
        for dropbox_files in self.__dropbox_get_files_list_return.entries:
            self.__dropbox_get_files_list_result.append(dropbox_files)
            self.__dropbox_files_paths.append(dropbox_files.path_lower)

    def dropbox_format_entries_recur(self, build_dict, level, file_list):
        '''
            This function is a recursive algorithm designed to create a multi layered dict that will
            root folder, and the subsequent folders that a application user may have available in th
            WARNING: For the Admins, the current functionaliry is based on the fact that a user does
            in their directory names, however, if they do the generated dict will be invalid and not
            files and folders, this would need to be changed to allow file/folder detection.
        '''
        for file_paths in file_list:
            if file_paths.count('/') == level and build_dict['path'] in file_paths:
                dict = {'path': file_paths}
                if '.' in file_paths:
                    dict['name'] = file_paths.split('/')[len(file_paths.split('/')) - 1]
                    if dict not in build_dict['files']:
                        build_dict['files'].append(dict)
                else:
                    dict['dirs'] = []
                    dict['files'] =[]
                    self.dropbox_format_entries_recur(dict, level + 1, file_list)
                    if dict not in build_dict['dirs']:
                        build_dict['dirs'].append(dict)

    def dropbox_dict_flatten_recur(self, dir):
        '''
            This function uses a recursive algorithm to strip off directories from a application use
            system, only displaying all of their current files.
        '''
        for files in dir['files']:
            self.dropbox_flat_dict['files'].append(files)
        for dirs in dir['dirs']:
            self.dropbox_dict_flatten_recur(dirs)

    def dropbox_download_selected_entries(self, download_list):
        '''
            This function is called after an application user has selected the files they wish to do

        For Admins:
            This function works off the premise that users do not put a '.' in their folder names, t
            the case, in which a change would need to be made to allow file/folder detection. Additi
            the software will not detect if there is a currently named file in the system with same
            file. In which case the download will be overwritten by the new download.
        '''
        for files in download_list:
            if '.' in files['path']:
                try:
                    self.dbx.files_download_to_file(self.__dropbox_download_path + '/' + files['name
                except dropbox.exceptions.ApiError as e:
```

```
                raise e
        else:
            try:
                self.dbx.files_download_zip_to_file(self.__dropbox_download_path + '/' + files['
            except dropbox.exceptions.ApiError as e:
                raise e
```

project_root/cloud_download/platforms/gDriveDownloader.py

```
#!/usr/bin/env python3

""" Google Drive Cloud Platform

Application:     Cloud Backup
File:                 /cloud_backup/cloud_download/platforms/gDriveDownloader.py
Description:    Google Drive Cloud Platform
Language:       Python 3.8 Django 2.2
Dev Env:         Linux x64

Authors:          Ryan Breitenfeldt
Class:              CptS 421/423 Fall '19 Spring '20
University:    Washington State University Tri-CIties
"""

from __future__ import print_function
import pickle
import os.path
import os
import io
from googleapiclient.discovery import build
from googleapiclient.http import MediaIoBaseDownload
from google_auth_oauthlib.flow import InstalledAppFlow
from google_auth_oauthlib.flow import Flow
from google.auth.transport.requests import Request
import google.oauth2.credentials
from google.oauth2.credentials import Credentials
from django.shortcuts import render, redirect

__author__ = 'Ryan Breitenfeldt'

#googleapiclient.discovery.build
#googleapiclient.discovery.build.files().list()
#googleapiclient.http.MediaIoBaseDownload
#google_auth_oauthlib.flow.InstalledAppFlow
#google.auth.transport.requests.Request


class GDriveDownloader():
    def __init__(self,creds=None,service=None):
        self.GDriveDownloader_creds = creds #stores the credetials from google
        self.GDriveDownloader_SCOPES = ['https://www.googleapis.com/auth/drive'] # the scope of what
        self.GDriveDownloader_service = service # the engine for googled api
        self.GDriveDownloader_file_List = None # holds all the meta data and file info from Drive
        self.GDriveDownloader_json = {'files':[]} # the json to be used by the ui
        self.GDriveDownloader_files_to_download = [] # the list of files to download

    #this functoin is for authenticting as a standalone class
    #use the redirect functions below for authentication in Django
    def GDriveDownloader_authentication(self):
        # If there are no (valid) credentials available, let the user log in.
        if not self.GDriveDownloader_creds or not self.GDriveDownloader_creds.valid:
            if self.GDriveDownloader_creds and self.GDriveDownloader_creds.expired and self.GDriveDo
                creds.refresh(Request())
            else:
                flow = InstalledAppFlow.from_client_secrets_file('client_secret_204730000731-c0gs1os
                self.GDriveDownloader_creds = flow.run_local_server(port=0)

    # saves the credencials to a file
    def GDriveDownloader_save_Token(self):
        with open('token.pickle', 'wb') as token:
            pickle.dump(self.GDriveDownloader_creds, token)

    # load creds from a file
    def GDriveDownloader_load_Token(self):
        if os.path.exists('token.pickle'):
            with open('token.pickle', 'rb') as token:
                creds = pickle.load(token)

    # creates the google drive service from the credentials
    def GDriveDownloader_build_Service(self):
        self.GDriveDownloader_service = build('drive', 'v3', credentials=self.GDriveDownloader_creds

# loops over GDriveDownloader_files_to_download and downloads each file to the working directory
    def GDriveDownloader__download_File(self):
        os.chdir("/Users/noahfarris/Desktop/downloads")
        for file_id in self.GDriveDownloader_files_to_download:
            request = self.GDriveDownloader_service.files().get_media(fileId=file_id["id"]) # reques
            fh = io.BytesIO()
            downloader = MediaIoBaseDownload(fh, request) # makes the downloader for the file
            done = False
            while done is False:
                status, done = downloader.next_chunk()
                print("Download {}%".format(int(status.progress() * 100)))
            f = open(file_id["name"], 'wb')
            f.write(fh.getvalue())
        os.chdir('/Users/noahfarris/Desktop/CAPSTONE_FINAL/git/cloud-backup/cloud_backup')
# makes a query to google drive to get the files. it filters out folders and google propriatary file
    def GDriveDownloader_get_Files(self):
        page_token = None
        self.GDriveDownloader_json['files'].clear() # clears the list of files to prevent duplicatio
        while True:
            self.GDriveDownloader_file_List = self.GDriveDownloader_service.files().list(q="mimeType
            for file in self.GDriveDownloader_file_List["files"]:
                self.GDriveDownloader_json['files'].append({"name": file["name"],"id": file["id"]})
            page_token = self.GDriveDownloader_file_List.get('nextPageToken', None)
            if page_token is None:
                break
```

```
# takes dictionary as input in this formate: {"name": "file name", "id": " file id"}
    def GDriveDownloader_add_file_to_download(self,fileId):
        self.GDriveDownloader_files_to_download.append(fileId)

    #creates the autheration flow for the redirect
    def GDriveDownloaded_authentication_flow(self):
        redirect_uri = "http://localhost:8000/cloud/google-auth-finish/"
        return Flow.from_client_secrets_file(
            'client_secret_204730000731-c0gs1os80ucalj6mto9c1etmaee70is7.apps.googleusercontent.com.
            scopes=self.GDriveDownloader_SCOPES,
            redirect_uri=redirect_uri)

    # redirects to the autheration url
    def GDriveDownloaded_authentication_start(self, request):
        auth_uri = self.GDriveDownloaded_authentication_flow().authorization_url()[0]
        return redirect(auth_uri[:-20])

    # takes a json returned from google and turns it into a credential object
    # it builds the service and gets the list of files.
    def GDriveDownloaded_authentication_finish(self, request):
        try:
            self.GDriveDownloader_creds = self.GDriveDownloaded_authentication_flow().fetch_token(co
            #print(self.GDriveDownloader_creds)
            self.GDriveDownloader_creds = Credentials(self.GDriveDownloader_creds['access_token'])
            self.GDriveDownloader_build_Service()
            self.GDriveDownloader_get_Files()
        except Exception as e:
            raise e
        return redirect('/cloud/files')
```

Test for Cloud Backup Application

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

ID	**Requirement**	**Result**	**Test Case**	**Expected**	**Actual**
01	Users selected files download from Dropbox	PASS	Select dropbox files to download	files downloaded	files downloaded
02	Users selected files download from Google Drive	PASS	Select Google Drive files to download	files downloaded	files downloaded
03	Users selected files download from AWS	PASS	Select AWS files to download	files downloaded	files downloaded
04	Can view user's Dropbox files	PASS	Authenticate to Dropbox	list of files	list of files
05	Can view user's Google Drive files	PASS	Authenticate to Google	list of files	list of files
06	Can view user's AWS files	PASS	Authenticate to AWS	list of files	list of files
07	Supported platforms display in menu	PASS	Navigate to app	Dropbox, Google, AWS	Dropbox, Google, AWS
08	Can navigate to app from localhost:8000	PASS	Navigate to localhost:8000	Index page	Index page
09	Navigating back to application starts over	PASS	Navigate to localhost:8000	Index page	Index page
10	User redirected to Google authentication	PASS	Select Google Drive	Google Login Page	Google Login Page
11	User redirected to Dropbox authentication	PASS	Select Dropbox	Dropbox login page	Dropbox login page
12	User redirected to AWS authentication	PASS	Select AWS	AWS login page	AWS login page
13	Emtpy pulldown menu w/o platforms enabled	PASS	Navigate to app w/o platforms	Empty menu	Empty menu
14	Empty list of files displays	PASS	Navigate to platform without files	No files message	No files message
15	Select All button selects all files	PASS	Press select all button	All files selected	All files selected
16	No files are downloade when none selected	PASS	Press download button w/o files selected	No files downloaded	no files downloaded
17	Start Over button on files returns to index	PASS	Press start over button	redirect to main menu	redirect to main menu
18	Start Over button on downloaded files returns to index	PASS	Press start over button	redirect to main menu	redirect to main menu
19	Application is standalone django app	PASS	Root URI and `/cloud/` direct to app	Directed to main page	Directed to main page
20	AWS form captures user input	PASS	Enter credentials in form	AWS accepts credentials	AWS accepts credentials

User's Guide for Cloud Backup

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

List of Figures

Revision History

Revision	Date	Author(s)	Description
1.0	05.03.2020	KT	Document Completion
0.1	04.20.2020	KT	Document Creation

1 Introduction

This document is an instruction manual for users on how to use the *Cloud Backup* software. There will be three parts. The first part will be how user's of application will interact with the application with the second part covering how system administrators will deploy the application. Followed by the third part, which is how developers will expand the application to include more cloud platforms.

2 Users

The users of the application are people who are visiting the URL of the application to import their files into the resiliency platform.

2.1 Invocation

To begin using the application the user will open their web browser and navigate to the URL that contains the application. This will be provided by the administrators hosting the application.

2.2 Stopping

To terminate their interaction with the application users will simply close their browser tab or window containing the application or navigate to another site. This can be done during any step of using the application.

2.3 Selecting a Cloud Provider

On the main page of the application the user will be presented with a pull down menu containing each cloud platform that is supported by the application. Selecting one of these options and clicking the "Submit" button will proceed the user to the next step (authenticating to the cloud platform).

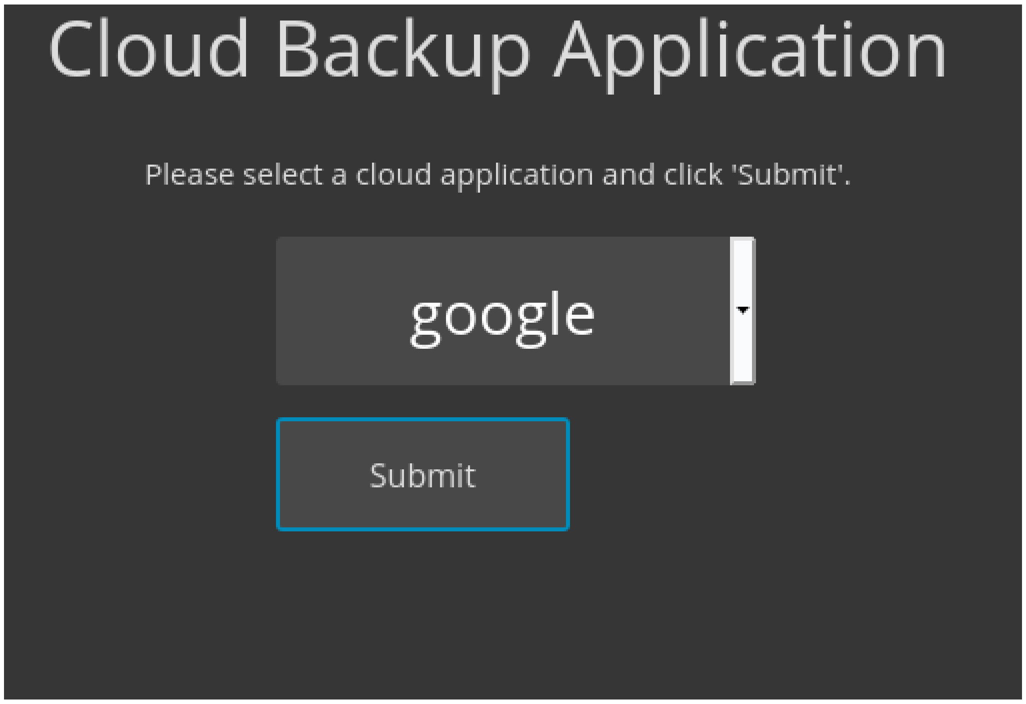


Figure 1: The cloud platform selection screen.

2.4 Authenticating to the Cloud Platform

After selecting a platform the user will redirected to an authentication page. Depending on the platform this page will be different and require different authentication methods. For example, the Amazon Web Services platform will direct the user to a page where they will enter their AWS client secret and AWS client token into a text box. Other platforms such as Google will redirect the user to Google's authentication platform. From there they will enter their Google username and platform.

Some of these application during authentication will also require that the user grant permissions for the application to interact with their account. The application needs these permissions in order to function properly.

After the user authenticates to their account on the cloud platform they will be redirected back to the application for the next step (Selecting and Downloading Files).

2.5 Selecting and Downloading Files

After the user has authenticated to their cloud account they will be redirected to a screen where they will be able to view the files that are contained on this platform. On this screen, the user will see a list of their files, and depending on the platform their directories. Each file will have a selection box next to it to mark for importing. There is also a "select all" button for the user to select all of their files. Once the user has selected the files they wish to import into the resiliency platform they can select the "Download" button to initiate the import.

Once the "Download" button is selected the user will be redirected to the last screen of the application where they will see a list of the files imported for confirmation.

Both of the screens mentioned in this section also contain a "Start Over" button that will allow the user to return to the main screen where they select a platform. This makes it easier to quickly visit all of their platforms to import their files.

3 Administrators

The administrators or operations team for the application has a few things that need consideration but is otherwise straight forward.

3.1 Starting and Stopping the Application

The number of ways to serve a Python-Django application is almost infinite so please refer to the Django documentation and the documentation of your web server on how to deploy the application.

Whichever deployment method is choses, there are several settings that need to be changed in *project_root/cloud_backup/settings.py*.
The two settings that must be changed is the **DEBUG** value must be set to **False**. This will limit the information sent to potential attackers.
The second setting that must be absolutely changed is the **SECRET_KEY**. The default key shipped is publicly available and should not be used. Please generate a random 50 character string to replace it with.

Other settings in that file that could be changed is the **DATABASES** driver section for your database (if you have one, if not the default SQLite is adequate). As well as the **LAN-**

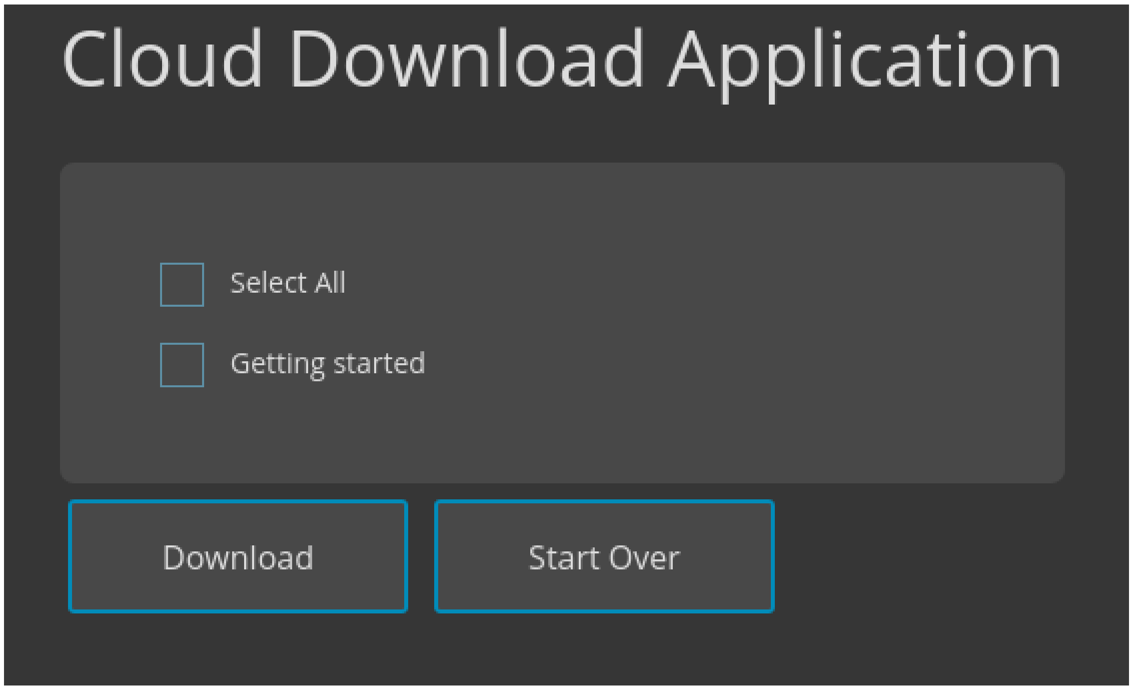


Figure 2: The file selection screen.

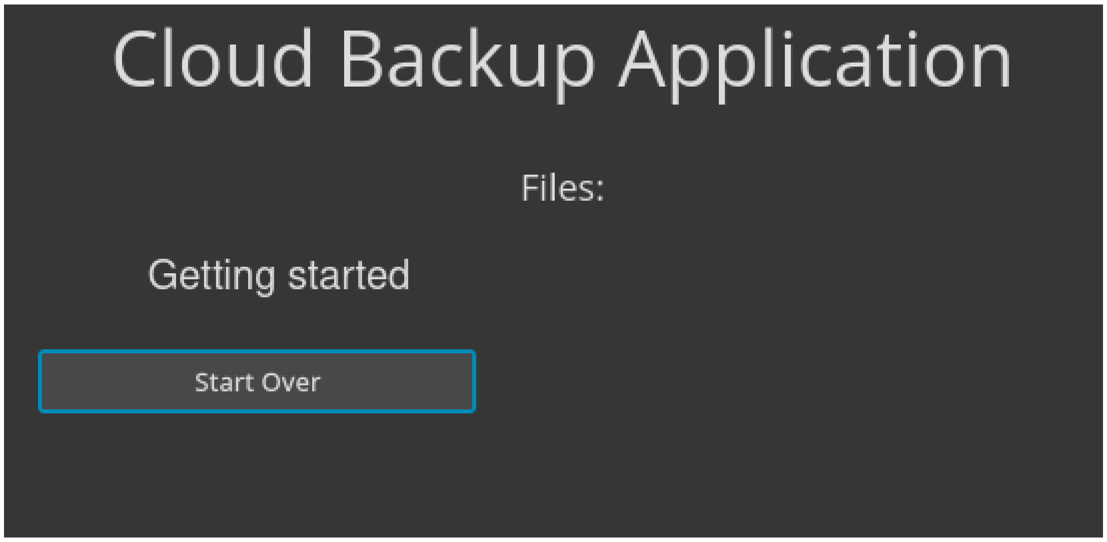


Figure 3: The files downloaded screen.

GUAGE_CODE and **TIME_ZONE** settings.

For the built-in development server one can simply issue the command:

```
$ python manage.py runserver 0:80
```

Note that this is not recommended in a production environment.

3.2 Special Requirements

Some special requirements for deploying the application is some platforms will need this application submitted for verification to become a trusted application. The application will work if it is unverified but users will get several warning screens to click through that may turn them away from the application.

The second special requirement is that the application should be served over HTTPS with a trusted TLS certificate. This will result in less warnings by the platform. Obtaining a certificate and using HTTPS is beyond the scope of this document. Administrators should refer to their web server documentation if they need more information.

4 Developers

Software developers that would like to add additional cloud platforms to the application will need to update several files in addition to the platform module. The modular design and implementation of the application makes this fairly straightforward and quick.

Once a cloud platform has been implemented following the template from the `design document` that module should be added into the *project_root/cloud_download/platforms* directory. Then inside that directory is a *__init__.py* file where the module will need to be imported.

The next step is adding the platform to the main screen. In the *project_root/cloud_download/views.py* the *Index* class will need to be modified. There is list variable called **platforms** that will need the name of the platform added (to be seen by the user) and also the **post** method will need to have the platform added with its appropriate redirect path for authentication.

How to integrate authentication will vary depending on the cloud platform. The *project_root/cloud_download/platforms/dr* contains a good example of using an **Oauth2** derived authentication. The *project_root/cloud_download/platforms/aws.py* contains a good example of using a custom page for prompting the user for their credentials. In addition, the *project_root/cloud_download/urls.py* will need to be modified with the correct URL redirect paths. The files are well documented for how to add additional platforms.

The final step in adding a cloud platform is modifying the **Files** class in the *project_root/cloud_download/views.py* file. There are already several examples of cloud platforms and adding additional ones should be straight forward.

The design of the application was focused on making it as trivial as possible to add additional platforms but that is not a guarantee. Special variables may need to be added, extra pages, views or other unforeseen modifications. But special care was taken to make it as easy as possible. The html templates for example, should not need to be modified to add additional platforms.