***Filecloud***

├── *main.py*

├── ***Handlers***

│ ├── *downloadhandler.py*

│ └── *uploadhandler.py*

├── ***Models***

│ ├── *dir.py*

│ ├── *user.py*

│ └── *file.py*

├── ***Operations***

│ ├── *directoryoperations.py*

│ ├── *useroperations.py*

│ └── *fileoperations.py*

├── *app.yaml*

├── ***Templates***

│ ├── *login.html*

│ ├── *error.html*

│ └── *main.html*

├── ***Stylesheet***

│ ├── *content-back.png*

│ └── *stylesheet.css*

├── ***Docs***

│ └── *README.docx*

└── *display.py*

***main.py***

└── ***Class MainHandler***

└── ***Def Get(self):***

Using response object, content type is set to *text/html* as an output. User log in is checked to start with, **if user is already logged in**. It checks if logged in user exists in *User (ndb.model),* in case user doesn’t exist in *User (ndb.model)* it is added into *User (ndb.model). It’s root and current directory is set to (userid+/).* It gets (*directory\_name)* from response object for navigation purposes, checks if it’s not *null* and it helps to navigate through files and folders. Navigation method is done by taking path from current directory object and file name. After getting (*ndb.Key)* it from (*Folder model)* set current user current to *ndb*.*Key*.

Using *(ndb.Folder)* directs entity list, getting lists for following:

1. Directory names

Using *(ndb.Folder)* files entity list, getting lists for following:

1. File names
2. File size, kind, create times

JINJA template main handler render method is called to send response object to display output in *html* and loading static *CSS* file.

**If is not logged in** JINJA template login handler render method is called to send login response object to display output in *html* and loading static *CSS* file. It takes user to *GAE* login interface.

***main.py***

└── ***Class MainHandler***

└── ***Def post(self):***

Using response object, content type is set to *text/html* as an output. Getting button values from response object each operation is performed based on value spitted out from response object.

|  |  |
| --- | --- |
| Values | Responses |
| *Add* | 1. Name Validation using regular expressions python 2. Adding directory linking it with parent directory, setting directory using (Folder *ndb.Model),* getting key from it. Append directs and setting up entities of *(Folder ndb.Model)* 3. Redirect to *self* |
| *Delete* | 1. Getting name, kind from response object. 2. Based on kind call operation accordingly  * given name file is removed from (*files entity*) of *(Folder ndb.Model)*. Remove from *blobstore* and delete *key.* * *Given name directory is removed if it’s not empty don’t delete else delete.* |
| *Up* | 1. Getting current user object with help of key 2. Check if not it is in root directory, set (*user\_current\_directory)* to *(parent\_directory\_key)* and redirect to self. |
| *Home* | 1. Getting current user object with help of key 2. Set (*user\_current\_directory)* to *(ndb.Key(Folder,user.id + /)* and redirect to self. |

***main.py***

└── ***Class MainHandler***

└── ***Var app***

It points mapping of various *def* to *urls*, using python webapp2 class (*WSGIApplication)*

***display.py***

└── ***Var*** ***JINJA\_ENVIRONMENT***

It holds settings for template rendered environment for *html/text.*

***Params***

├── ***loader*** its file system loader in this case it points to*(jinja2.FileSystemLoader)*

├── ***extensions*** added extensions in this case its*(jinja2.ext.autoescape)*

└── ***autoescape*** set to *true*

***display.py***

└── ***Def render\_login(self, url)***

Main objective of *render\_login* is to convey user to login. It points to html template in “*Templates/login.html”* to jinja2 to get it loaded with jinja2 environment and using response object write template with parameter *{url}* to render output.

***display.py***

└── ***Def render\_error(self, url, error)***

Main objective of *render\_error* is to convey user to message in case user’s file is uploaded successfully, user has chosen already uploaded file or has not selected any file at all. It points to html template in *“Templates/error.html”* to jinja2 to get it loaded with jinja2 environment and using response object write template with parameter *{url,errorl}* to render output.

***display.py***

└── ***Def render\_main(self,url,dirs,files,size,create,kind,length,totalsize,totalfiles,totaldirs,current\_path, is\_in\_root, upload\_url)***

Main objective of *render\_main* is to display filecloud structure to user and provide functions, details and navigation through them. It serves as backbone of user application experience. It points to html template in *“Templates/main.html”* to jinja2 to get it loaded with jinja2 environment and using response object write template with parameter *{****self, url,dirs,files,size,create,kind,length,totalsize,totalfiles,totaldirs,current\_path, is\_in\_root, upload\_url****}* to render output.

***Handlers***

└── ***downloadhandler.py***

└── ***Class DownloadHandler(blobstore.handlers.BlobstoreDownloadHandler)***

└── ***Def Get(self):***

From request object file get *file\_name* and create *file\_object* of it. In process, *path* is created for file and using user\_*id* and *created path, file\_id is created. Then, file\_id* is used to get entity from (*model.File).* Entity (*file\_object.bolb*)is handed over to blob\_handler, along with *save as* argument to achieve name integrity of file to be downloaded.

***Handlers***

└── ***uploadhandler.py***

└── ***Class UploadHandler(blobstore.handlers.BlobstoreUploadHandler)***

└── ***Def post(self):***

*(var e*) is used to hold string message, receives when user is trying to upload files. In case where file already exists in directory is avoids file duplication. On other hand, if user have not selected any file it triggers no file selected message. *(Var e)* is to bind string message *with jinja2 template* for designated template in *Templates file.* In this situation*, “error.html”. Get\_uploads is* blobstore upload handler *def* toaccess all uploads sent to handler*.* Using *BlobInfo* file name is obtained, time is stamped on creation and adding process occurs*.*

***Operations***

└── ***fileoperations.py***

└── ***Def get\_files\_in\_current\_obj()***

It returns list of keys all files in current directory using current directory object.

***Operations***

└── ***fileoperations.py***

└── ***Def file\_object()***

In process, *path* is created for file and using *file\_name* and *current directory object, file\_id is* created using *user key id* and created path*. Then, file\_id* is used to get entity from (*model.File) and its object is returned.*

In process, *path* is created for file and using *file\_name* and *current directory object, file\_id is* created using *user key id* and created path*. Then, file\_id* is used to get key from (*model.File.* Then, using key current directory object file list is checked if file does not exist: *file\_id (user key id + path*) is used to get file object from *(model.File).* Attributes like *file name, date and* blob is set according to given arguments and using *blobinfo.*