**Title:** CMS Documentation Report

**Date Started:** 5/3/2019

**Done by:** Bohui and Terris

**Purpose Of Document:**

This document shares the development of the CMS Website. This document will include explanations on development platforms, languages used, required libraries/packages, the CMS Website, and the project folder. It will also cover error handlers and how to control them. The purpose of this document is to hopefully teach what we have shared about the CMS Website in this document.

**Content Of Report:**

1. CMS Introduction
2. Development Platforms
3. Languages Used
4. Frontend Libraries
5. Backend Packages
6. CMS Admin Features
7. CMS General Features
8. CMS Common Features
9. Project Folder
10. Error Handlers
    1. Error 503: DialogFlow API Save Error
    2. Login Flash Messages
    3. Creating/Updating FAQ Flash Messages
    4. Save To WonderBoy Popup Dialogue
    5. Deleting FAQ Popup Dialogue
    6. Changing Password Flash Messages
11. Maintainable Codes
    1. Page Header Navigation
    2. User Guide Instructions
    3. FAQ Rules Topics To Avoid
12. Codes in process.py
13. Codes in util\_funcs.py
14. AJAX Introduction
15. Files With AJAX
16. Overview Diagram Of CMS Website

**CMS Introduction:**

With prior knowledge about the GT WonderBoy, the CMS Website project is dedicated to allow clients to create unique speech and responses for their GT WonderBoy, referred to as FAQs. This allows clients to be able to create custom questions with answer for their own GT WonderBoy. Example: Qn.“Where is the toilet?” will cause GT WonderBoy to say Ans.“At level 1 beside the fountain”.

Each set of FAQ will consist of one intent, up to 10 questions, and one answer. It also allows an additional/optional Youtube video link to be part of the answer. Currently, the Youtube link is just an addon, hence, users must still provide a basic text answer for their FAQ.

**Development Platforms:**

PyCharm (IDE), BitBucket (Browser), DialogFlow (Browser), MongoDB Compass (Database)

PyCharm is the IDE used to develop the CMS Website. Whenever a build is made to the CMS Website, it is required to commit the changes with an appropriate description about the changes made.

Once committed, it is recommended to push the build into BitBucket to save the build online. BitBucket is a web-based version control repository to track and save progress made to a project, in short, an online shared repository.

When handling with any FAQ manipulation in the CMS Website, please use DialogFlow and MongoDB Compass to check for successful changes made. DialogFlow will track the intents saved into GT WonderBoy, while MongoDB will track all database entries such as FAQs and accounts.

**Languages Used:**

**Backend:** Python

**Frontend:** HTML, CSS, JavaScript

Python is the default backend language used in PyCharm. It is used to render, link, and process each webpage with the use ot the Flask Framework (explained later on). It is also used to create the logic behind manipulating data with MongoDB and DialogFlow.

HTML and CSS will be used in creating the responsive UI of the webpages, while JavaScript is used for implementing user interactivity such as dropdown lists and interactive tables.

The HTML codes are structured with the Flask Framework inplace. With this, base.html is the master page where all other HTML files will inherit codes from. Such as the header, footer, and connection to external files.

CSS and JavaScript for the CMS Website are stored separately into external stylesheets and JavaScripts files, respectively. While creating classes and functions, avoid using simple naming conventions to prevent same names. In the external CSS, to make a webpage responsive, use the @media query. Reference: <https://www.w3schools.com/css/css_rwd_mediaqueries.asp>.

**Frontend Libraries:** JQuery, AJAX (From Jquery)

JQuery is used in conjunction with AJAX. With AJAX, it allows web pages to reload itself without refreshing the page. This is required for filter buttons, popup dialogue boxes, and loading animiations. As AJAX is very extensive in implementation, it is also used to call the Python backend codes to link to other pages as well.

When using AJAX, it is required to create a duplicate .html file of the target page and edit the JQuery codes to enable AJAX. It is also required to create a separate function of the targeted page in the backend Python codes for AJAX to call to.

**Backend Packages:** DialogFlow Python API, JSON, BSON, Flask Framework

DialogFlow Python API is required to connect the CMS Website to DialogFlow to GT WonderBoy. Functions used from the API only includes adding/updating and deleting intents.

JSON is used to make human-readable text when coding data to be transferred. Commonly used in conjunction with DialogFlow API. Whereas BSON is used mostly of its ObjectId function in conjunction with MongoDB.

Flask Framework is used to allow Python codes to be used in a web framework. This allows us to use template inheritance in HTML and create functions for each web page in Python backend.

**CMS Admin Features:**

The CMS Website is currently equipped with unique features for admin and general users. An admin account is equipped with the tools to create and manage accounts in the CMS Website. Admins are able to create and reset the passwords of admins/general accounts. Admins can also modify the particulars of general accounts; client name/alias, user Id, and bot Id.

**CMS General Features:**

General accounts are equipped with the tools to create and manage their own FAQs. An FAQ is a set of data consisting of an intent, up to 10 questions, and an answer. It also allows for an additional/optional Youtube video link as an answer. When creating a FAQ, it will be sent to draft with action type “Add”. Once drafted, they have to select and send the FAQ to “Save to WonderBoy”. Now they have finished creating the FAQ into their GT WonderBoy.

Whenever clicking on “Save to WonderBoy”, except for action type “Delete”, the FAQ will remain the same action type as it serves as the last known action. The type will change to “Draft” and the changes will be saved into GT WonderBoy.

Whenever they edit or delete any FAQ, it will not take immediate effect. Instead, the FAQ will be sent to drafted with the action type “Edit” or “Delete”, respectively, where they must confirm it by clicking on “Save to WonderBoy” again. If it is already type “Drafted”, it will only change the action type if not already action type “Edit”.

When editing or deleting any FAQ, it will change the action type to “Edit” or delete the FAQ permanently. While doing either, it also make the corresponding changes to GT WonderBoy if applicable.

To understand the flow better, it is recommended to run the website.

Here is an illustration of all possible routes based on the FAQ type:

Type(Draft) = ActionType(Add/Edit/Delete)

Type(Saved) = ActionType(Add/Edit)

Deleted = GONE

Draft(Add) > \*Clicks Save to WB\* → Saved(Add)

Draft(Edit) > \*Clicks Save to WB\* → Saved(Edit)

Draft(Delete) > \*Clicks Save to WB\* → GONE

Anything > \*Clicks Edit Icon\* → Draft(Edit)

Saved(Add/Edit) > \*Clicks Delete Icon\* → Draft(Delete)

Draft(Add/Edit) > \*Clicks Delete Icon\* → No Change

Draft(Delete) > \*Clicks Delete Icon\* → GONE

**CMS Common Features:**

For both accounts, they are able to access the webpages for “About Us”, “Guide”, “GT Wonder Boy”, and “FAQ Rules”.

The “About Us” shares insights about GT Robots and a short description about the company, not the CMS.

The “Guide” shares a short introduction about the CMS and instructions on how to use the CMS Website. It also covers the “Youtube Video Information”, “additional information”, “CMS Tips And Tricks”, and “Example Of Good Intents”.

The “GT Wonder Boy” will display a long explanation on Natural Language Processing (NLP), and the GT WonderBoy manual for users to read more about GT WonderBoy. The NLP explanation is mentioned to give a high impression on our GT WonderBoy, and how it is used. The GT WonderBoy manual is provided to allow users to read our manual for more information.

The “FAQ Rules” provides users with a list of pre-built dialogues in GT WonderBoy, with intents, keywords/phrases, and examples. The purpose of this webpage is to teach users the rules for creating their own FAQ. The rule is to avoid using any of the keywords in the same context or sentencing of the original intent.

In both accounts, they are also equipped with the feature to reset passwords. However, this feature functions differently for both account types. General accounts can only change their own password, while admin accounts can reset the passwords of any account.

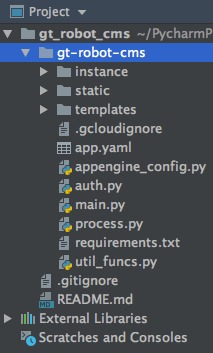
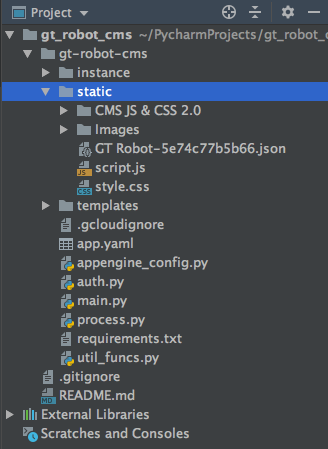
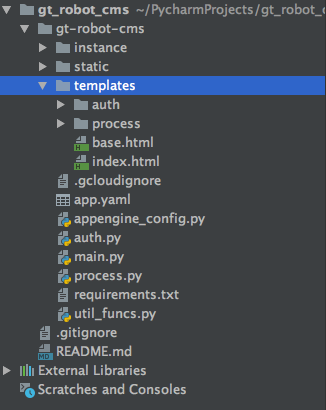
**CMS Project Folder:**

In the CMS Website, we have made comments within the codes for further clarifications. The project folder contains many subfolders and python files. For set-up instructions, read README.md.

Within the static folder contains the external stylesheet, external JavaScript, DialogFlow json file to use the API (Maybe removed in the future), and all images used.

Within the templates folder contains all HTML files, categorised to either auth or process. The main master page is labelled base.html. Ignore index.html (If it is still there). The auth folder will contain HTML web pages that handles with sensitive data. Example: login, changePass, modifyAcc, register, updUser. The process folder will contain all other HTML web pages.

Python files to be used are auth.py, process.py, and util\_funcs.py. Both auth.py and process.py contains methods to control each web page. The util\_funcs.py contains common reusable methods that can be reused, it also contain codes that handles the database layer. Note: To use the methods in util\_funcs.py, remember to import the method to the corresponding .py file at the very top of the file.

**  **

Legend:

- File layer 1

- File layer 2

- File layer 3

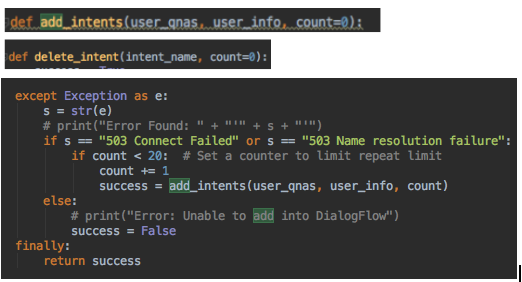
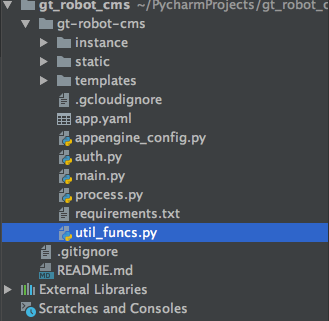
**Error Handlers:**

In the CMS Website contains many codes that handles with connectivities to external APIs and the database. Mainly towards DialogFlow and MongoDB. To resolve these issues, we have implemented bug fixes with codes that can deal with these errors, however, it can cause long loading time to users as well. Hence, we implemented AJAX for a loading cursor upon loading, and popup dialogues and flash messages to notify users of any errors or action successful.

In this section of the document, we will be sharing these error handlers, and how to control them. Specifically, adjusting their settings from within the codes. We will also be showing sample UIs for the flash messages and popup dialogues.

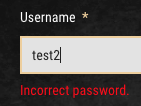
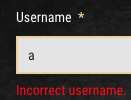
**Error 503: DialogFlow API Save Error:**

When saving a FAQ into DialogFlow, there will be an error named “Error 503 - Connect Failed”. This error has a high occurrence rate and appears whenever using add/update or delete intents from the DialogFlow API. To counteract this error case, we have implemented a measure into the codes to repeat the add/update or delete method until successful. To prevent the case where it can fail indefinitely, we also placed a countermeasure by limiting the repetition of these codes to 20 repeats. To observe the live codes, refer to file “util\_functions.py”, method name “add\_intents” or “delete\_intents”.



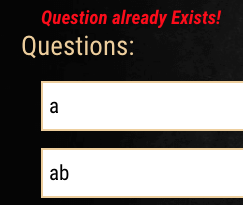
**Login Flash Messages:**

When failure to login into an account, a flash message will appear with the two possible messages as shown below.

****

**Creating/Updating FAQ Flash Messages:**

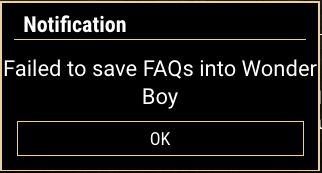
In either create or update FAQ page of general accounts, when the user clicks “Create/Update”, the backend codes will validate and verify the data inputs. Here are samples of the flash messages. The method of displaying the error message may differ over time. It will also check the Youtube link is valid.

****

**Save To WonderBoy Popup Dialogue:**

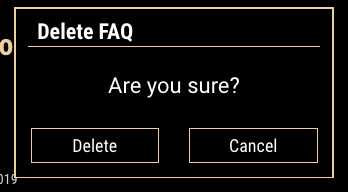
In the main page of any general user account, when the user clicks on “Save to Wonder Boy”,

a popup dialogue will appear and will display the corresponding message based on the results of the codes when saving a FAQ into DialogFlow. The following are the possible display messages.

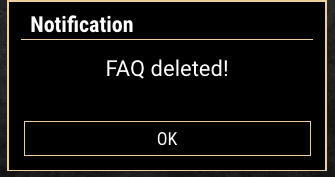
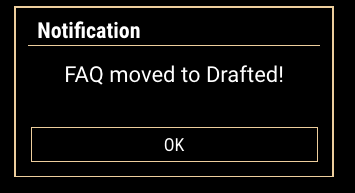


**Deleting FAQ Popup Dialogue:**

Whenever deleting a FAQ, a popup message will to prompt the user for a confirmation.

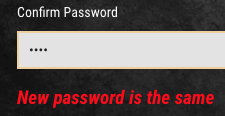
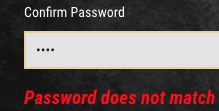


Based on the type of FAQ, a followup message will be displayed. If the FAQ was previously “Saved” into WonderBoy, it will be sent to draft. Whereas if the FAQ was previously “Drafted”, it will be immediately deleted. As shown in the example below.



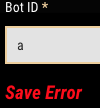
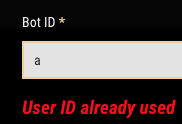
**Changing Password Flash Messages:**

When general users change their password, it will validate and verify the input. The following are the two flash messages put into action.



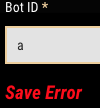
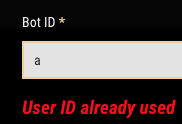
**Registering Account Flash Messages:**

In the admin account, when registering, the system will verify the inputs. Specifically the username and user ID. These are the following flash messages displayed.



**Modifying Account Flash Messages:**

Similarly when modifying an account, the system will verify the inputs. As of current time, only general accounts can be modified. These are the following flash messages displayed.

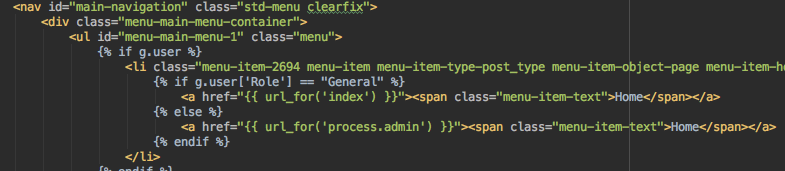
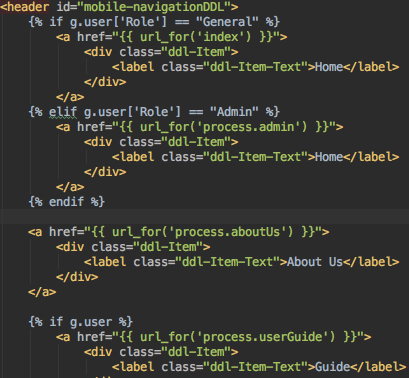


**Maintainable Codes:**

The maintainable codes section will explain any codes that is maintainable and requires a brief explanation in order to maintain it. Codes explained will include front-end and back-end codes depending on the target webpage. We will be explaining this codes in parts based on the webpage and function.

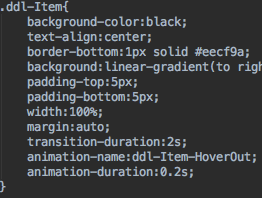
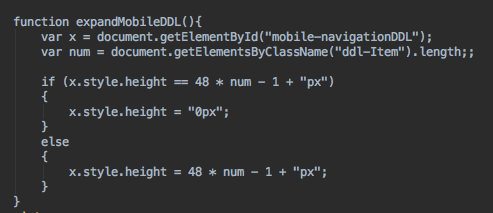
**Page Header Navigation:**

The codes for the page header is located in the base.html. There are two different headers, each for desktop and mobile. The tag id for each are “header-wrap” and “mobile-header”. To add/edit/remove a navigation item, edit along both the “main-navigation” and “mobile-navigationDDL” HTML tag id.



The header navigation bar is currently designed to be flexible and expand accordingly based on the number of items inside the mobile navigation bar. Nevertheless, the codes to edit the Javascript function is named “expandMobileDDL()”. From there you can edit the height per row.

To edit the appearance, you will have to search for the CSS class “ddl-Item”.

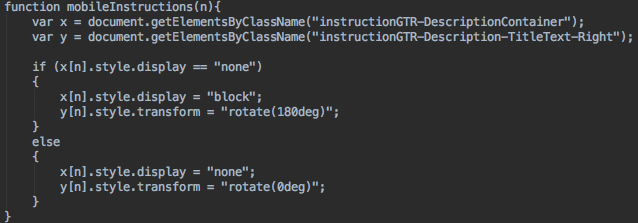


**User Guide Instructions:**

The user guide instructions is designed to explain the process of the CMS Website to the users. To modify the contents of the page, change the HTML contents from the userGuide.html.

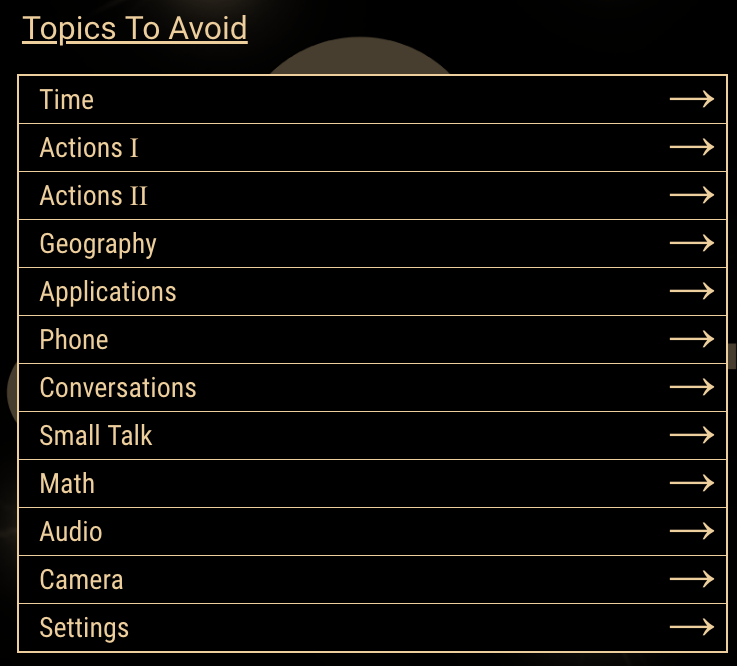


In mobile perspective, the table will switch into an interactive dropdownlist. To modify the height of the dropdown space, look for the function “mobileInstructions(n)”.



**FAQ Rules Topics To Avoid:**

The topics to avoid is designed to tell users the already existing topics built into the GT WonderBoy to avoid. The topics are actually intents that have been streamlined to make it easier to understand and presentable. These topics are then categorised into functions. The table comes in both desktop and mobile perspective.



To maintain them requires the editing of the HTML, CSS and Javascript. To edit the description of a topic, look for the function “rulesGTRSideContent(topic)” inside newScript.js, then look for the topic name to modify the description displayed for both mobile and desktop. Testing changes on the browser requires a hard refresh sometimes. Hard refresh can be accomplished by holding CMD+Shift+R (For Mac), Ctrl+Shift+R (For desktop).



When trying to add or remove a row in topic or function, it is required to also change the CSS height of the tables in desktop and mobile to allow the table to display all function and topic entries properly as both tables have the property “overflow-y: hidden;”. Once done, modify the html and Javascript accordingly to include the new function or topic. It is generally easier to add/remove topics than functions in this webpage. Below is a list of associated functions to the FAQ Rules table.

Mobile Table



PC Table











General (Used in both Tables)



**Codes in process.py**

The process.py contains methods associated with each webpage of the CMS Website. This methods are the backend controllers that renders and handle the displayed data for their corresponding webpages. I will also be mentioning util\_funcs.py codes that are associated for each method.

Note: It does not contain all methods as more confidential data will be placed inside auth.py instead.

def index() :

This method controls index.html, the webpage for general users homepage. Using fetch\_faqs() to retrieve the list of FAQs for that account.

def refreshDraftList():

This method controls indexDraftFAQList.html, a duplicated webpage from index.html. These two duplicated method and html is required for the use of AJAX. It also uses fetch\_faqs().

def saveToWB():

This method controls the “Save To WonderBoy” buttons in index.html, indexDraftFAQList.html, updateMenu.html, and updMenuListSect.html. It uses the save\_qa() method to save changes to DialogFlow to reflect the changes in GT WonderBoy, and updates MongoDB data.

def add():

This method controls the add.html, a webpage for creating intents. This method just displays the webpage.

def addFAQ():

This method controls the add.html, a webpage for creating intents. This method actually controls the “create” button of the webpage. It uses the check\_intent() and check\_qa() to verify the data, then add\_qa() to create the data into MongoDB.

def updateMenu(type):

This method controls the updateMenu.html, a webpage displaying all draft/saved FAQs. It uses fetch\_faqs() to retrieve then renders the page with the retrieved data.

def grabQAList():

This method controls the updMenuListSect.html, a duplicate of updateMenu.html. It uses fetch\_faqs() to retrieve then renders the page with the retrieved data. Duplicated for AJAX reasons.

def update(id):

This method controls the update.html, a webpage displaying the UI for updating the selected intent. It uses fetch\_one\_qa() to retrieve the details of the selected FAQ, then renders it.

def updateFAQ():

This method is the controls for the “Edit” button inside of update.html. It uses check\_intent() and check\_qa() to verify the data in MongoDB, then uses fetch\_qa\_type(), del\_qa(), and add\_qa() to update the FAQ. The updating process is such as we use fetch\_qa\_type() to determine its current type, then delete then add instead of just updating the FAQ.

def delete():

This method controls the “delete icon” inside the updateMenu.html and updMenuListSect.html. It uses the fetch\_qa\_type(), fetch\_one\_qa(), del\_qa(), and add\_qa(). It uses the fetch\_qa\_type() to retrieve the FAQ type, then will do the following changes based on the type. If it is “Draft”, it will just delete the FAQ. While a “Saved” FAQ will first delete the FAQ, then add the FAQ back but with a FAQ type of “Draft”

def preview(action):

This method controls the preview.html, a webpage that only renders the preview after creating or editing a FAQ.

def aboutUs():

This method controls the aboutUs.html, a webpage that only renders the contents of aboutUS.html.

def userGuide():

This method controls the userGuide.html, a webpage that only renders the contents of userGuide.html. To change the information, you will have to change the hardcode HTML contents.

def gtwbManual():

This method controls the gtwbManual.html, a webpage that only renders the contents of gtwbManual.html. To change the information, you will have to change the hardcode HTML contents.

def rulesFAQ():

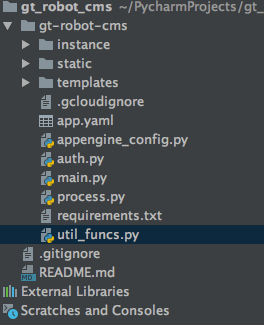
This method controls the rulesFAQ.html, a webpage that only renders the contents of rulesFAQ.html. To change the information, you will have to change the hardcode HTML contents. To change the table of rules, you must edit the JavaScript codes of it inside the external javascript, newScript.js.

def admin():

This method controls admin.html, the webpage for admin users homepage. It only renders the webpage.

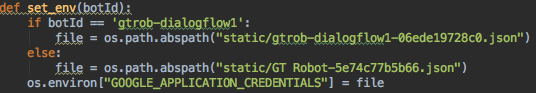
**Codes in util\_funcs.py**

In util\_funcs.py contains all of the common methods used in auth.py and process.py. This section will be explaining each method.



def set\_env(botId) :

Sets the connection between the CMS Website and DialogFlow. This method sets the os.environ[“GOOGLE\_APPLICATION\_CREDENTIALS”] in the project to the file path of the .json file provided by DialogFlow.



def new\_user(role, username, password, client, client\_name=None, client\_alias=None, user\_id=None, bot\_id=None, createdAt=time.asctime()) :

Used to create new users in the CMS Website by inserting the data into MongoDB “gtrobtest” collection. It is used to create both admin and general accounts.

def update\_user(original\_user, client, nrole=None, nclient\_name=None, nclient\_alias=None, nuser\_id=None, nbot\_id=None, mdb=None) :

Used to update information about the general users. It is used by admin accounts. Currently, this method only modifies general user accounts.

def clientname\_exist(client\_name, client) :

Compares the input for client name with MongoDB data, to check if the client name is already used.

def user\_id\_exist(user\_id, client) :

Compares the input for user Id with MongoDB data, to check if the user Id is already used.

def bot\_id\_exist(bot\_id, client) :

Compares the input for bot Id with MongoDB data, to check if the bot Id is already used.

def forget\_password():

Currently not used.

def change\_password(update, value1, key1, value2, key2, client) :

Allows both admin and general accounts to change their password. The variables key1 and value1 is the current password, while variables key2 and variable2 is the new password.

def fetch\_user(value, key, client) :

Used to fetch the user information. It is currently used for logging into the CMS Website, and is used in admin account to retrieve information for a specific account at the user list.

def fetch\_accounts(value, key, client):

Admin accounts use this function to be able to display the list of users in the user list.

def fetch\_one\_qa(value1, key1, value2, key2, qna\_db, qnaInt\_db):

Retrieves a specific FAQ from MongoDB. The variables key1 and value1 are the intent, while variables key2 and value2 are the username. These two sets of variables are the select queries.

def fetch\_faqs(value1, key1, value2, key2, qna\_db, qnaInt\_db):

This method is used to display all FAQs for belonging to that general account. Variables key1 and value1 are the type, while key2 and value2 are the username. These two sets of variables are the select queries.

def fetch\_qa\_type(value1, key1, value2, key2, qnaInt\_db):

Used to retrieve the FAQ type from a specific FAQ. Variables key1 and value1 are the intent, while key2 and value2 are the username. Both sets are select queries. Types include “Saved” and “Draft”.

def update\_qa\_action(value1, key1, value2, key2, value3, key3, qnaInt\_db):

Used to change the last known action of the FAQ between “Add”, “Edit”, “Delete”. Variables key1 and value1 are action, key2 and value2 are intent, and key3 and value3 are user\_id.

def add\_qa(user\_id, username, intent, questions, answer, qna\_db, qnaInt\_db, type, prevAka=None, videoId=None, action=None, check=True, faq\_id=None) :

Used to create FAQs by general users. The FAQ created consists of 1 intent, up to 10 questions, and 1 answer. With an optional Youtube video. Each intent is linked to a general account to identify which account it belongs to.

def check\_qa(user\_id, faq\_id, question, qna\_db, qnaInt\_db) :

Used to check if any of the 1-10 questions in that intent is already used. This method is used by giving 1 question into the parameter, then compared to every other existing question in MongoDB to check if any are the same. To use this method for multiple questions in an intent, place this method into a for loop upon calling it.

def check\_intent(user\_id, faq\_id, intent, qnaInt\_db) :

Used to check if the intent name of the FAQ already exists inside MongoDB.

def del\_qa(value, user\_id, qaType, qna\_db, qnaInt\_db) :

Used to delete a FAQ from MongoDB.

def save\_qa(intents,username, qna\_db, qnaInt\_db, uinfo) :

Used to save multiple FAQs into Wonder Boy. This method is called upon clicking on the “Save To WonderBoy” button. It has multiple purposes, to “Add” a FAQ, “Edit” a FAQ, or “Delete” a FAQ in the MongoDB and DialogFlow collection This method is the main feature of the CMS Website to save the changes to FAQs.

def user\_exists(username, client) :

Used to check if the username is taken when registering an account.

def add\_intents(user\_qnas, user\_info, count=0) :

Used to add/update intents in DialogFlow. With the user\_qnas array, data of each qna is formatted into a JSON formatted dictionary. Than this JSON formatted dict will then be used to send over to DialogFlow via their API methods to make the changes remotely. Connection errors may occur while using their API methods.

def delete\_intent(intent\_names, clientAlias, bot\_id, count=0) :

Used to delete intents into DialogFlow. It is structured differently from add\_intents. It instead takes intent names, then calls the API methods to delete any intents with the same intent names in the parameter.

def fetch\_intents\_dialogflow(user,client,intent\_names=None):

Used to retrieve any intents in DialogFlow via the DialogFlow API method. It retrieves the intents by being provided with the parameters of the list of names of intents.

def grp\_questions(questions) (INCOMPLETE):

Used to group up an FAQ’s questions together into a list object. This list object will then be inserted into DialogFlow as training phrases for the Dialogflow intent that is being created at that moment.

def crt\_payload(qna) (INCOMPLETE):

Used to create a Dialogflow custom intent payload. This is used when inserting a video into a FAQ. The payload will include a YouTube video ID, an intent, the video will be related to, and the response to the intent.

**AJAX Introduction (NOT-UP-TO-DATE):**

AJAX is a set of web development techniques used in creating client side asynchronous web applications. With Ajax, web applications can send and retrieve data from a server asynchronously without interfering with the display and behavior of the existing page. In short, it allows the refreshing of the UI without actually refreshing the webpage.

The CMS is also using AJAX to exchange data between front-end and back-end layers. For example, change password feature passes data to back end for processing. Another case is Create and Update FAQs, whereby an FAQs data along with other miscellaneous data is being passed to the back-end.

To use AJAX and maintain it in this project, you must first be aware that the AJAX codes located in “gt-robot-cms/static/CMS JS & CSS 2.0/ajaxScript.js”, and is written in JavaScripts jQuery library.

When incorporating AJAX to a single webpage, it is important to link the desired button, on the html page, to the jQuery function that will trigger the AJAX code. Next, ensure that there is a second copy of the HTML, and a python method (Inside either auth.py or process.py) to call the second HTML copy.

To maintain the each AJAX code, you must maintain the ajaxScript.js and necessary html files such as indexDraftFAQList and updMenuListSect.

**HOW DOES THE REFRESHING WORK?:**

We will be seeing this in the scenario of changing between Draft FAQ list and Save FAQ list.

1. User clicks a button that triggers an jQuery event.
   1. The button contains a value(Saved) and this value is taken by jQuery.
2. The jQuery event then retrieves the value that is in the button and stores it.
3. jQuery event triggers an AJAX request, which uses the HTTP POST method.
4. The AJAX request will take the data it has been given and send it to the specified destination.
   1. In our case, the specified destinations are python methods. We are using AJAX as a gateway to call upon these python methods.
5. The specified destination will process the data received, from the AJAX request, and return an output.
   1. In this scenario, the python method triggered returns a rendered web page and the list of Saved FAQs.
6. AJAX request receives the output and processes it to produce the expected output the jQuery event is intended to do.
   1. In this scenario, the list of questions on the web page will be replaced with a new list of questions, that are from the AJAX request.
   2. Please note that the displaying of question data on the list is done by Jinja.

Another item to highlight is that AJAX can be used to get information from other websites. It will work so long as the specified destination, given to the AJAX request, is a point of contact or link to request information from a server.

For example, the YouTube video feature’s verification uses AJAX request to gain information from YouTube’s server. The AJAX request goes through a google API link that requests for information from YouTube’s server. The AJAX request uses the data specified in its data payload to retrieve the information from YouTube’s server.

**Files With AJAX:**

As of writing this documentation:

* updUser / updUserList
* index / indexDraftFAQList
* updateMenu / updMenuListSect

The above are the webpage/HTML files that have AJAX, organised by “original / copy”. Most popup/dialogue message boxes and loading animations are done with AJAX.

The popup/dialogue message can be found in the popUpErrMsg.html while the loading animation spinner can be found in the base.html with the class name “loaderScrn”. To edit them, be sure to also edit the ajaxScript.js file. (NOTE: There are two types of popups, jQuery and normal html. For jQuery popups, please refer to ajaxScript.js file and look through codes related to dialog boxes.)

**Overview Diagram of CMS Website**