Module **api**

Flask API for BDMParse's Angular frontend to communicate with to perform document functions, login functions, and administrative functions

Functions

```
def add_term()
```

Adds a new term code to the database

Note

The POST request should contain the following information:

```
{
    'termCode': String of the new term code to be added,
    'collegeName': College list to add the term code to
}
```

Aborts with a 403 error code if the user is not an admin. This is used to protect against direct calls to the API from software such as Postman.

Returns

`str`
: The unique database ID of the newly added term code

```
def add_type()
```

Adds a document type to the database

Note

The POST request should contain the following information:

```
{
    'documentType': String of the new document type to be added,
    'collegeName': College to add the document type to
}
```

Aborts with a 403 error code if the user is not an admin. This is used to protect against direct calls to the API from software such as Postman.

Returns

`str`
: The unique database ID of the newly added document type

```
def add_user()
```

Adds a new user based on their VT username to the database

Note

The POST request should contain the following information:

```
{
    'role': 'admin' if to be added with admin priveleges, else 'standard',
    'username': VT username of user to be added,
    'college': College the new user belongs to
}
```

Aborts with a 403 error code if the user is not an admin. This is used to protect against direct calls to the API from software such as Postman.

Returns

`str`
The unique database ID of the newly added user

```
def allowed_file(filename)
```

Checks if the filename provided by the front-end is .pdf file

Parameters

`filename` : `str`
A string containing the filename of the file to be parsed

Returns

`bool`
True if filename is a .pdf filename, false otherwise

```
def change_role()
```

Changes the role level of the user specified in the database

Note

The POST request should contain the following information:

```
{
    'params': {
        'role': New role to set the user to,
        'user': VT username of the user to change
    }
}
```

Aborts with a 403 error code if the user is not an admin. This is used to protect against direct calls to the API from software such as Postman.

Returns

`JSON object`
JSON object with `success: True` and a 200 status code

```
def is_admin()
```

Checks whether current user accessing the API has admin privileges

Returns

`bool`
: True if user has admin privileges, false otherwise

```
def is_logged_in()
```

Checks whether user accessing the API is currently logged in

Returns

`int`
: User's role level with 0 being standard user, 1 being admin user if user is logged in, -1 otherwise to indicate not logged in.

```
def is_valid_user(username)
```

Checks if username is a valid user of the web application

Queries the database to determine if information exists for the username specified

Parameters

`username` : `str`
: String of the username to check for existence in the database

Returns

`bool`
: True if information for the user exists, false otherwise

```
def login()
```

Ensures user is logged in and then redirects to parsing page OR redirects to CAS for login if not already logged in before CAS redirects here to check again

Note

The GET request should containing the following arguments

```
'next'
    The link to redirect to after CAS login
'ticket'
    The ticket provided by CAS login
```

Returns

`str`
: CAS login URL if the user is not logged in yet, username if the user is logged in, empty string if the user is invalid

```
def logout()
```

Forms call to CAS to logout and then performs call and will lead to logout_callback

Returns

`str`
: The logout url to send the user to for logging out of CAS

```
def logout_callback()
```

```
def parse()
```

Parses an uploaded bulk PDF file into separate PDFs based on PID

Note

The POST request should contain multipart form data consisting of a file with key `fileKey` and arguments for `documentType` and `termCode`

Returns

`JSON object`
: Upon success the JSON object will be formatted as follows:

```
{
    'total': The number of parsed PDFs,
    'num_success': The number of PDFs successfully uploaded to
                   Banner,
    'num_error': The number of erroneous PIDs according to Banner,
    'successes': List of successful PIDs,
    'errors': List of erroneous PIDs,
    'file_path': String of file path containing erroneous pdfs for
                 later download
}
```

`str`
: If the user is not logged in, a 403 code will be sent. If the provided document type, term code, or file type are invalid a 500 code will be sent with an appropriate message

```
def remove_term()
```

Deletes a term code from the database

Note

The POST request should contain the following information:

```
{
    'params': {
        'id': Unique database ID of the term code,
    }
}
```

Aborts with a 403 error code if the user is not an admin. This is used to protect against direct calls to the API from software such as Postman.

Returns

`JSON object`
: JSON object indicating successful deletion with a 200 status code

```
def remove_type()
```

Deletes a document type from the database

Note

The POST request should contain the following information:

```
{
    'params': {
        'id': Unique database ID of the document type,
    }
}
```

Aborts with a 403 error code if the user is not an admin. This is used to protect against direct calls to the API from software such as Postman.

Returns

`JSON object`
: JSON object indicating successful deletion with a 200 status code

```
def remove_user()
```

Deletes a user from the database

Note

The POST request should contain the following information:

```
{
    'params': {
        'id': Unique database ID of the user,
    }
}
```

Aborts with a 403 error code if the user is not an admin. This is used to protect against direct calls to the API from software such as Postman.

Returns

`JSON object`
: JSON object indicating successful deletion with a 200 status code

```
def user_json()
```

Builds a JSON of the requested user information

Note

The GET request should contain the username in the query parameters as `username`

Returns

`JSON object`

A JSON object with the following format:

```
{
    'username': username,
    'id': id according to the database,
    'role': Either admin or standard depending on the users database
            level,
    'college': College the user belongs to,
    'documentTypes': List of document types for that specific
                     college,
    'userList': List of users belonging to that college for admin
                functions,
    'termCodes': List of term codes for that specific college
}
```

```
def zip_error_files()
```

Zips erroneous PID PDFs into a zipfile to be downloaded by the front-end

Note

The POST requests should contain parameters passed along in the format:

```
{
    'params': {
        'file_path': String of directory containing erroneous files
    }
}
```

Returns

`zipfile`

A zipfile containing the parsed erroneous PDFs

Index

Functions

allowed_file
change_role
is_admin
is_logged_in
is_valid_user
login
logout
logout_callback
parse
remove_term
remove_type
remove_user
user_json
zip_error_files

Module **banner**

Banner communication functions for uploading files to VT's document management system

Functions

```
def bulk_upload(folder_name, doc_type, term_code)
```

Uploads a bulk amount of PDFs to VT's Banner system

Note

It is expected that each file in the directory to be uploaded be named according to the student ID they are associated with. i.e. 905000001.pdf

Parameters

folder_name : `str`
String of the directory containing the multiple PDFs

doc_type : `str`
Document type to be associated with the batch of PDFs

term_code : `str`
Term code to be associated with the batch of PDFs

Returns

`tuple (list, list, str)`
A tuple containing of list of successfully uploaded ID numbers, a list of erroneous ID numbers, and string containing any error information if there were errors with term codes or document types

```
def fields_to_json(id, doctype, termcode)
```

Converts fields to a JSON object that complies with Banner's field specs

Parameters

id : `str`
Student ID number

doctype : `str`
Document type for uploaded file

termcode : `str`
Term code associated with uploaded file

Returns

`str`
String containing a JSON formatted according to Banner's field specifications

```
def test_connection()
```

Test connection with Banner system

Useful for ensuring provided links and authorization credentials are valid before attempting to upload anything

```
def upload_file(file_name, id, doc_type, term_code)
```

Uploads file to VT's Banner system

Parameters

`file_name` : `str`
Relative or absolute file path of the file to be uploaded

`id` : `str`
Student ID to be associated with the file

`doc_type` : `str`
Document type to be associated with the file

`term_code` : `str`
Term code to be associated with the file

Returns

`tuple (int, str)`
Either 200 and Success if upload successful, or the Student ID that was erroneous with a message indicating the error

Index

Functions

Module **db**

Functions for calling stored procedures of a MySQL database

Functions

```
def call_change_user_privilege_level(username, privilege_level)
```

Calls changeUserPrivelegeLevel stored procedure

Parameters

username : str
Username of user who's privelege level to change

privilege_level : int
int of new privilege level for the user 0 = standard, 1 = admin

```
def call_delete_college(cid)
```

Calls deleteCollege stored procedure

Parameters

cid : int
Integer of unique ID corresponding to college to be deleted

```
def call_delete_document_type(did)
```

Calls deleteDocumentType stored procedure

Parameters

did : int
Integer of unique ID corresponding to document type to be deleted

```
def call_delete_term_code(tid)
```

Calls deleteTermCode stored procedure

Parameters

tid : int
Integer of unique ID corresponding to term code to be deleted

```
def call_delete_user(uid)
```

Calls deleteUser stored procedure

Parameters

uid : int
Integer of unique ID corresponding to user to be deleted

```
def call_doc_types_by_college(college)
```

Calls getDocumentTypesByCollege stored procedure

Parameters

college : str
String of college to get document types from

Returns

list
List of tuples with each tuple containing (unique DB ID as int, document type as string)

```
def call_get_all_colleges()
```

Calls getAllColleges stored procedure

Returns

list
List of tuples with each tuple containing (College name as string)

```
def call_get_all_doc_types()
```

Calls getUsersByCollege stored procedure

Returns

list
List of tuples with each tuple containing (unique DB ID as int, document type as string)

```
def call_get_all_term_codes()
```

Calls getAllTermCodes stored procedure

Returns

list
List of tuples with each tuple containing (Unique DB ID as int, term code as string)

```
def call_get_all_users()
```

Calls getAllUsers stored procedure

Returns

`list`
: List of tuples with each tuple containing (unique DB ID as int, username as string, privilegeLevel as int)

```
def call_get_term_codes_by_college(college)
```

Calls getTermCodesByCollege stored procedure

Parameters

`college` : `str`
: String of college to get term codes from

Returns

`list`
: List of tuples with each tuple containing (Unique DB ID as int, term code as string)

```
def call_get_users_by_college(college)
```

Calls getUsersByCollege stored procedure

Parameters

`college` : `str`
: String of college to retrieve a list of users from

Returns

`list`
: List of tuples with each tuple containing (unique DB ID as int, username as string, privilegeLevel as int)

```
def call_insert_college(college)
```

Calls insertCollege stored procedure

Parameters

`college` : `str`
: String of the new college to be added, i.e. 'College of Engineering'

```
def call_insert_document_type(doc_type, college)
```

Calls insertDocumentType stored procedure

Parameters

`doc_type` : `str`
: String of new document type to be added, i.e. 'N-NEW DOC TYPE'

`college` : `str`
: String of college to add document type to

Returns

`int`
Unique database ID of newly added document type

```
def call_insert_term_code(term_code, college)
```

Calls insertTermCode stored procedure

Parameters

`term_code` : `str`
String of term code to be added

`college` : `str`
String of college to add new term code to

Returns

`int`
Unique database ID of newly added term code

```
def call_insert_user(username, privilege_level, college)
```

Calls insertUser stored procedure

Parameters

`username` : `str`
String of the user's username

`privilege_level` : `int`
int of the new user's privilege level, 0 = standard, 1 = admin

`college` : `str`
String of the college the new user belongs to

Returns

`int`
Unique database ID of newly added user

```
def call_validate_user(username)
```

Calls validateUser stored procedure

Parameters

`username` : `str`
String of the username to validate

Returns

`tuple (int, str, str, int)`
Tuple containing the unique database ID of the user, the username, the college name the user belongs to, and

the privelege level to indicate if the user has admin privileges or not. The tuple is empty if the user does not exist in the database.

Index

Functions

Module **doc_map**

Function for mapping non-Banner compliant document types to Banner compliant ones

Functions

```
def map_document_type(input_string)
```

Maps provided input to banner compliant document type if one exists

Parameters

input_string : str
String of document type to be mapped if necessary or able to

Returns

str
String of newly mapped banner compliant document type if possible or the original string to allow for untracked additions of new document types that haven't been added to DOC_TYPE_STANDARD_SET

Index

Functions

Module **doc_splitter**

Function for splitting a bulk PDF into smaller PDFs

Functions

```
def doc_splitter(docName)
```

Parses bulk PDF into individual PDFs named after the PID of the student

Parameters

docName : str
String containing the absolute or relative path to the bulk PDF to be parsed

Returns

str
String containing the temporary output directory of the newly parsed PDFs

Index

Functions