ImageProcessorS18 Documentation

Release v1.0.0

Harvey Shi, Edward Liang, Michelle Wei

Feb 08, 2019

Contents:

CHAPTER 1

actions module

Module for handling requests.

actions.**act_download**(*request*)
Handles download request for images

Parameters **request** – json request from client

Returns b64 image string of processed image

actions.**act_list**(*username*)
Lists the original and processed images for a user

Parameters **username** – client username

Returns json including uuid's of original and processed images

actions.**act_login**(*request*)
Authenticates login with email and password

Parameters **request** – json request from client

Returns json of jwt

actions.**act_process**(*request*)
Processes the original image that has been uploaded

Parameters **request** – request from client

Returns uuid of processed image

actions.**act_upload**(*request*)
Uploads original user image

Parameters **request** – request from client

Returns uuid of uploaded image

CHAPTER 2

database module

Module for interacting with database.

class database.**User**(**args*, ***kwargs*)

Bases: pymodm.base.models.MongoModel

exception DoesNotExist

Bases: pymodm.errors.DoesNotExist

exception MultipleObjectsReturned

Bases: pymodm.errors.MultipleObjectsReturned

objects

The default manager used for MongoModel instances.

This implementation of BaseManager uses QuerySet as its QuerySet class.

This Manager class (accessed via the objects attribute on a MongoModel) is used by default for all MongoModel classes, unless another Manager instance is supplied as an attribute within the MongoModel definition.

Managers have two primary functions:

1. Construct QuerySet instances for use when querying or working with MongoModel instances in bulk.
2. Define collection-level functionality that can be reused across different MongoModel types.

If you created a custom QuerySet that makes certain queries easier, for example, you will need to create a custom Manager type that returns this queryset using the from_queryset() method:

```
class UserQuerySet(QuerySet):
    def active(self):
        '''Return only active users.'''
        return self.raw({"active": True})

class User(MongoModel):
    active = fields.BooleanField()
```

(continues on next page)

(continued from previous page)

```
    # Add our custom Manager.
    users = Manager.from_queryset(UserQuerySet)
```

In the above example, we added a *users* attribute on *User* so that we can use the *active* method on our new QuerySet type:

```
active_users = User.users.active()
```

If we wanted every method on the QuerySet to examine active users *only*, we can do that by customizing the Manager itself:

```
class UserManager(Manager):
    def get_queryset(self):
        # Override get_queryset, so that every QuerySet created will
        # have this filter applied.
        return super(UserManager, self).get_queryset().raw(
            {"active": True})

class User(MongoModel):
    active = fields.BooleanField()
    users = UserManager()

active_users = User.users.all()
```

original_image
A field that stores unicode strings.

password
A field that stores unicode strings.

processed_image
A field that stores unicode strings.

username
A field that stores email addresses.

database.**add_user**(*username*, *password*)
Creates new user if user does not exist in the mongo database

Parameters

- **username** – user email as string type which serves as user id
- **password** – user password as string type

Returns updates user information in mongo database

database.**delete_image**(*name*)
Deletes image stored in server :param name: name (uuid) of an image stored in the VM server

database.**delete_user**(*username*)
Deletes user from mongo database :param username: user email as string type which serves as user id

database.**get_original_image**(*username*)
Gets the original image uuid for a user :param username: user email as string type which serves as user id :returns: uuid of user's original image as a string

database.**get_processed_image**(*username*)
Gets the processed image uuid for a user :param username: user email as string type which serves as user id :returns: uuid of user's processed image as a string

database.**get_user**(*username*)

Gets user by unique username :param username: user email as string type which serves as user id :returns: user information

database.**login_user**(*username*, *password*)

Returns true if user exists and has the correct password :param username: user email as string type which serves as user id :param password: user password as string type :returns: True if password is correct, False if incorrect

database.**remove_images**(*username*)

Removes all images associated with a user :param username: user email as string type which serves as user id

database.**save_original_image_uuid**(*username*, *uuid*)

Updates existing user by adding the uuid of a user-uploaded image :param username: user email as string type which serves as user id :param uuid: UUID4 of user-uploaded image :returns: adds uuid of user-uploaded image to mongo database

database.**save_processed_image_uuid**(*username*, *uuid*)

Updates existing user by adding the uuid of the processed image :param username: user email as string type which serves as user id :param uuid: UUID4 of processed image :returns: adds uuid of processed image to mongo database

CHAPTER 3

images module

Module for managing image storage on backend.

images.**get_image_as_b64**(*uuid*, *filetype='png'*)
Gets b64 image string by uuid

Parameters

- **uuid** – uuid of image
- **filetype** – file type to output, options are jpeg, png, or gif

Returns b64 string of image

images.**get_image_by_uuid**(*uuid*)
Retrieves uint array of image by its uuid

Parameters **uuid** – UUID of image as string

Returns grayscale image array

images.**save_image**(*img_str*)
Converts image string to binary and saves onto drive

Parameters **img_str** – base64 image string

Returns uuid of image

images.**save_image_from_arr**(*img_arr*)
Converts uint array to png file (intermediary format stored on server)

Parameters **img_arr** – uint array of image

Returns uuid of image

images.**split_img_str**(*img_str*)
Splits image string from frontend into metadata and b64 binary

Parameters **img_str** – raw string from frontend

Returns tuple of image type and binary

CHAPTER 4

segment module

Module to handle segmentation process.

segment.**segment**(*uuid*)

Segments image with input uuid, saves processed image to server and returns its uuid

param uuid uuid of original image

returns uuid of processed image, saves b64 string of image on server

CHAPTER 5

validate module

Module for validating requests.

validate.**val_download**(*input*)
Validate download format for username and image uuid

Parameters **input** – input json data from user request

Returns validation true or false

validate.**val_login**(*input*)
Validate login format for username and password

Parameters **input** – input json data from user request

Returns validation true or false

validate.**val_process**(*input*)
Validate process format for username

Parameters **input** – input json data from user request

Returns validation true or false

validate.**val_upload**(*input*)
Validate upload format for username and b64 image file

Parameters **input** – input json data from user request

Returns validation true or false

validate.**val_wrapper**(*input*, *schema_format*)
Validation endpoints

Parameters

- **input** – input json data from request
- **schema_format** – schema format to use

Returns validation true or false

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index