Dass Code Documentation

1)Backend Server for Social Media Content Management:

This Flask app provides endpoints to manage social media content in a MongoDB database.
Endpoints include:

(i) '/': Default route to indicate the server is running

(ii) '/rej': Endpoint to receive and process rejected content data

(iii) '/edit': Endpoint to receive and process edited content data

(iv) '/view-doc': Endpoint to retrieve viewable documents for approval

(v) '/get-image-url': Endpoint to retrieve URLs and descriptions for upcoming posts

(vi) '/send-edited-response': Endpoint to receive and update edited content data

(vii) '/post': Endpoint to prepare and serve posts for approval

(viii) '/send-approval': Endpoint to receive and update content approval status

Variables Used:

1) **Approved**: This tells whether it is approved for posting or not along with the information on which platform it is going to be posted. Value ranges from 1 to 6 where each number depicts a combination of platform on which it would be posted (platform include Instagram post, Instagram story and Facebook post).
2) **Recommended:** This tells whether the approved post has been posted to the desired platform or not. Value 0 means it is not yet posted, 1 means it is posted and 2 means it is posted on one of the two platforms it was supposed to go.

Endpoints Used:

i. ***/rej:***
Endpoint to receive and process rejected content data. Updates the 'approved' field in the database for rejected content.
JSON Parameters: - url: URL of the rejected content
- approved: Approval status (-1 for rejected)
- date: Date of the action Returns:
- JSON response with success message or error message

ii. **/edit:**
Endpoint to receive and process edited content data from the view schedule edit page.
Updates the 'caption', 'date_to_post', and 'time' fields in the database.
JSON Parameters: - url: URL of the edited content
- caption: Updated caption
- date: Updated date
- time: Updated time Returns:
- JSON response with success message or error message
Returns: - JSON response with success message or error message

iii. **/view-doc:**
Endpoint to retrieve approved documents for viewing or any editing if required.
Retrieves documents from the database based on approval(already approved) and recommended status(not yet posted).
Returns: - JSON response with list of viewable documents

iv. **/get-image-url:**
Endpoint to retrieve URLs and descriptions for upcoming potential posts for client's approval on the website.
Retrieves documents with specific approval(approved = 0) and recommended status(recommended = 0).
Returns: - JSON response with image URLs and descriptions

v. **/send-edited-response:**
Endpoint to receive and update edited content data.
Updates the 'caption' field in the database for edited content.
JSON Parameters: - url: URL of the edited content
- updateData: Updated caption
Returns:
- JSON response with success message or error message

vi. **/post**:
Endpoint to prepare and send the posts for posting on selected platforms. Retrieves documents for approval based on date and time.
Returns: - JSON response with image URL, description, width, height, and platform

vii. **/send-approval:**
Endpoint to receive and update content approval status.
Updates the 'approved' field in the database for approved content.
JSON Parameters: - url: URL of the approved content
- approved: Approval status (1 for approved)
- date: Date of the action

- time: Time of the action

Returns: - JSON response with success message or error message

2) Instagram Posting Script

This script is used to post images and stories on Instagram using the Instagram Private API (instagram-private-api) and Jimp for image manipulation.

Requirements:

- Node.js environment with installed packages
- dotenv: For loading environment variables from a .env file (having the sensitive information of username and password hidden)
- Instagram-private-api: Instagram private API wrapper for posting
- jimp: Image processing library for resizing and manipulating images
- request-promise: HTTP request library for making API calls
- express: Web server framework for handling API requests

Endpoints:

- '/get-info': GET endpoint to the server to retrieve any scheduled post on that day and time for the Instagram posting process

Functions:

- **get_info():** Function to fetch data from Flask API and determine the Instagram post type, url of image to be posted and the caption along with it.
- **postToInsta():** Function to post an image to Instagram with a caption
- **addStoryToInsta():** Function to post a story to Instagram with a caption
- **post_and_add_story():** Function to post both an image and a story to Instagram

3) Facebook Posting Script

This script is used to post images on Facebook using the Facebook SDK (facebook-sdk) and Flask for creating a web server.

Requirements:

Python environment with installed packages:

- **facebook-sdk:** Facebook SDK for Python for interacting with the Facebook Graph API
- **Flask:** Web server framework for handling API requests
- **requests:** HTTP library for making API calls
- **Pillow (PIL):** Python Imaging Library for image manipulation
- **dotenv:** For loading environment variables from a .env file
- **flask-cors:** Flask extension for handling Cross-Origin Resource Sharing (CORS)

Endpoints:

- **'/':** Default endpoint to indicate the web server is running
- **'/post_to_facebook':** POST endpoint to trigger the Facebook image posting process
- Method: POST
- Parameters: None
- Returns: JSON response with message indicating success or failure

Functions:

- post_to_facebook(): Function to post an image on Facebook
- Fetches image data and description from a Flask server

- Downloads the image, resizes it, and adds a logo of the company

- Posts the image to Facebook using the Facebook SDK

MongoDB Data Loader

Introduction:

This Python script is designed to load data into a MongoDB database using PyMongo. It reads data from text files containing product information and inserts it into the MongoDB collection.

Setup:

1. Install MongoDB on your system.
2. Install the PyMongo package using pip:
   pip install pymongo

Functionality:

`create_items_for_product(product_name, url, image_url, description, image_width, image_height, caption, date_to_post, type, time, date_hash)`:

- Creates a dictionary representing an item for a product with the given information.
- Parameters:
- **`product_name`**: Name of the product.
- **`url`**: URL of the product.
- **`image_url`**: URL of the product image.
- **`description`**: Description of the product.
- **`image_width`**: Width of the product image.
- **`image_height`**: Height of the product image.
- **`caption`**: Caption for the product.
- **`date_to_post`**: Date to post the product (not currently used).
- **`type`**: Type of the product (e.g., "product", "craft", "blog").
- **`time`**: Time to post the product (not currently used).
- **`date_hash`**: Hash of the scrape date (not currently used).

3. **`update_item_to_approved(url_link, approved_val)`**:

- Updates the "approved" field of an item in the MongoDB collection.
- Parameters:
- **`url_link`**: URL of the item to update.
- **`approved_val`**: New value for the "approved" field.

`update_item_to_recommended(url_link, recommended_val)`:

- Updates the "recommended" field of an item in the MongoDB collection.
- Parameters:
- **`url_link`**: URL of the item to update.
- **`recommended_val`**: New value for the "recommended" field.

4. **`load_database()`**:

- Loads data from text files into the MongoDB collection.

- Reads product information from separate text files (e.g., product_names.txt, product_links.txt) and creates items for each product.
- Inserts the items into the MongoDB collection.

Usage:

5. Ensure that MongoDB is running on your system.
6. Run the script using Python:

python database.py

Note: run the scraper code before running this

File Structure:

- **`product_names.txt`**: Text file containing names of products.
- **`product_links.txt`**: Text file containing URLs of products.
- **`product_image.txt`**: Text file containing URLs of product images.
- **`product_descriptions.txt`**: Text file containing descriptions of products.
- **`product_captions.txt`**: Text file containing captions for products.
- **`product_dim_w.txt`**: Text file containing widths of product images.
- **`product_dim_h.txt`**: Text file containing heights of product images.

Other files are named similarly for blogs and craft stories.

Web Scraping and Database Uploading Documentation

Introduction: This Python script performs web scraping to extract product, craft, and blog information from the Club Artizen website. It then uploads this data into a MongoDB database.

Dependencies:

- `requests`: Used for making HTTP requests to fetch web pages.
- `BeautifulSoup`: Used for parsing HTML content.
- `pymongo`: Used for interacting with MongoDB.
- `cohere`: Used for generating Instagram captions.
- `datetime`: Used for obtaining the current date.

Functionality:

1) Scarper functionality:

- `scraper_products()`: Scrapes product information from the Club Artizen catalog.
- `scraper_crafts()`: Scrapes craft information from Club Artizen's journal.
- `scraper_blogs()`: Scrapes blog information from Club Artizen's blogs.

2) Data extraction

- Extracts product, craft, and blog names, links, descriptions, image URLs, and image dimensions from the website.
- Utilizes `requests` and `BeautifulSoup` for scraping.

3) Caption Generation

- Utilizes the `cohere` library to generate Instagram captions based on product descriptions.
- Adds hashtags and creates captions for Instagram posts.

4) Database Interaction:

- Connects to a MongoDB database using `pymongo`.
- Inserts the scraped data into the database as documents.

5) Data Structure:

- Each item in the database contains fields such as name, image URL, description, caption, image width, image height, etc.
- Items are categorized based on their type (product, craft, blog).

Usage: It has been hosted and will run automatically at 2:00 a.m IST.