Project Documentation

Overview

This project is a web application that interacts with an ad management system. It includes functionalities for adding credentials, creating order lines, and managing tasks through a queue system persisted to disk. The application uses Flask for the web server, Selenium for browser automation, and cryptography for secure password handling.

The application provides several endpoints to interact with the ad management system. Below is a summary of the available endpoints:

GET /alive

Checks if the API is alive and if the worker is running. **Response:**

- `200 OK` with a JSON object indicating the status of the API and the worker.

GET /queue

Retrieves the current tasks in the queue.

Response:

- `200 OK` with a JSON object containing the list of tasks in the queue.

PUT /clear_queue

Clears all tasks from the queue.

Response:

- `200 OK` with a JSON object indicating the queue has been cleared.

GET /start_worker

Starts the worker thread to process tasks in the queue.

Response:

- `200 OK` with a JSON object indicating the worker has started.

GET /stop_worker

Stops the worker thread.

Response:

- `200 OK` with a JSON object indicating the worker has stopped.

PUT /add_creds

Adds credentials for a dealer to the ad management system.

Request Body:

```
{
    "dealer": "admanager",
    "username": "rolf@mandrup.dk",
    "password": "Adm@1234"
}
```

Response:

- `200 OK` with a JSON object indicating the credentials have been saved.
- `400 Bad Request` if any required parameters are missing.

POST /create

Adds a task to create an order line to the queue.

Request Body:

```
{
    "dealer": "admanager",
    "worksheet": "558",
    "product_nr": "10",
    "product_amount": "5"
}
```

Response:

- `200 OK` with a JSON object indicating the task has been added to the queue.
- `400 Bad Request` if any required parameters are missing.

Example Usage

1. **Check if the API is alive:**

   ```
   curl -X GET http://localhost:5000/alive
   ```

2. **Add credentials:**

   ```
   curl -X PUT http://localhost:5000/add_creds -H "Content-Type:
   application/json" -d '{"dealer": "admanager", "username":
   "rolf@mandrup.dk", "password": "Adm@1234"}'
   ```

3. **Create an order line:**

```
curl -X POST http://localhost:5000/create -H "Content-Type:
application/json" -d '{"dealer": "admanager", "worksheet": "558",
"product_nr": "10", "product_amount": "5"}'
```

4. **Start the worker:**

```
curl -X GET http://localhost:5000/start_worker
```

5. **Stop the worker:**

```
curl -X GET http://localhost:5000/stop_worker
```

6. **Get the current queue:**

```
curl -X GET http://localhost:5000/queue
```

7. **Clear the queue:**

```
curl -X PUT http://localhost:5000/clear_queue
```

Folder Structure

```
__pycache__/
.env
add_creds_client.py
admanager_api.log
admanager_create_orderline.log
admanager_create_orderline.py
admanager_creds.txt
api.py
create_client.py
encrypt_test.py
persistQueueTest.py
pw.txt
requirements.txt
task_queue/
    info
    q00002
```

Files and Directories

add_creds_client.py - Script to add credentials to the ad management system.

admanager_api.log - Log file for the API.

admanager_create_orderline.log - Log file for admanager integration

admanager_create_orderline.py - Script for creating order lines using Selenium.

admanager_creds.txt - Encrypted credentials for the ad management system.

api.py - Main API server file using Flask.

create_client.py - Script for creating an orderline via the api

requirements.txt - File listing all the dependencies required to run the application.

.env - Environment file containing sensitive information such as the `SECRET_KEY`.

task_queue - Directory containing task queue files.

Start API

Start the API server:

```
python api.py
```

Setup

Install dependencies

The solution uses the following python components:

- Flask
- requests
- cryptography
- dotenv
- selenium
- persistqueue

Install the dependencies using pip:

```
pip install -r requirements.txt
```

Environment Variables

`SECRET_KEY`: Secret key for encryption, stored in the .env file:

```
SECRET_KEY=your_secret_key_here
```

Generate the secret key from commandline:

1. **Generate the secret key**: You can use the following Python script to generate a secret key and print it to the command line:

   ```
   python -c "from cryptography.fernet import Fernet;
   print(Fernet.generate_key().decode())"
   ```

2. **Set the generated key in the

   ```

   This will output something like:
   ```

   b'GkZy9J8Q7f8GkZy9J8Q7f8GkZy9J8Q7f8GkZy9J8Q7f8='

   ```

   ```

3. **Open your .env file and add the following line**:

   ```
   SECRET_KEY=GkZy9J8Q7f8GkZy9J8Q7f8GkZy9J8Q7f8GkZy9J8Q7f8=
   ```

Start API

```
python api.py
```