**Code Documentation (SERVER)**

**Importing Required Modules**

* **threading**: Provides support for multi-threading.
* **Flask** and related modules: Used for creating a web application.
* **datetime**: Provides classes for manipulating dates and times.
* **psycopg2** and related modules: Used for connecting to and interacting with PostgreSQL database.
* **psutil**: Provides an interface for retrieving system information and process utilities.
* **subprocess**: Allows the creation of new processes and the management of input/output pipes.
* **time**: Provides various time-related functions.
* **json**: Used for working with JSON data.
* **hashlib**: Provides various hashing algorithms.
* **requests**: Allows sending HTTP requests.
* **re**: Provides support for regular expressions.
* **os**: Provides a way of using operating system-dependent functionality.
* **socketio**: Provides support for building WebSocket applications.

**Flask Application Setup**

* An instance of the Flask application is created with **Flask(\_\_name\_\_)**.
* The **secret\_key** is set to 'mark' for session management.
* The **UPLOAD\_FOLDER** is set to the directory for file uploads.
* The **MAX\_CONTENT\_LENGTH** is set to 16 MB for limiting file sizes.
* An instance of SocketIO is created with **SocketIO(app)**.

**Global Variables and Constants**

* **clients**: A dictionary used for storing client connections.
* Database configuration variables (**db\_host**, **db\_port**, **db\_name**, **db\_user**, **db\_password**) for connecting to the PostgreSQL database.

**Route Handlers**

* **/**: Handles the login page. Authenticates the user and sets session variables.
* **/machines**: Retrieves machine data from the database and returns it as JSON.
* **/category**: Retrieves category data from the database and returns it as JSON.
* **/get\_name**: Retrieves fetched IP data from the database based on the given ID and returns it as JSON.
* **/insert\_machine\_name**: Updates the machine name and area in the fetched IP table based on the given ID and returns the updated data as JSON.
* **/insert\_controller**: Updates the controller name in the fetched IP table based on the given IP and returns the updated data as JSON.
* **/card\_details\_table**: Retrieves data from the machine\_data\_tbl table and returns it as JSON.
* **/delete\_data**: Deletes data from the machine\_data\_tbl table based on the given ID and returns success status as JSON.
* **/card\_details**: Retrieves card details data from the machine\_data\_tbl table and returns it as JSON.
* **/card\_details\_wirebond**: Retrieves wirebond card details data from the fetched\_ip\_tbl and machine\_data\_tbl tables and returns it as JSON.
* **/card\_details\_eol1**: Retrieves Eol1 card details data from the fetched\_ip\_tbl and machine\_data\_tbl tables and returns it as JSON.
* **/card\_details\_eol2**: Retrieves Eol2 card details data from the fetched\_ip\_tbl and machine\_data\_tbl tables and returns it as JSON.
* **/card\_details\_mold**: Retrieves Mold card details data from the fetched\_ip\_tbl and machine\_data\_tbl tables and returns it as JSON.

**Other Functions**

* **socketio.on\_event('connect')**: Event handler for client connections.
* **socketio.on\_event('disconnect')**: Event handler for client disconnections.
* **socketio.on\_event('get\_live\_data')**: Event handler for live data requests.
* **socketio.on\_event('get\_dashboard\_data')**: Event handler for dashboard data requests.
* **socketio.on\_event('stop\_program')**: Event handler for stopping a program.

**Configuration and Execution**

* Sets the **host** and **port** for the Flask application.
* Starts the Flask application with **socketio.run(app, host=host, port=port)**.

Title: Client

Description:

This code represents a client application that interacts with a Socket.IO server using Tkinter. It allows the client to connect to the server, send and receive data, and control the flow of operations.

Modules Used:

- tkinter: Used for creating the GUI window.

- json: Used for working with JSON data.

- socketio: Used for creating a Socket.IO client.

- uuid: Used for generating a unique client ID.

- os: Used for working with file paths.

- re: Used for string manipulation.

- signal: Used for handling signals.

- sys: Used for system-specific parameters and functions.

Global Variables:

- window: Represents the Tkinter window object.

- sio: Represents the Socket.IO client object.

- client\_id: A unique identifier for the client.

- filename: The name of the current file.

- json\_data: A dictionary containing sample JSON data.

- index: A counter for tracking the current index.

Functions:

- connect(): Event handler for the 'connect' event from the Socket.IO server.

- disconnect(): Event handler for the 'disconnect' event from the Socket.IO server.

- start(): Function to start sending data to the server.

- stop(): Function to stop sending data to the server.

- signal\_handler(signal, frame): Signal handler for the SIGINT signal.

Main Execution Flow:

1. Create a Tkinter window.

2. Create a Socket.IO client.

3. Initialize global variables.

4. Define the JSON data.

5. Define event handlers for 'connect' and 'disconnect' events.

6. Define the start() function to send data to the server.

7. Define the stop() function to stop sending data to the server.

8. Create the GUI buttons for starting and stopping.

9. Connect to the Socket.IO server.

10. Register the signal handler for handling SIGINT signal.

11. Run the Tkinter event loop.

Usage:

- Execute the code to start the client application.

- Click the "Start" button to send data to the server.

- Click the "Stop" button to stop sending data to the server.

- The client connects to the server automatically.

- The client disconnects from the server when the application is closed.

Note:

- Modify the JSON data (json\_data) and the Socket.IO server URL (sio.connect) according to your requirements.

- Customize the GUI and add more functionality as needed.