# API Testing Documentation

This document provides a detailed explanation of the API client written in Python to interact with a system that includes authentication, document ingestion, retrieval, chat session management, and WebSocket communication. The client utilizes aiohttp for asynchronous HTTP requests and WebSocket connections.

## Overview

The API client performs the following actions:

1. **Login**: Authenticates the user to obtain a token for further requests.
2. **Document Ingestion**: Uploads a document with a name and content.
3. **Document Retrieval**: Retrieves a list of documents.
4. **Chat Session Creation**: Creates a chat session to interact with documents.
5. **Adding Documents to Session**: Adds previously ingested documents to a chat session.
6. **WebSocket Communication**: Sends a query to the chat session and listens for a response via WebSocket.
7. **Logout**: Logs out the user and invalidates the token.

## Configuration

Before using the API client, ensure the following constants are correctly set:

ENDPOINT = "127.0.0.1:8000" # API base URL

AUTH\_BASE\_URL = f"http://{ENDPOINT}/auth" # Authentication URL

DOCUMENT\_BASE\_URL = f"http://{ENDPOINT}/documents/ingestion" # Document ingestion URL

CHAT\_BASE\_URL = f"http://{ENDPOINT}/qna/session" # Chat session URL

WEBSOCKET\_URL = f"ws://{ENDPOINT}/qna/docChat" # WebSocket URL for chatting

USERNAME = "admin" # Username for authentication

PASSWORD = "admin" # Password for authentication

TOKEN = None # Token for authentication (initially set to None)

## Functions

### 1. post\_request(session, url, data, headers=None)

**Description**: Handles HTTP POST requests.

**Parameters**:

* session: The aiohttp.ClientSession instance.
* url: The URL for the request.
* data: The data to send in the body of the request.
* headers: Optional headers for the request.

**Returns**:

* JSON response and HTTP status code.

### 2. get\_request(session, url, params=None, headers=None)

**Description**: Handles HTTP GET requests.

**Parameters**:

* session: The aiohttp.ClientSession instance.
* url: The URL for the request.
* params: Optional query parameters.
* headers: Optional headers for the request.

**Returns**:

* JSON response and HTTP status code.

### 3. websocket\_request(url, session\_uuid, token)

**Description**: Establishes a WebSocket connection, sends a query, and listens for a response.

**Parameters**:

* url: The WebSocket URL.
* session\_uuid: The UUID of the chat session.
* token: The authentication token for the WebSocket connection.

**Returns**: None (Prints the response to the console).

### 4. login()

**Description**: Authenticates the user and retrieves the token.

**Returns**:

* The token if login is successful, or None if authentication fails.

### 5. logout()

**Description**: Logs out the user by invalidating the authentication token.

**Returns**: None (Prints the logout status to the console).

### 6. ingest\_document()

**Description**: Ingests a document by sending its name and content.

**Returns**:

* The UUID of the ingested document if successful, or None if the ingestion fails.

### 7. get\_documents()

**Description**: Retrieves a list of documents.

**Returns**:

* The list of documents if successful, or None if retrieval fails.

### 8. create\_chat\_session()

**Description**: Creates a new chat session to interact with documents.

**Returns**:

* The UUID of the created session if successful, or None if session creation fails.

### 9. add\_documents\_to\_session(session\_uuid, document\_uuids)

**Description**: Adds documents to the specified chat session.

**Parameters**:

* session\_uuid: The UUID of the chat session.
* document\_uuids: A list of document UUIDs to add to the session.

**Returns**: None (Prints the status to the console).

### 10. test\_api()

**Description**: Main function to test the API flow by:

1. Logging in
2. Ingesting a document
3. Retrieving documents
4. Creating a chat session
5. Adding documents to the chat session
6. Sending a query through WebSocket
7. Logging out

**Returns**: None

## Running the Client

To run the client, simply execute the script as follows:

python api\_client.py

This will trigger the following sequence of actions:

1. **Login**: Authenticate using the provided credentials.
2. **Document Ingestion**: Upload a document to the system.
3. **Document Retrieval**: Retrieve documents and display them.
4. **Chat Session Creation**: Create a chat session and display the session UUID.
5. **Add Documents to Session**: Add the ingested document to the chat session.
6. **WebSocket Communication**: Send a query through the WebSocket connection and print the response.
7. **Logout**: Log out and invalidate the session token.

## Error Handling

The client handles HTTP status codes to determine success or failure:

* **200**: OK (Request succeeded).
* **201**: Created (Document or session was successfully created).
* **401**: Unauthorized (Login failed or invalid token).
* **400**: Bad Request (Invalid input data).
* **500**: Internal Server Error (Server-side issue).