AI Clone UI API Documentation

This document provides focused documentation for the API endpoints used by the AI Clone UI application. These are the specific endpoints that the Streamlit UI interacts with to provide functionality to end users.

API Endpoints Used by the UI

Domain Management

Get All Domains

`GET /domains`

Returns a list of all domains.

Response:

```json
[
  {
    "domain_name": "string",
    "expert_names": ["string"]
  }
]
```

Expert Management

Get All Experts

`GET /experts`

Returns a list of all experts.

Response:

```json
[
  {
    "id": "uuid",
    "name": "string",
    "domain": "string",
    "context": "string"
  }
]
```

Get Expert Context

`GET /experts/{expert_name}/context`

Gets the context for a specific expert.

Response:

```
{
  "context": "string"
}
```

Get Expert Domain

`GET /experts/{expert_name}/domain`

Gets the domain name for a specific expert.

Response:

```
{
  "domain_name": "string"
}
```

Update Expert Persona

`PUT /experts/persona/update`

Generates a persona from QA data and updates the expert's context.

Request Body:

```
{
  "expert_name": "string",
  "qa_pairs": [
    {
      "question": "string",
      "answer": "string"
    }
  ]
}
```

Response:

```
{
  "expert_name": "string",
  "persona": "string",
  "message": "string"
}
```

Document Management

Get Documents

`GET /documents`

Gets documents filtered by domain, expert, and client.

Query Parameters: - `domain`: string (optional) - `created_by`: string (optional) - `client_name`: string (optional)

Response:

```
[
  {
    "id": "uuid",
    "name": "string",
    "document_link": "string",
    "domain_name": "string",
    "created_by": "string",
    "client_name": "string (optional)",
    "file_id": "string"
  }
]
```

Vector Store Management

Update Vector Store

`POST /vectors/update`

Updates an existing vector store by adding new documents. Used for both domain and expert memory updates in the UI.

Request Body:

```
{
  "domain_name": "string (optional)",
  "expert_name": "string (optional)",
  "document_urls": {
    "document_name1": "url1",
    "document_name2": "url2"
  }
}
```

Response:

```
{
  "status": "string",
  "message": "string",
  "vector_id": "string",
  "domain_name": "string",
  "expert_name": "string (optional)",
  "client_name": "string (optional)",
  "new_file_ids": ["string"],
  "all_file_ids": ["string"],
  "batch_id": "string"
}
```

Memory Management

Initialize Expert Memory

`POST /memory/expert/initialize`

Initializes an expert's memory by creating domain, adding files, generating persona, and creating expert. This is used in the "Create expert" page of the UI.

Request Body:

```json
{
  "expert_name": "string",
  "domain_name": "string",
  "qa_pairs": [
    {
      "question": "string",
      "answer": "string"
    }
  ],
  "document_urls": {
    "document_name1": "url1",
    "document_name2": "url2"
  }
}
```

Response:

```json
{
  "expert_name": "string",
  "domain_name": "string",
  "status": "string",
  "message": "string",
  "results": {
    "domain": {},
    "domain_files": {},
    "persona": {},
    "expert": {},
    "expert_files": {}
  }
}
```

Query and Chat

Query Expert with Assistant

`POST /query_expert_with_assistant`

Queries an expert using the OpenAI Assistant API. This is a simplified endpoint that combines multiple Assistant API calls into a single request.

Request Body:

```
{
  "expert_name": "string",
  "query": "string",
  "memory_type": "string", // Options: "llm", "domain", "expert", "client"
  "client_name": "string (optional)",
  "thread_id": "string (optional)" // If provided, uses an existing thread
}
```

Response:

```
{
  "response": "string",
  "thread_id": "string",
  "assistant_id": "string"
}
```

OpenAI Assistant Integration

Create Assistant

`POST /create_assistant`

Creates an OpenAI Assistant for a specific expert and memory type.

Request Body:

```
{
  "expert_name": "string",
  "memory_type": "string", // Options: "llm", "domain", "expert", "client"
  "client_name": "string (optional)",
  "model": "string" // Default: "gpt-4o"
}
```

Response:

```
{
  "assistant_id": "string",
  "expert_name": "string",
  "memory_type": "string",
  "client_name": "string (optional)",
  "model": "string"
}
```

Create Thread

`POST /create_thread`

Creates a new thread for conversation.

Request Body:

```
{
  "expert_name": "string",
  "memory_type": "string", // Options: "llm", "domain", "expert", "client"
  "client_name": "string (optional)"
}
```

Response:

```
{
  "thread_id": "string",
  "expert_name": "string",
  "memory_type": "string",
  "client_name": "string (optional)"
}
```

Add Message

`POST /add_message`

Adds a message to a thread.

Request Body:

```
{
  "thread_id": "string",
  "content": "string",
  "role": "string" // Default: "user"
}
```

Response:

```
{
  "message_id": "string",
  "thread_id": "string",
  "content": "string",
  "role": "string"
}
```

Run Thread

`POST /run_thread`

Runs a thread with an assistant.

Request Body:

```
{
  "thread_id": "string",
  "assistant_id": "string"
}
```

Response:

```
{
  "run_id": "string",
  "thread_id": "string",
  "assistant_id": "string",
  "status": "string"
}
```

Get Run Status

`POST /get_run_status`

Gets the status of a run.

Request Body:

```
{
  "thread_id": "string",
  "run_id": "string"
}
```

Response:

```
{
  "run_id": "string",
  "thread_id": "string",
  "status": "string",
  "required_action": "object (optional)",
  "last_error": "object (optional)"
}
```

Get Thread Messages

`POST /get_thread_messages`

Gets messages from a thread.

Request Body:

```
{
  "thread_id": "string",
  "limit": "integer (optional)",
  "order": "string (optional)", // Default: "desc"
  "after": "string (optional)",
  "before": "string (optional)"
}
```

Response:

```
{
  "messages": [
    {
      "id": "string",
      "role": "string",
      "content": [
        {
          "type": "string",
          "text": "string"
        }
      ],
      "created_at": "integer"
    }
  ]
}
```

UI Workflow Examples

Creating a New Expert

1. Use `POST /memory/expert/initialize` to create a new expert with domain, persona, and documents.

Querying an Expert

Method 1: Direct Query (Simplified) 1. Use `POST /query_expert_with_assistant` to query an expert in a single API call

Method 2: Step-by-Step 1. Use `POST /create_assistant` to create an assistant for the expert 2. Use `POST /create_thread` to create a conversation thread 3. Use `POST /add_message` to add a user query to the thread 4. Use `POST /run_thread` to process the query with the assistant 5. Use `POST /get_run_status` to check if processing is complete 6. Use `POST /get_thread_messages` to retrieve the assistant's response

Updating Expert Memory

1. Use `GET /documents` to retrieve existing documents
2. Use `POST /vectors/update` with expert_name to add new documents to the expert's memory

Updating Domain Memory

1. Use `GET /documents` to retrieve existing documents
2. Use `POST /vectors/update` with domain_name to add new documents to the domain memory

Updating Expert Context

1. Use `PUT /experts/persona/update` to update the expert's persona based on QA pairs