Profiler

Release v1.0.0

Aadarsh Verma

Oct 16, 2024

CONTENTS:

CHAPTER

ONE

USER PROFILE API

This API manages user profiles, supporting CRUD operations, listing, filtering, and pagination.

Base URL: `/profiles/`

1.1 Available Endpoints

1. **List User Profiles** (GET)
2. **Retrieve a User Profile** (GET)
3. **Create a User Profile** (POST)
4. **Update a User Profile** (PUT)
5. **Partial Update a User Profile** (PATCH)
6. **Delete a User Profile** (DELETE)

—

1.2 1. List User Profiles

Method: `GET`

URL: `/profiles/`

Query Parameters:

- *limit* (int, optional): The number of profiles to return. Default is all.
- *offset* (int, optional): The starting point for profiles to return. Default is 0.
- *q* (string, optional): Search query for filtering profiles by username, first name, last name, or email.

1.3 Success Response (200 OK):

```
{
    "count": 10,
    "limit": 5,
    "offset": 0,
    "results": [
        {
            "id": 1,
            "first_name": "John",
            "last_name": "Doe",
            "email": "john.doe@example.com",
            "username": "johndoe"
        },
        {
            "id": 2,
            "first_name": "Jane",
            "last_name": "Doe",
            "email": "jane.doe@example.com",
            "username": "janedoe"
        }
        // More user profiles...
    ]
}
```

1.4 Error Response (400 Bad Request):

```
{
    "error": "limit and offset should be integers."
}
```

—

1.5 2. Retrieve a User Profile

Method: `GET`

URL: `/profiles/{id}/`

Retrieve a specific user profile by its unique *id*.

1.6 Success Response (200 OK):

```
{
    "id": 1,
    "first_name": "John",
    "last_name": "Doe",
    "email": "john.doe@example.com",
    "username": "johndoe"
}
```

1.7 Error Responses:

- **404 Not Found**: User profile not found.

```
{
    "detail": "Not found."
}
```

—

1.8 3. Create a User Profile

Method: `POST`

URL: `/profiles/`

1.9 Request Body (application/json):

```
{
    "first_name": "John",
    "last_name": "Doe",
    "email": "john.doe@example.com",
    "username": "johndoe",
    "password": "securepassword123"
}
```

1.10 Success Response (201 Created):

```
{
    "id": 1,
    "first_name": "John",
    "last_name": "Doe",
    "email": "john.doe@example.com",
    "username": "johndoe"
}
```

1.11 Error Responses:

- **400 Bad Request**: Invalid request data.

```
{
    "error": "Invalid data provided."
}
```

—

1.12 4. Update a User Profile

Method: PUT

URL: /profiles/{id}/

Update all fields of an existing user profile.

1.13 Request Body (application/json):

```
{
    "first_name": "John",
    "last_name": "Doe",
    "email": "john.doe@example.com",
    "username": "johndoe",
    "password": "newpassword123"
}
```

1.14 Success Response (200 OK):

```
{
    "id": 1,
    "first_name": "John",
    "last_name": "Doe",
    "email": "john.doe@example.com",
    "username": "johndoe"
}
```

1.15 Error Responses:

- **400 Bad Request**: Invalid data provided.

—

1.16 5. Partial Update a User Profile

Method: `PATCH`

URL: `/profiles/{id}/`

Update specific fields of an existing user profile.

1.17 Request Body (application/json):

You can provide only the fields you want to update.

```
{
    "email": "john.doe@newdomain.com"
}
```

1.18 Success Response (200 OK):

```
{
    "id": 1,
    "first_name": "John",
    "last_name": "Doe",
    "email": "john.doe@newdomain.com",
    "username": "johndoe"
}
```

1.19 Error Responses:

- **400 Bad Request**: Invalid data provided.

—

1.20 6. Delete a User Profile

Method: `DELETE`

URL: `/profiles/{id}/`

Delete a specific user profile by its *id*.

1.21 Success Response (204 No Content):

The user profile is deleted successfully.

1.22 Error Responses:

- **404 Not Found**: User profile not found.

```
{
    "detail": "Not found."
}
```

CHAPTER TWO

CHATBOT API

This API handles requests to interact with the chatbot.

URL: `/chatbot/`

Method: `POST`

Request Body Parameters: - *input* (string): The message or query sent to the chatbot. - *history* (array): An array of previous messages or interactions to provide context for the conversation. Each message is an object containing the *message*, *role*, and *time*.

Request Example:

```json
{
    "input": "Hello",
    "history": [
        {
            "message": "Hi, how are you?",
            "role": "user",
            "time": "2024-10-13T12:30:00"
        },
        {
            "message": "I'm good, thank you!",
            "role": "bot",
            "time": "2024-10-13T12:31:00"
        }
    ]
}
```

2.1 Success Response (200 OK):

```json
{
    "input": "Hello",
    "output": "Hi! How can I help you today?",
    "timestamp": "2024-10-13T12:35:00",
    "processing_time": 1.234
}
```

2.2 Error Response (400 Bad Request):

This response is returned when there is an error in the request data, such as missing fields or invalid format.

```
{
    "input": [
        "This field is required."
    ],
    "output": [
        "This field is required."
    ],
    "timestamp": [
        "This field is required."
    ],
    "processing_time": [
        "This field is required."
    ]
}
```

Possible Status Codes:

- **200 OK**: The chatbot returns a valid response.
- **400 Bad Request**: The request contains invalid or missing data, and validation errors are returned.

CHAPTER

THREE

DESCRIPTION GENERATOR API

This API generates a funny description based on user details like name, username, and email.

URL: `/description-generator/`

Method: `POST`

Request Body Parameters:

- *first_name* (string): User's first name.
- *last_name* (string): User's last name.
- *email* (string): User's email address.
- *username* (string): User's username.

Request Example:

```
{
    "first_name": "John",
    "last_name": "Doe",
    "email": "john.doe@example.com",
    "username": "johndoe"
}
```

3.1 Success Response (200 OK):

```
{
    "input": "Generate a funny description for John Doe, who goes by the username␣
↪johndoe. Their email is john.doe@example.com.",
    "output": "Meet John Doe, the master of dad jokes and a digital wizard! Known in the␣
↪virtual realm as 'johndoe', he can turn any email into a laugh-fest. Get ready for a␣
↪rollercoaster of fun!",
    "timestamp": "2024-10-13T12:35:00",
    "processing_time": 1.234
}
```

3.2 Error Response (400 Bad Request):

This response is returned when the input data is invalid or if required fields are missing.

```
{
    "error": {
        "first_name": ["This field is required."],
        "last_name": ["This field is required."],
        "email": ["This field is required."],
        "username": ["This field is required."]
    }
}
```

Possible Status Codes:

- **200 OK**: The request was successful, and the funny description is returned.
- **400 Bad Request**: The request contains invalid or missing parameters.