API Documentation

This API allows for basic user management, including adding, retrieving, updating, and deleting users. It supports both SQLite (in-memory for testing) and MySQL databases.

Endpoints

1. Add User

Endpoint: `/add-user`

Method: POST

Description: Adds a new user to the database.

Request:

- **`Content-Type:`** `application/json`
- **`Body:json`**

```
{
  "fullName": "John Doe",
  "username": "jdoe"
}
```

Response:

- **Status Code:** `201 Created`
- **Body:json**

```
{
  "id": 1,
  "fullName": "John Doe",
  "username": "jdoe"
}
```

Error Responses:

- **Status Code:** `400 Bad Request`

- **Body:json**

```
{
  "error": "Both full name and username are required"
}
```

Status Code: `500 Internal Server Error`

Body:json

```
{
  "error": "Error message"
}
```

2. Get Users

Endpoint: `/users`

Method: `GET`

Description: Retrieves users based on optional query parameters.

Request Parameters:

- **Query Parameters:**
    - `id`: User ID (optional)
    - `fullName`: User's full name (optional)
    - `username`: User's username (optional)

Response:

- **Status Code:** `200 OK`
- **Body:json**

```
[
 {
   "id": 1,
   "fullName": "Jane Doe",
   "username": "janedoe"
 }
]
```

Error Responses:

- **Status Code:** `404 Not Found`
- **Body:json**

```
[]
```

Status Code: `500 Internal Server Error`

Body:json

```
{
  "error": "Error message"
}
```

3. Update User by ID

Endpoint: `/update-user/<int:user_id>`

Method: PUT

Description: Updates a user based on their ID.

Request:

- **Content-Type:** `application/json`
- **Body:json**

```
{
  "fullName": "Jane Smith",
  "username": "janesmith"
}
```

Response:

- **Status Code:** `204 No Content`
- **Body: ""** (empty body)

Error Responses:

- **Status Code:** `400 Bad Request`

Body:json

```
{
  "error": "Both new full name and new username are required"
}
```

Status Code: `404 Not Found`

Body:json

```
{
  "error": "User not found"
```

}

Status Code: `500 Internal Server Error`

Body:json

{

 "error": "Error message"

}

4. Update User by Username

Endpoint: `/update-user`

Method: PUT

Description: Updates a user based on their old username.

Request Parameters:

- **Query Parameters:**
  - `oldName`: Old username (required)

Request:

- **Content-Type:** `application/json`
- **Body:json**

{

 "fullName": "Jane Smith",

 "username": "janesmith"

}

Response:

- **Status Code:** `204 No Content`

- **Body:** `""` (empty body)

Error Responses:

- **Status Code:** `400 Bad Request`

Body:json

```
{
  "error": "Old name, new full name, and new username are required"
}
```

Status Code: `404 Not Found`

Body:json

```
{
  "error": "Error message"
}
```

5. Delete User

Endpoint: `/delete-user`

Method: `DELETE`

Description: Deletes a user based on either user ID or username.

Request Parameters:

- **Query Parameters:**
  - `id`: User ID (optional)
  - `username`: User's username (optional)

Response:

- **Status Code:** `204 No Content`
- **Body:** `""` (empty body)

Error Responses:

- **Status Code:** `400 Bad Request`

Body:json

```
{
  "error": "Either user ID or username is required"
}
```

Status Code: `404 Not Found`

Body:json

```
{
  "status": "User not found"
}
```

Status Code: `500 Internal Server Error`

Body:json

```
{
  "error": "Error message"
}
```

Error Handling

- **400 Bad Request:** Indicates that the client request is malformed or missing required parameters.
- **404 Not Found:** Indicates that the requested resource (e.g., user) does not exist.
- **500 Internal Server Error:** Indicates a server-side issue or unexpected error.

Notes

- The API uses Flask and supports CORS to handle cross-origin requests.
- Database connections and schema creation are managed dynamically based on the environment (testing or production).

To run the API locally, follow these instructions:

Prerequisites

1. **Python**: Ensure you have Python 3.7 or later installed. You can download it from the official Python website.
2. **Pip**: Ensure `pip` is installed. It comes with Python installations.

Setup

3. **Clone the Repository**

First, clone the repository containing the API code:

git clone https://github.com/BlueHatThebe/FLASK-API-Task-

cd <repository-directory>

Create and Activate a Virtual Environment

It's a good practice to use a virtual environment to manage dependencies. Run the following commands to create and activate one:

On Windows:

python -m venv venv

Venv\Scripts\activate

On macOS/Linux:

source venv/bin/activate

Install Required Packages

Install the required Python packages using `pip`. If a `requirements.txt` file is available, use:

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

If there's no `requirements.txt` file, you can manually install the necessary packages. For this project, install:

```
pip install flask flask-cors mysql-connector-python
```

Set Up Environment Variables

Set the `TESTING` environment variable to use SQLite for testing. This step is optional and depends on whether you're running tests or working with MySQL.

On Windows:

```
$env:TESTING = "1"
```

On macOS/Linux:

```
export TESTING=1
```

Initialize the Database

Run the `app.py` script to set up the database schema:

```
python app.py
```

This command will create the necessary tables in the database.

Running the API

To start the Flask API server, run:

```
python app.py
```

By default, Flask will start the server on `http://127.0.0.1:5000/`. You should see output indicating the server is running: * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

Testing the API

- **Using `curl`**

Use `curl` to test API endpoints. For example, to add a user, run:

curl -X POST http://127.0.0.1:5000/add-user -H "Content-Type: application/json" -d '{"fullName": "John Doe", "username": "jdoe"}'

- **Using Postman**

You can use Postman to interact with the API through a graphical interface. Create requests to test different endpoints and view responses.

- **Running Automated Tests**

If you have automated tests, run them with:

python -m unittest discover

Make sure the database is correctly set up before running tests.

Live Link

After all instructions have been followed you may access the deployed app on this link below :

https://flask-api-task-4ptddn7wf-thebe-nkhasis-projects.vercel.app/

Stopping the API

To stop the Flask server, press `CTRL+C` in the terminal where the server is running.

Troubleshooting

- **Missing Packages**: Ensure all dependencies are listed in `requirements.txt` and installed.
- **Database Issues**: Verify database credentials and ensure the MySQL server is running if you're using MySQL.

Server Errors: Check terminal output for error messages and ensure the `app.py` script is correctly set up.