Project Report: Flask Application using RESTful API with MySQL Database

---

1. Introduction

This project is a **Flask CRUD Application** that allows users to manage (Create, Read, Update, and Delete) records in a MySQL database. The frontend is built with HTML, Bootstrap, and JavaScript, while the backend utilizes Flask (Python) to handle API requests. The application is connected to a MySQL database to store and manage user data.

---

2. Technology Stack

- **Backend**: Flask (Python)
- **Frontend**: HTML, Bootstrap, JavaScript (AJAX)
- **Database**: MySQL
- **API**: RESTful API built using Flask to perform CRUD operations

---

3. Folder Structure

The main components of the project include:

```
/project_folder
|
├── app.py                 # Main Flask application
├── index.html             # Frontend of the application
├── flask_api_db.sql       # SQL script to set up the database
```

---

4. Backend: Flask Application (app.py)

The backend logic is written in **Flask**. The `app.py` file sets up the routes for the following operations:

- **GET /users**: Fetch all users from the database.
- **POST /users**: Add a new user to the database.
- **PUT /users/<id>**: Update an existing user's information.
- **DELETE /users/<id>**: Delete a user from the database.

---

5. Frontend: HTML (index.html)

The **index.html** file provides the user interface for interacting with the Flask API. It uses **Bootstrap** for styling and **JavaScript (jQuery)** to handle AJAX requests and communicate with the backend.

Key functionalities include:

- **User List**: Displays all users fetched from the `/users` API.
- **Add User**: Allows the addition of new users via the form.
- **Update User**: Opens a modal to update the selected user.
- **Delete User**: Deletes the selected user from the table and the database.

JavaScript handles fetching and submitting data to the backend through AJAX requests:

---

6. Database: SQL Setup

The SQL file `flask_api_db.sql` contains the necessary SQL commands to set up the MySQL database and the `users` table.

Key components of the SQL setup:

Database Creation:
```
CREATE DATABASE IF NOT EXISTS flask_api_db;
```

Table Creation:
```
CREATE TABLE IF NOT EXISTS users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100) UNIQUE
);
```

You can use MySQL Workbench or the MySQL command line to execute the SQL script and set up your database.

---

7. How to Run the Project

1. **Set up the database**:

Import the `flask_api_db.sql` file into MySQL to create the necessary database and tables.

```
mysql -u root -p < /path/to/flask_api_db.sql
```

2. **Install dependencies**:

Ensure you have Flask and MySQL dependencies installed:

```
pip install Flask mysql-connector-python
```

3. **Run the Flask app**:

Start the Flask server:

```
python app.py
```

4. **Access the frontend**:

Open `index.html` in your browser and interact with the user management interface.

Alternatively, serve the HTML through Flask or a local server.

---

8. Conclusion

This Flask CRUD Application provides a fully functioning example of how to connect a Flask backend with a MySQL database and build a responsive frontend using Bootstrap and JavaScript. It demonstrates basic CRUD operations and can be extended for additional functionalities like user authentication, validation, or more complex database relationships.