Title: **Employee Data Updating** Date: 02/03/2024

Author: **Hardik Pachange**

Contact: **7028987616**

---------------------------------------------------------------------------------

Introduction:

The Employee Information Microservice is a Python script designed to interact with a database to manage employee records. It accepts Employee ID, Name, and Address from the user and stores this information in the database.

Features:

- Accepts input for Employee ID, Name, and Address.

- Stores the provided information in the database.

- Utilizes microservices architecture for seamless integration.

Here’s how the microservice architecture principles are followed in the provided code:

**Modularity**: The application is divided into multiple files, each responsible for a specific aspect:

app.py: Contains the Flask application and routes.

dbcheckprint.py: Handles database-related operations.

main.py: Entry point for running the Flask application.

Decentralized Data Management: The data management is decentralized to some extent. The dbcheckprint.py file encapsulates database operations, providing an abstraction for managing records. This separation allows for better maintainability and scalability.

* Independent Deployment: Each microservice (in this case, the Flask application and the database operations) can be deployed independently. This allows for easier scaling and updating of individual components without affecting the entire system.
* Resilience and Fault Tolerance: While the code doesn't explicitly handle resilience or fault tolerance mechanisms, Flask applications can be deployed in a resilient manner using techniques like load balancing, containerization, and service orchestration.
* Flexibility and Scalability: Flask applications can easily scale horizontally by deploying multiple instances behind a load balancer. Additionally, microservices can be independently scaled based on their resource demands.
* Inter-service Communication: In this application, there's no explicit inter-service communication as there's only one service (the Flask application) interacting with a database. However, in more complex microservices architectures, inter-service communication is often achieved through REST APIs, message queues, or other communication protocols.

Usage:

To use this microservice:

1. Run the script. (python main.py)
2. Open the browser and type (http://localhost:5000) with respective port number shown in terminal.
3. Enter Employee ID, Name, and Address.
4. Click Submit.

Dependencies:

Flask: Flask is a micro web framework written in Python. It is used for building web applications. In this script, Flask is used to create web routes, handle HTTP requests, and render HTML templates.

sqlite3: This is a built-in Python module that provides a lightweight disk-based database. In this script, it is used to interact with SQLite database to store and retrieve data.

dbcheckprint: This is a custom module (dbcheckprint.py) that contains functions create\_database() and fetch records which are used to create the database schema and fetch records from the database, respectively.

GitHub Link : <https://github.com/HardikSP111/PythonScript>

Screenshots:







