CRUD API Using Flask – Detailed Documentation

1. Introduction to CRUD APIs

CRUD stands for Create, Read, Update, and Delete. These operations represent the basic functionalities required to manage data in web applications. Flask is commonly used to build CRUD APIs due to its simplicity and flexibility.

2. What is a RESTful CRUD API

A RESTful CRUD API follows REST principles and uses HTTP methods to perform data operations:
• POST – Create data
• GET – Read data
• PUT / PATCH – Update data
• DELETE – Delete data

3. Basic Flask Application Setup

A Flask CRUD API starts by creating a Flask application instance and defining routes for each CRUD operation.

4. CREATE Operation (POST)

The CREATE operation is used to add new records to the database.

Endpoint: /api/items
Method: POST
Description: Creates a new item
Expected Input: JSON data
Response: Success message with status code 201

5. READ Operation (GET)

The READ operation retrieves existing data from the database.

Endpoint: /api/items
Method: GET
Description: Fetch all items
Response: List of items with status code 200

6. READ Single Record (GET by ID)

Endpoint: /api/items/<id>
Method: GET
Description: Fetch a single item by ID
Response: Item data or 404 if not found

7. UPDATE Operation (PUT)

The UPDATE operation modifies existing data.

Endpoint: /api/items/<id>
Method: PUT
Description: Update item details
Expected Input: JSON data
Response: Success message with status code 200

8. DELETE Operation (DELETE)

The DELETE operation removes data from the database.

Endpoint: /api/items/<id>
Method: DELETE
Description: Deletes an item
Response: Confirmation message with status code 200

9. HTTP Status Codes Used

• 200 OK – Request successful
• 201 Created – Resource created
• 400 Bad Request – Invalid input
• 404 Not Found – Resource not found
• 500 Internal Server Error – Server issue

10. CRUD API Flow Diagram (Textual)

Client → HTTP Request → Flask Route → Business Logic → Database → Response to Client

11. Best Practices for Flask CRUD APIs

- Validate request data
- Use proper HTTP methods
- Return meaningful status codes
- Handle errors gracefully
- Separate logic using Blueprints