## API Documentation for Attendance Management System

This document provides detailed information about the API, including supported HTTP methods, endpoints, input formats, and responses. The API manages user data, including creating, reading, updating, and deleting records in the `users` table of the database.

---

## Base URL

Copy code
http://<your-server-host>/attendance_system/api.php

---

## Supported HTTP Methods

1. **GET**: Retrieve all users.
2. **POST**: Add a new user.
3. **PUT/PATCH**: Update an existing user's details.
4. **DELETE**: Delete a user by `user_id`.
5. **HEAD**: Respond with headers only (no content body).
6. **OPTIONS**: Provides supported HTTP methods and CORS headers.

---

## Database Schema

The `users` table has the following structure:

| Field | Type | Description |
|---|---|---|
| user_id | VARCHAR(50) | Unique identifier for user |
| user_name | VARCHAR(100) | Name of the user |
| user_email | VARCHAR(100) | Email address of the user |
| user_fingerprint | VARCHAR(255) | Fingerprint data |

---

## Endpoints and Methods

### 1. Retrieve All Users

- **Method**: GET
- **Endpoint**: /api.php
- **Description**: Fetches all user records from the database.
- **Response**:

```
json
Copy code
[
  {
    "user_id": "1",
    "user_name": "John Doe",
    "user_email": "john@example.com",
    "user_fingerprint": "fingerprint123"
  },
  ...
]
```

---

### 2. Add a New User

- **Method**: POST
- **Endpoint**: /api.php
- **Description**: Adds a new user to the database.
- **Headers**:

  ```
  bash
  Copy code
  Content-Type: application/json
  ```

- **Request Body**:

  ```
  json
  Copy code
  {
    "user_id": "1",
    "user_name": "John Doe",
    "user_email": "john@example.com",
    "user_fingerprint": "fingerprint123"
  }
  ```

- **Response**:
  - Success:

    ```
    json
    Copy code
    {"message": "User added successfully!"}
    ```

  - Error:

    ```
    json
    Copy code
    {"error": "Invalid input data"}
    ```

---

### 3. Update an Existing User

- **Method**: PUT/PATCH
- **Endpoint**: /api.php
- **Description**: Updates an existing user's details.
- **Headers**:

  bash
  Copy code
  Content-Type: application/json

- **Request Body**:

  json
  Copy code
  {
  "user_id": "1",
  "user_name": "John Updated",
  "user_email": "john.updated@example.com"
  }

- **Response**:
  - Success:

    json
    Copy code
    {"message": "User updated successfully!"}

  - Error:

    json
    Copy code
    {"error": "No fields to update"}

---

### 4. Delete a User

- **Method**: DELETE
- **Endpoint**: /api.php
- **Description**: Deletes a user by user_id.
- **Headers**:

  bash
  Copy code
  Content-Type: application/json

- **Request Body**:

  json
  Copy code
  {
  "user_id": "1"

}

- **Response**:
  - Success:

    json
    Copy code
    {"message": "User deleted successfully!"}

  - Error:

    json
    Copy code
    {"error": "Invalid input data"}

---

### 5. HEAD Request

- **Method**: HEAD
- **Endpoint**: /api.php
- **Description**: Returns headers without a body.
- **Response**:

  bash
  Copy code
  HTTP/1.1 200 OK
  Content-Type: application/json

---

### 6. CORS and Allowed Methods

- **Method**: OPTIONS
- **Endpoint**: /api.php
- **Description**: Provides CORS headers and allowed HTTP methods.
- **Response Headers**:

  mathematica
  Copy code
  Access-Control-Allow-Methods: GET, POST, PUT, PATCH, DELETE, HEAD, OPTIONS
  Access-Control-Allow-Headers: Content-Type
  Access-Control-Allow-Origin: *

---

## Error Handling

- If an unsupported HTTP method is used:

  json
  Copy code

```
{"error": "Method not allowed"}
```

- If there is an error with database operations:

```
json
Copy code
{"error": "Error: <details>"}
```

- If required data is missing:

```
json
Copy code
{"error": "Invalid input data"}
```

---

## Security Considerations

1. **SQL Injection Prevention**:
   - Use prepared statements to secure database queries.
2. **Authentication**:
   - Add token-based authentication (e.g., JWT) for secure API access.
3. **Rate Limiting**:
   - Implement rate limiting to prevent abuse.

---

## Setup and Testing

1. **Setup**:
   - Import the attendance_system database schema.
   - Place this file in the web server's root directory.
2. **Testing**:
   - Use tools like **Postman** or **cURL** for API testing.

---

This documentation ensures that you can effectively use and expand upon the API.