Order Management System API's

❖ User APIs

1. Create User

- **Method: POST**
- **URL:/api/CreateUser**
- **Description:** Creates a new user in the database.
- **Request body:**

```
{
        "name": "John Doe",
        "email": "john.doe@example.com",
        "password": "password123"
}
```

- **Response:**
  - **200 Ok:-**

```
{
"message": "User added successfully",
"user": {
"id": 5,
"name": "John Doe",
"email": "john.doe@example.com",
"created_at": "2024-12-09T13:03:08.551371Z",
"updated_at": "2024-12-09T13:03:08.551371Z",
"deleted_at": {
"Time": "0001-01-01T00:00:00Z",
"Valid": false
                }
        }
}
```

- **400 Bad Request:-**

```
{

        "error": "Invalid input"

}
```

2. Fetch All Users

- **Method:GET**
- **URL:/api/FetchAllUser**
- **Description:** Retrieves a list of all users.
- **Request:** No request body required.

- **Response:**
  - **200 (OK):**

```
[
  {
        "id": 2,
        "name": "john dei ",
        "email": "johndei@gmail.com",
        "created_at": "2024-12-06T12:08:14.68829Z",
        "updated_at": "2024-12-06T12:08:14.68829Z",
        "deleted_at": {
        "Time": "0001-01-01T00:00:00Z",
        "Valid": false
        }
  },
  {
        "id": 3,
        "name": "sam ro",
        "email": "samro@gmail.com",
        "created_at": "2024-12-06T12:16:19.332165Z",
        "updated_at": "2024-12-06T12:16:19.332165Z",
        "deleted_at": {
        "Time": "0001-01-01T00:00:00Z",
        "Valid": false
                    }
  }
]
```

  - **500 (Internal server error):**

```
{

  "error": "Unable to fetch users"

}
```

3. Get User Details By User ID

- **Method:**GET
- **URL:** `/api/GetUserDetailByUserId/:id`
- **Description:** Retrieves details of a specific user by their user ID.
- **Request:**
  - **Path Parameter:** `id` **-** The ID of the user.
- **Response:**
  - **200 (OK):**

```
{
  "code": 200,
  "message": "User found",
  "description": "Successfully fetched the user details.",
  "user": {
    "id": 1,
    "name": "John Doe",
    "email": "john.doe@example.com",
    "created_at": "2024-01-01T00:00:00Z",
    "updated_at": "2024-01-01T00:00:00Z",
    "deleted_at": null
  }
}
```


- **404 Not Found:-**


```
{
  "code": 404,
  "message": "User not found"
}
```


- **500 Internal Server Error:**


```
{
  "code": 500,
  "message": "Unable to fetch data"
}
```

4.Update User Details

- **Method:**`PUT`
- **Endpoint:** `/users/:id`
- **Description:** Updates the details of a specific user by their ID.
- **Request Body:**

```
{

  "name": "JOHN",

  "email": "john@gmail.com"

}
```

- **Response Body:**

  200 OK:-

```
{

    "message": "User updated successfully"

}
```

  400 Bad Request: Invalid input.

  500 Internal Server Error: Failed to update user.

5. Soft Delete a User

- **Method:**`DELETE`
- **Endpoint:** `/users/:id`
- **Description:** Soft deletes a user by setting their `deleted_at` timestamp.
- **Request:**
  - **Path Parameter:** `id` **-** The ID of the user.
- **Response**

- **200 OK:-**

  {

    "message": "User deleted successfully"

  }

- **400 Bad Request:** User is already deleted.
- **500 Internal Server Error:** Failed to check or delete user.

❖ Order APIs

1. Create Order

- **Method:**`POST`
- **URL:** `/api/CreateOrder`
- **Description:** Creates a new order in the database. Calculates the total price, applies discounts, and inserts the order data along with the items into the database.

Request Body:

```
{
  "userID": 1,
  "status": "pending",
  "items": [
    {
      "itemID": 101,
      "quantity": 2
    },
    {
      "itemID": 102,
      "quantity": 1
    }
  ]
}
```

❖ **Response Body:**

- **200 OK:**


```
{
  "message": "Order created successfully",
  "order": {
    "id": 123,
    "userID": 1,
    "status": "pending",
    "totalPrice": 100.00,
    "finalPrice": 90.00,
    "created_at": "2024-12-09T13:03:08.551371Z",
    "updated_at": "2024-12-09T13:03:08.551371Z"
  },
  "order_items": [
    {
      "itemID": 101,
      "quantity": 2,
      "price": 20.00
    },
    {
      "itemID": 102,
      "quantity": 1,
      "price": 60.00
    }
  ]
}
```

- **400 Bad Request:**

  {

  "error": "Invalid input"

  }

- **500 Internal Server Error:**

  {

  "error": "Failed to create order"

  }

2. Get All Orders

- **Method:**`GET`
- **Endpoint:** `GET /orders`
  Description: Fetches a list of all orders along with their items.

Response:

- **200 OK:** Returns a list of orders with their items.

  {

  "Order": [

  {

  "ID": 1,

  "UserID": 101,

  "TotalPrice": 100.00,

  "Status": "Completed",

  "FinalPrice": 90.00,

  "Items": [

```
        {
          "ItemID": 201,
          "Quantity": 2,
          "Price": 45.00
        }
      ],
      "CreatedAt": "2024-12-09T00:00:00Z",
      "UpdatedAt": "2024-12-09T00:00:00Z",
      "DeletedAt": null
    }
  ]
}
```


- **500 Internal Server Error:**


```
{
  "error": "Unable to fetch orders"
}
```


3. Get Order by ID

- **Method:**`GET`
- **Endpoint:** `GET /orders/:id`
  Description: Fetches a specific order by its ID, including its items.

Response:

- **200 OK:** Returns the specific order with its items.

```
{
  "ID": 1,
  "UserID": 101,
  "TotalPrice": 100.00,
  "FinalPrice": 90.00,
  "Status": "Completed",
  "Items": [
    {
      "ItemID": 201,
      "Quantity": 2,
      "Price": 45.00
    }
  ],
  "CreatedAt": "2024-12-09T00:00:00Z",
  "UpdatedAt": "2024-12-09T00:00:00Z",
  "DeletedAt": null
}
```


- **404 Not Found:**


```
{
  "error": "Order not found"
}
```


- **500 Internal Server Error:**


```
{
```

```
  "error": "Unable to fetch order data"
}
```

4. Update Order by ID

- **Method:**`PUT`
- **Endpoint:** `PUT /orders/:id`
- **Description:** Updates the details of a specific order and its items by the order ID.

Request Body:

```
{
  "Status": "Shipped",
  "Items": [
    {
      "ItemID": 202,
      "Quantity": 1
    }
  ]
}
```

Response:

- **200 OK:**

```
{
  "message": "Order and items updated successfully"
}
```

- **400 Bad Request:**

  {

  "error": "Invalid input"

  }

- **404 Not Found:**

  {

  "error": "Order not found"

  }

- **500 Internal Server Error:**

  {

  "error": "Failed to update order"

  }

5. Update Order Status by Order ID

- **Method:PUT**
- **Endpoint**: `PUT/orders/:id/status`
- **Description**:Updates the status of an order to "Confirm" if the current status is "Pending".

Request Parameters:

- **id** (path parameter): The unique ID of the order to update.

Response:

- **200 OK**: If the order is successfully updated.

  {

  "message": "Order has been confirmed and placed successfully"

```
}
```

- **400 Bad Request**: If the order status is not "Pending".

```
{
  "error": "Order status is not 'Pending' (current status: <current status>)"
}
```

- **404 Not Found**: If the order with the given ID does not exist.

```
{
  "error": "Order not found"
}
```

- **500 Internal Server Error**: If there is a failure in the transaction or any other internal error.

```
{
  "error": "Failed to update order status"
}
```

6. Delete Order by Order ID

- **Method:**`DELETE`
- **Endpoint**: `DELETE /orders/:id`
- **Description**:Marks an order as deleted by updating the `deleted_at` field and sets its status to "Cancelled".

Request Parameters:

- **id** (path parameter): The unique ID of the order to delete.

Response:

- **200 OK**: If the order is successfully marked as deleted.

```
{
  "message": "Order deleted and status set to 'Cancelled' successfully"
}
```

- **400 Bad Request**: If the order is already deleted.

```
{
  "error": "Order already deleted"
}
```

- **500 Internal Server Error**: If there is a failure in the transaction or any other internal error.

```
{
  "error": "Failed to delete order"
}
```

❖ Item APIs

1.Add Item

- **Method:**`POST`
- **Endpoint:** `POST /items`
- **Description:** Add a new item to the database.
- **Request Body:**

```
{
   "name": "string",
   "description": "string",
   "price": "float"
}
```

Response:

- **Success** (HTTP 200 OK):

```
{
   "message": "Item added successfully",
   "item": {
      "id": "integer",
      "name": "string",
```

```
    "description": "string",
    "price": "float",
    "created_at": "timestamp",
    "updated_at": "timestamp"
  }
}
```


- **Error** (HTTP 400 Bad Request):


```
{
  "error": "Invalid input"
}
```


- **Error** (HTTP 500 Internal Server Error):


```
{
  "error": "Failed to insert item"
}
```


2.Get Items

- **Method:`GET`**
- **Endpoint**: `GET /items`
- **Description**: Retrieves all non-deleted items from the database.
- **Response**:
  - **Success** (HTTP 200 OK):


```
[
  {
    "id": "integer",
    "name": "string",
    "description": "string",
```

```
        "price": "float",
        "created_at": "timestamp",
        "updated_at": "timestamp",
        "deleted_at": "timestamp or null"
    },
    ...
]
```

- **Error** (HTTP 500 Internal Server Error):

```
{
    "error": "Unable to fetch data"
}
```

3.Get Item by ID

- **Method:**`GET`
- **Endpoint**: `GET /items/:id`
- **Description**: Retrieves a single item by its ID.
- **Path Parameters**:
  - `id` (integer): The ID of the item.
- **Response**:
  - **Success** (HTTP 200 OK)

```
{
    "id": "integer",
    "name": "string",
    "description": "string",
    "price": "float",
```

```
    "created_at": "timestamp",
    "updated_at": "timestamp",
    "deleted_at": "timestamp or null"
}
```

- **Error** (HTTP 404 Not Found):

```
{
  "error": "Item not found"
}
```

- **Error** (HTTP 410 Gone):

```
{
  "error": "Item has been deleted"
}
```

- **Error** (HTTP 500 Internal Server Error):

```
{
  "error": "Unable to fetch data"
}
```

4.Update Item

- **Method:PUT**
- **Endpoint**: `PUT /items/:id`
- **Description**: Updates an existing item based on its ID.
- **Path Parameters**:
  - `id` (integer): The ID of the item.
- **Request Body**:

```
{
  "name": "string",
```

```
"description": "string",
"price": "float"
}
```

- **Response**:
  - **Success** (HTTP 200 OK):

```
{
"message": "Item updated successfully"
}
```

  - **Error** (HTTP 400 Bad Request):

```
{
"error": "Invalid input"
}
```

  - **Error** (HTTP 500 Internal Server Error):

```
{
"error": "Failed to update item"
}
```

5.Delete Item (Soft Delete)

- **Method:**`DELETE`
- **Endpoint**: `DELETE /items/:id`

- **Description**: Marks an item as deleted by setting its `deleted_at` field to the current timestamp.
- **Path Parameters**:
  - `id` (integer): The ID of the item.
- **Response**:
  - **Success** (HTTP 200 OK):

```
{
    "message": "Item deleted successfully"
}
```

  - **Error** (HTTP 400 Bad Request):

```
{
    "error": "Item is already deleted"
}
```

  - **Error** (HTTP 500 Internal Server Error):

```
{
    "error": "Failed to delete item"
}
```