Homework 5

ECE567: Software Engineering 1

Tanya Sharma
tds104
30 October 2023

Design

The project involved the creation of a functional database, housing a single table with diverse records and attributes. The database was implemented using MySQL database management system. Additionally, a web server was developed to handle incoming client requests and facilitate interactions with the database. The server's capabilities centered on providing support for fundamental CRUD (Create, Read, Update, Delete) operations. To ensure a robust user experience, comprehensive error handling mechanisms were integrated. These included the identification and management of scenarios such as malformed URLs, requests for non-existent resources, and attempts to modify attributes that did not exist. When clients interacted with non-existent resources or provided incorrect attributes, the server was designed to respond with appropriate HTTP error codes and informative error messages, effectively communicating the nature of the issue to the client.

Commands

In order to execute the both the program as well as test cases the following instructions can be followed:

1. **npm install**
Install all necessary dependencies

2. **node main.js**
This command will execute the main function where all our logic takes place

3. **jasmine**
This command will execute the test suite

Database:

My project has two schemas, hw5 and hw5_test. The scheme hw5 is used for the main code functioning and has a table called People with the following attributes:
Table: People

CustomerID	int PK
FirstName	varchar(50)
LastName	varchar(50)
Email	varchar(100). Key
Phone	varchar(15)
Address	varchar(255)
City	varchar(50)

State	varchar(50)
ZipCode	varchar(10)
RegistrationDate	Date

Both CustomerID and Email must be unique.
The schema hw5_test is used for testing using jasmine and contains an identical People_test table.

Postman Requests

GET Request: Retrieve Data

Description: This API allows you to retrieve data from a MySQL database. It retrieves all records from the "People" table.
URL: http://localhost:8080/get-data or http://localhost:8080/get-data?CustomerID=1
HTTP Method: GET
Request: You can make a GET request to this URL to retrieve data.
Response: The server responds with a JSON array containing all records from the "People" table in the database. In case a CustomerID is provided as a parameter, the server returns the record for that corresponding CustomerID. If no such record exists, the server returns an error.
Postman: To retrieve data from the "People" table in the database, open Postman, create a new request, set the method to GET, and set the URL to http://localhost:8080/get-data. Send the request, and you will receive a JSON response containing the data from the "People" table. You can also try http://localhost:8080/get-data?CustomerID=1 with the CustomerID set to any existing record's value for CustomerID. The corresponding data will be retrieved.

POST Request: Add Data

Description: This API allows you to add a new record to the "People" table in the MySQL database.
URL: http://localhost:8080/add-data
HTTP Method: POST
Request: Send a POST request with a JSON body that contains data to be added. The JSON should include the fields: CustomerID, FirstName, LastName, Email, and optional fields such as Phone, Address, City, State, and ZipCode. Multiple records can also be inserted.
Response: If the data is added successfully, the server responds with a "Data Added Successfully" message. If there are missing required fields, it returns a "Bad Request" message.
Request Body(JSON):
(For adding a single record)- {

```
    "CustomerID": 456,
    "FirstName": "Alice",
    "LastName": "Johnson",
    "Email": "alice@example.com",
    "Phone": "789-456-1230"
}
(For adding multiple records)-
[
    {
        "CustomerID": 111,
        "FirstName": "John",
        "LastName": "Doe",
        "Email": "john.doe@example.com",
        "Phone": "555-555-5555",
        "Address": "123 Main St",
        "City": "Anytown",
        "State": "CA",
        "ZipCode": "12345"
    },
    {
        "CustomerID": 112,
        "FirstName": "Jane",
        "LastName": "Smith",
        "Email": "jane.smith@example.com",
        "Phone": "555-555-5556",
        "Address": "456 Elm St",
        "City": "Othertown",
        "State": "NY",
        "ZipCode": "54321"
    }
]
```

Postman: To add a new record to the "People" table, create a new request in Postman, set the method to POST, set the URL to http://localhost:8080/add-data, and include the JSON data in the request body as shown above. Send the request, and you should receive a response confirming that the data was added.

PUT Request: Update Data

Description: This API allows you to update existing records in the "People" table based on the CustomerID.
URL: http://localhost:8080/update-data
HTTP Method: PUT
Request: Send a PUT request with a JSON body that contains the CustomerID and the fields you want to update (e.g., FirstName, LastName, Email). You can include one or more fields to update. Since our updates are based on CustomerID which is a primary key of the table, only one record can be updated per request.

Response: If the data is updated successfully, the server responds with a "Data updated successfully" message. If the CustomerID is missing or no valid fields are provided, it returns a “Record not found” and "Bad Request" message respectively.
Request Body(JSON): {
"CustomerID": 456,
"FirstName": "UpdatedJohn",
"LastName": "UpdatedDoe",
"Email": "updatedjohndoe@example.com"
}
Postman: To update an existing record in the "People" table, create a new request in Postman, set the method to PUT, set the URL to http://localhost:8080/update-data, and include the JSON data with the CustomerID and the fields you want to update as shown above. Send the request, and you should receive a response confirming that the data was updated.

DELETE Request: Delete Data

Description: This API allows you to delete a record from the "People" table based on the CustomerID.
URL: http://localhost:8080/delete-data?CustomerID=<CustomerID>
HTTP Method: DELETE
Request: Send a DELETE request with the CustomerID as a query parameter in the URL.
Response: If the data is deleted successfully, the server responds with a "Data deleted successfully" message. If the CustomerID is missing or not provided in the query parameters, it returns a "Bad Request" message.
Postman: To delete a record from the "People" table, create a new request in Postman, set the method to DELETE, and set the URL to http://localhost:8080/delete-data?CustomerID=456 (replace 456 with the actual CustomerID you want to delete). Send the request, and you should receive a response confirming that the data was deleted.

Test Cases

Here is a brief description of the test cases in the provided code:

1. GET Request

- should return 200 when retrieving all records: This test checks if the GET request to retrieve all records returns a 200 status code and the response contains JSON data.
- should return 404 when route is invalid for GET: It tests that a GET request to an invalid route returns a 404 status code and a "Not Found" message.
- should return 200 when retrieving a specific record that exists: Verifies that a specific record can be retrieved with a valid CustomerID and returns a 200 status code.

- should return 404 when retrieving a specific record that does not exist: Ensures that a GET request for a non-existent record returns a 404 status code and a "Record not found" message.

2. POST request

- should return 200 when adding a new record with valid data: Validates that a POST request with valid data returns a 200 status code and a "Data Added Successfully" message.
- should return 400 Bad Request for a POST request with missing data: Verifies that a POST request with missing data fields returns a 400 status code and a "Bad Request" message.
- should return 400 Bad Request for a POST request with invalid JSON data: Ensures that a POST request with invalid JSON data returns a 400 status code.
- should return 404 when route is invalid for POST: Tests that a POST request to an invalid route returns a 404 status code and a "Not Found" message.

3. PUT request

- should return 200 when updating a valid record: Checks if a PUT request to update a valid record returns a 200 status code and a "Data updated successfully" message.
- should return 400 Bad Request for PUT request with missing or invalid data: Verifies that a PUT request with missing or invalid data fields returns a 400 status code and a "Bad Request" message.
- should return 404 when route is invalid for PUT: Tests that a PUT request to an invalid route returns a 404 status code and a "Not Found" message.
- should return 404 when updating a non-existent record: Ensures that attempting to update a non-existent record returns a 404 status code and a "Record not found" message.

4. DELETE request

- should return 200 when deleting a record with a valid CustomerID: This test checks if a DELETE request with a valid CustomerID returns a 200 status code and a "Data deleted successfully" message.
- should return 400 Bad Request when deleting a record with an invalid parameter: Verifies that a DELETE request with an invalid parameter returns a 400 status code and a "Bad Request" message.
- should return 400 Bad Request when deleting a record with a missing parameter: Ensures that a DELETE request with a missing parameter returns a 400 status code and a "Bad Request" message.
- should return 404 when route is invalid for DELETE: Tests that a DELETE request to an invalid route returns a 404 status code and a "Not Found" message.
- should return 404 when deleting a non-existent record: Verifies that attempting to delete a non-existent record returns a 404 status code and a "Data not found" message.

5. Concurrency test

- should handle multiple concurrent requests without errors: This test ensures that the server can handle multiple concurrent requests (5 in this case) without encountering errors. It checks if all requests return a 200 status code, indicating successful handling of concurrent requests.