Assignment 4

Objective: In this assignment, you will demonstrate your ability to write unit tests for a Flask application that interacts with a MongoDB database. You will create three specific unit tests for different routes and database operations, and then integrate these tests into your CI/CD pipeline in GitHub to ensure automated testing on every code change.

Use the project that you created for assignment 3 to implement these tests.

Write Three Unit Tests

In this step, you'll create three unit tests for the Flask application you developed previously. Make sure each test covers different aspects of the application's routes and database operations.

- **Test 1: Route Test**
    - Write a unit test for one of your routes (e.g., /home, /api/data) to verify that the endpoint returns the expected status code when an invalid request is sent. For example, if your route accept GET request, implement a POST request in your test and the route should return 405 status code.
    - Use the Flask test client to simulate a request to the route and validate the response.
- **Test 2: Database Read Operation**
    - Write a unit test to check the correct connection of a MongoDB read operation. Hint: Use the ping to verify the correct connection.
    - Refer to this link for more information: https://stackoverflow.com/questions/42565304/is-it-possible-to-ping-mongodb-from-pymongo
- **Test 3: Database Write Operation**
    - Write a unit test for a MongoDB write operation (e.g., inserting a new document).
    - Use assertions to ensure the document was successfully inserted by querying the database and checking the data's presence. For example:

```python
def test_write_data_to_db(db):
    new_data = {"field": "new_value"}
    db.collection.insert_one(new_data)
    inserted_data = db.collection.find_one({"field": "new_value"})
    assert inserted_data is not None
    assert inserted_data['field'] == 'new_value'
```

Note: Include comments in the code to describe what each test does.

2. Integrate Unit Tests with CI/CD Pipeline in GitHub

Once the tests are written, you will integrate them with your CI/CD pipeline in GitHub. Modify your GitHub Actions configuration file to include these tests, ensuring that they run automatically whenever code is pushed to the repository.

- **Create or Update .github/workflows/ci.yml**:
  - Ensure the CI/CD pipeline installs dependencies and sets up the test environment.
  - Add a "Run Tests" step to execute the tests.

3. Submission Requirements:

- **GitHub Repository**:
  - Ensure your tests are included in the "tests/" directory within your project.
  - Include your CI/CD pipeline configuration file (.github/workflows/ci.yml) in your repository.
  - Submit the link of the GitHub project in the submission box.

Grading Criteria:

- **Correctness of Tests (40%)**: The tests should accurately verify the route response, MongoDB read operation, and MongoDB write operation.
- **CI/CD Integration (40%)**: The GitHub Actions pipeline should run your unit tests automatically and report the results.
- **Documentation (20%)**: Clearly explaining each test's purpose and integration with the CI/CD pipeline.