CSCC01 Assignment 2: CI/CD Pipeline Implementation

Team Name: Microsofties

---

1. Project Overview

This project implements a CI/CD pipeline for a full-stack web application using GitHub Actions, Docker, and Render. The application includes a React frontend and a Node.js/Express backend, with MongoDB Atlas used for persistent data storage. The pipeline's purpose is to ensure automation of building, testing, and deploying containers, enabling streamlined development and delivery.

---

2. CI/CD Design

Our CI/CD pipeline consists of the following stages:

- **Continuous Integration (CI)**
    - Runs automatically on each push to the main branch.
    - Executes unit tests on both frontend and backend using Jest.
    - Builds Docker images for frontend and backend.
    - Tags the images with both latest and semantic versions (e.g., v1.0.0).
    - Pushes the tagged images to Docker Hub.
- **Continuous Deployment (CD)**
    - Deploy hooks are triggered for both backend and frontend services on Render.
    - After deployment, integration tests are executed to verify the health and functionality of the live containers.

---

3. Technologies Used

- **GitHub Actions** - for automating the CI/CD workflow
- **Docker & Docker Hub** - for containerization and versioned image storage
- **Render** - for hosting and running the deployed containers
- **MongoDB Atlas** - for remote database services
- **Jest** - for unit testing
- **curl** - for post-deployment integration testing

---

4. CI/CD Workflow Steps

1. Checkout code from GitHub.
2. Run unit tests in both frontend and backend folders.
3. Build Docker images.
4. Tag with latest and v4.0.0.
5. Push images to Docker Hub.
6. Trigger Render deploy hooks.
7. Wait for containers to boot.
8. Run integration tests via a shell script.

---

5. Automated Testing Strategy

- **Unit Testing:**
  - Backend and frontend each contain Jest test suites.
  - Tests are automatically executed during the CI phase in GitHub Actions.
- **Integration Testing:**
  - A shell script (integration-test.sh) uses curl to validate deployed containers.
  - Tests include:
    - Backend health check (/api/health)
    - Backend data route (/api/water-data)
    - Frontend root response (/)

---

6. Environment Variables

- **Backend (on Render):**
  - MONGODB_URI: MongoDB Atlas connection string
  - JWT_SECRET: secret used for signing JWTs
- **Frontend (on Render):**
  - REACT_APP_API_URL: live backend URL (https://glow-backend-v4-0-0.onrender.com)

---

7. Challenges and Solutions

- **Problem:** MongoDB connection errors due to invalid credentials and missing DB name.
  Solution: Ensured MongoDB Atlas users and connection strings were properly configured and URL-encoded.
- **Problem:** Frontend failed to connect to backend after deployment.
  Solution: Added environment variable REACT_APP_API_URL before building and deploying frontend.
- **Problem:** Render did not allow URL renaming via service rename.
  Solution: Recreated the service with the desired subdomain.

---

8. Final Remarks

All requirements of the assignment have been satisfied:

- CI and CD pipelines are implemented.
- Docker images are versioned and deployed.
- Unit and integration tests are fully automated.
- Final artifacts are included in the /CICD directory as required.

Live services are hosted on Render and are accessible publicly. The system is robust, repeatable, and production-ready.
