Project Title: Birthday Reminder Service

Objective

Develop a backend application that stores user data, including their birthdays. Additionally, a worker should be implemented to send a "Happy Birthday" message to users at 9 AM in their local time zones on their birthday.

Requirements

Functionality

1. **User Management**
    - Create a user with the following fields:
        - `name` (string, required)
        - `email` (string, required, unique, must be a valid email address)
        - `birthday` (ISO 8601 date format, required)
        - `timezone` (string, required, valid IANA timezone, e.g., `America/New_York`)
    - Retrieve a user's details by ID.
    - Update user details.
    - Delete a user.
2. **Birthday Messaging Worker**
    - Implement a worker process that:
        - Sends a "Happy Birthday" message (e.g., via console log or email simulation) to each user at 9 AM in their local timezone on their birthday.
        - Processes birthdays accurately, considering timezones and scheduling.
3. **Validation**
    - Ensure all input data is validated (e.g., check for valid email and timezone formats).

Technology Stack

- **Backend**: Node.js with Express or NestJS.
- **Database**: MongoDB.
- **Worker**: Use a library like `node-cron` or `agenda` for scheduling.
- **Containerization**: Docker (with a `docker-compose.yml` file to set up both the app and MongoDB).
- **Testing**: Use a framework like Jest or Mocha for unit testing.

Deliverables

1. **Codebase**
    - Backend API implementation for user management.
    - Worker implementation for birthday message scheduling.
    - Unit tests for:
        - User creation and validation.
        - Worker functionality (e.g., ensuring the correct message is sent at the correct time).
2. **Docker Setup**
    - A `Dockerfile` for the application.
    - A `docker-compose.yml` file to run both the application and MongoDB.
3. **Documentation**
    - Instructions are given to run the application and the worker using Docker.
    - API examples
    - Brief notes on assumptions, limitations, and design decisions.

Expectations

- Clean, modular, and maintainable code.
- Proper handling of edge cases (e.g., invalid input, timezone errors).
- Effective use of scheduling libraries for the worker.
- Meaningful unit test cases with high code coverage.

Evaluation Criteria

1. Code quality and maintainability.
2. Functionality completeness and adherence to requirements.
3. Robust handling of time zones and scheduling.
4. Test coverage and quality.
5. Clarity and completeness of documentation.