BRAD API Testing Policy

Purpose

This policy defines the approach and standards for testing the BRAD API to ensure code quality, reliability, and maintainability. Automated tests verify that the API behaves correctly under various conditions, prevents regressions, and ensures seamless integration of system components.

Frameworks and Tools

- **Jest**: Primary test framework for unit and integration. Supports mocking, spies, and test configuration via jest-unit.json and jest-integration.json.
- **mongodb-memory-server**: Provides an in-memory MongoDB instance for integration tests, ensuring isolated and repeatable test environments.
- **Supertest**: Used for testing HTTP endpoints in integration .
- **Nodemailer Mock**: Used to mock email sending (e.g., OTP and lockout notifications) in unit and integration tests to avoid external dependencies.
- **GitHub Actions**: Automates the testing pipeline, ensuring tests run on every pull request and deployment branch.

Justification

- Jest is well-integrated with NestJS, supports TypeScript, and provides robust mocking capabilities for deterministic tests.
- mongodb-memory-server ensures integration tests are isolated from production databases, reducing flakiness.
- Supertest simplifies HTTP request testing for API endpoints.
- Nodemailer Mock removes external email service dependencies, ensuring reliable tests.
- GitHub Actions provides continuous integration, automated test execution, and deployment triggers, ensuring code stability before merging or deploying.

Policy

1. Unit Test Creation

- Every new function, class, or module (e.g., services, controllers) must have corresponding unit tests.
- Unit test files are stored in test/unit/ and named to match the source module, e.g.:
    - src/auth/auth.service.ts → test/unit/auth.unit-spec.ts
- Unit tests must mock external dependencies (e.g., MongoDB, Redis, HTTP requests, Nodemailer) using Jest mocks to ensure deterministic behavior.
- Example: Mock transporter.sendMail in auth.unit-spec.ts to simulate email failures without sending actual emails.

Policy

1. Unit Test Creation

- Every new function, class, or module (e.g., services, controllers) must have corresponding unit tests.
- Unit test files are stored in test/unit/ and named to match the source module, e.g.:
    - src/auth/auth.service.ts → test/unit/auth.unit-spec.ts
- Unit tests must mock external dependencies (e.g., MongoDB, Redis, HTTP requests, Nodemailer) using Jest mocks to ensure deterministic behavior.
- Example: Mock transporter.sendMail in auth.unit-spec.ts to simulate email failures without sending actual emails.

2. Integration Test Creation

- For each new API workflow (e.g., user registration → OTP verification → login), an integration test must be created in test/integration/.
- Integration tests must validate:
    - MongoDB interactions (using mongodb-memory-server).
    - Authentication flows (e.g., JWT generation, OTP verification).
    - HTTP endpoints (e.g., /auth/login, /auth/register) using Supertest.
    - Queue interactions (e.g., Bull queue for report processing) using mocked or in-memory queues.
    - Error handling for invalid inputs, expired tokens, or service failures.

3. Test Coverage

Unit and integration tests should cover:

- **Core API Functions**:
    - User authentication (AuthService: register, login, verifyOtp).
    - Password management (forgotPassword, resetPassword, changePassword).
    - Report submission and processing (ReportService: submitReport, report_analysis).
- **Error Handling**:
    - Invalid credentials or tokens (UnauthorizedException).
    - Duplicate email/username (ConflictException).
    - Invalid password formats (BadRequestException).
    - Database or queue failures.

- **End-to-End Flow**:
    - User registration → OTP email sent → OTP verification → JWT issued.
    - Report submission → queued for analysis → processed → API response.

4. Test Maintenance

- Tests must be updated alongside code changes to reflect new functionality or API contract changes.
- Deprecated tests must be removed or refactored if functionality is retired.
- Mocks (e.g., Nodemailer, MongoDB, Bull) must be reviewed regularly to align with real service contracts.
- Address issues like the worker process failed to exit warning by ensuring proper teardown (e.g., closing MongoDB connections in afterEach/afterAll).

5. Execution

- **Local**: Developers run tests with:

npm run test

This executes both unit and integration tests sequentially (npm run test:unit && npm run test:integration).

Merge Policy: No code is merged into main unless all tests pass.

Deployment Gate: CI/CD ensures tests pass before deployment to staging/production.

Goal

This testing policy ensures the BRAD API is thoroughly validated at unit and integration levels, catching regressions early and maintaining confidence in deployments. Automated tests, combined with CI/CD, uphold a consistent quality standard throughout the API's lifecycle.