Save-n-Bite Testing Policy

Version: 1.0 Date: 2025-09-28 Repository: COS301-SE-2025/Save-n-Bite

1. Purpose

- **Ensure quality and reliability** of the Save-n-Bite backend via automated testing.
- **Prevent regressions** by running tests on every change locally and in CI.
- **Document a repeatable process** for contributors and CI to execute tests consistently.

2. Scope

This policy applies to the backend service located at `save-n-bite-backend/`, covering: - **Unit and small integration tests** (within Django apps) - **Integration & End-to-End (E2E)** flows across components - **Non-Functional Requirements (NFR)** tests (performance, reliability) - **Security and performance** sanity checks

3. Test Strategy

- **Test pyramid** with a strong base of unit tests and targeted , Integration, E2E & NFR suites.

3.1 Testing Tools Justification

Core Testing Framework

- **pytest**: Chosen over unittest for its:
  - More concise test syntax with fixtures
  - Better assertion introspection
  - Rich plugin ecosystem
  - Parallel test execution support

Test Coverage

- **pytest –cov**: For measuring code coverage to:
  - Identify untested code paths
  - Ensure critical paths are tested
  - Maintain coverage standards (target 80%+)

CI/CD Pipeline

- **GitHub Actions**: Selected over Travis CI because:
  - Native GitHub integration
  - Better performance and concurrency

  - Built-in container support
  - Free for public repositories
  - Simpler YAML-based configuration
- **Isolated, deterministic runs** using dedicated test settings (`in-memory SQLite` locally) and **ephemeral PostgreSQL** in CI.
- **Idempotent setup/teardown** to avoid state leakage between runs.

3.1 Test Levels & Locations

- **App-level unit/integration tests**: Each Django app contains its own tests, discovered automatically by Django under either:
  - `app_name/tests.py` (single file), or
  - `app_name/tests/` (package with multiple test modules).
- **Integration tests**: `save-n-bite-backend/tests/test_integration_simple.py`, `test_integration.py`
- **End-to-End tests**: `save-n-bite-backend/tests/test_end_to_end.py`
- **NFR tests**: `save-n-bite-backend/tests/test_nfr_simple.py`, `test_non_functional.py`
- **Support & config**: `save-n-bite-backend/tests/test_config.py`, `tests/README.md`

3.2 Test Data & Environment

- **Local**: `save-n-bite-backend/test_settings.py` uses SQLite `:memory:` DB for speed; lightweight email/cache backends, fast password hashing.
- **CI**: GitHub Actions spins up PostgreSQL 15 and Redis 7 services. Dedicated test DBs are created and cleaned each run.

4. How To Run Tests

4.1 Local

- Run all suites with integrated report:

```
cd save-n-bite-backend
python run_integration_tests.py --all --verbose
```

- Simple runner (select suites):

```
python run_tests_simple.py --all --verbose
# or specific suites
python run_tests_simple.py --integration
python run_tests_simple.py --e2e
python run_tests_simple.py --nfr
```

- Direct Django invocation:

```
python manage.py test tests.test_integration_simple --settings=backend.test_settings --
```

- Run tests for a specific app (Django test discovery picks up `tests.py` and `tests/`): “‘bash # Example: run only the interactions app tests (which live under interactions/tests/)
  - `poetry run python manage.py test` (with proper flags), and
  - `poetry run python run_integration_tests.py --all --verbose`
- **Coverage**: upload step prepared; run step commented to keep pipelines fast. Enable by uncommenting the coverage block.

5. Test Data Management

- Tests construct data programmatically; no persistent fixtures required.
- Where applicable, tests enforce isolation and deterministic outcomes.
- Password hashing uses MD5 in tests (fast-only, never in prod) as configured in `test_settings.py`.

6. Test Reports

Test Reports

How to Generate Reports

```
# Run tests with coverage
pytest --cov=./ --cov-report=xml --cov-report=html --junitxml=test-reports/junit.xml

# Generate integration test report
python run_integration_tests.py --all --report-dir=test-reports/
```

8. Reporting & Artifacts

- **Coverage reports (optional)**: When enabled, uploaded via Actions artifact at `save-n-bite-backend/htmlcov/`.

8. Quality Gates & Policy Enforcement

- All PRs must pass CI test runs on `main` before merge.
- Non-zero exit codes fail the pipeline.
- NFR tests validate performance and reliability criteria regularly.

9. Contribution Workflow

1. Create a feature branch from `main`.
2. Write/adjust tests:
   - At the app level inside each Django app (`app_name/tests.py` or `app_name/tests/`).
   - For cross-app flows and system scenarios, use `save-n-bite-backend/tests/`.
3. Run tests locally (see Section 4.1).

4. Open a PR; GitHub Actions will run the test workflow.
5. Address any failures; ensure the pipeline is green.