Save n Bite – Coding Standards Document

Project: Save n Bite
Team: Secure Web & Mobile Guild (SWMG)
Version: 4.0
Date: September 27, 2025

Secure Web & Mobile Guild

Team:
Sabrina-Gabriel Freeman
Marco Geral
Chisom Emekpo
Vanè Abrams
Capleton Chapfika

Contact Email: swmguild@gmail.com

CONTENTS

Secure Web & Mobile Guild
swmguild@gmail.com

1. OVERVIEW

Defines conventions to ensure clarity, maintainability, and security.

Stack

- Django + DRF, PostgreSQL, Redis
- React + Vite, Tailwind
- Github
- Azure

2. CORE PRINCIPLES

Readability: Self-documenting, consistent naming.

DRY & SOLID: Reuse, modular design.

Error Handling: Centralized logging + custom exceptions.

Documentation: Docstrings, API docs, migration notes, READMEs.

3. REPOSITORY STRUCTURE

save-n-bite/

├── backend/ (Django apps: auth, food_listings, analytics, etc.)

├── frontend/ (React src: components, hooks, services, utils)

├── documentation/

└── README.md

Rules: Apps = single domain, React by feature, tests mirror source, configs at root.

4. BACKEND (PYTHON/DJANGO)

Style: PEP 8, Black (88 chars), isort, mypy.

Naming: snake_case (vars), PascalCase (classes), UPPER_SNAKE_CASE (constants).

Models: Use *`db_table`*, *`ordering`*, helper methods (*`is_expired()`*).

Views/Serializers: RESTful, permissions, validation in serializers.

Error Handling: Custom exceptions, logging best practices.

5. FRONTEND (REACT/JS)

Style: ESLint + Prettier.

Naming: PascalCase (components), camelCase (utils/vars).

Components: Functional, prop-types, async handling with try/catch.

Hooks: Encapsulate fetch + state (*`useFoodListings`*).

Styling: Tailwind utility-first; avoid inline styles.

6. DATABASE (POSTGRESQL)

Tables/Columns: lowercase_with_underscores.

PK/FK: *`id`*, *`{table}_id`*.

Indexes & Constraints: For performance/integrity.

Migrations: Always documented.

7. API STANDARDS

- RESTful endpoints, plural nouns.
- Proper HTTP methods + status codes.
- JSON response format with *`success`*, *`data`*, *`error`*.
- Pagination for lists.

8. TESTING

Coverage: 70%+ overall, 90% critical flows.

Types: Unit (utils), integration (API/db), E2E (key journeys).

Frameworks: Django Test Framework | Jest + RTL.

9. VERSION CONTROL

Branches: *`main`*(prod), *`dev`*(integration), `backend`, `frontend', *'feature/*`*, *`bugfix/*`*, *`release/*`*.

Commits: Conventional (*`feat`*, *`fix`*, *`docs`*, *`style`*, *`refactor`*, *`test`*, *`chore`*).

PRs: Descriptive, reference issues, require review + passing tests.

10. ENVIRONMENT & CONFIG

- *`.env`*for secrets (never committed).
- Use *`decouple`*for Django settings.
- Consistent config across dev/prod.

11. QUALITY TOOLS

Backend: Black, isort, flake8, mypy.

Frontend: ESLint, Prettier, Jest, RTL.

12. SECURITY

Auth: JWT, RBAC, hashed passwords.

Data: Encryption, validation, SQL injection prevention, file restrictions.

Privacy: Minimal storage, retention policies, export/delete support.

Environment: HTTPS, HSTS, CSRF protection, strict CORS.

Secure Web & Mobile Guild
swmguild@gmail.com

13. MAINTENANCE

Follow standards, perform code reviews, run automated checks, and update guidelines as project evolves.

Maintainers: Marco Geral, Sabrina-Gabriel Freeman

Last Updated: Sep 27, 2025