Contributing Guide

Thank you for considering contributing to sailaps! This guide will help you get started.

Code of Conduct

Be respectful and inclusive in all interactions with the project and community members.

Getting Started

1. Fork and Clone

```
# Fork the repository on GitHub
# Clone your fork
git clone https://github.com/YOUR_USERNAME/sailapps.git
cd sailapps

# Add upstream remote
git remote add upstream https://github.com/jstanbri/sailapps.git
```

2. Create Feature Branch

```
# Update from upstream
git fetch upstream
git checkout upstream/main

# Create feature branch
git checkout -b feature/your-feature-name
```

3. Set Up Development Environment

```
python -m venv .venv
.\.venv\Scripts\Activate.ps1

pip install -r requirements.txt
pip install pytest pytest-cov pylint black isort mypy
```

Development Workflow

1. Make Changes

Follow the code standards defined in `.agent-context/PYTHON.md`:

- Use snake_case for functions and variables
- Use PascalCase for classes
- Add type hints to all function signatures
- Write docstrings for all public functions

2. Write Tests

For every feature or bug fix:

```python
# tests/test_your_feature.py
import pytest

class TestYourFeature:
    def test_feature_works(self):
        """Test your feature."""
        # Arrange
        expected = "result"

        # Act
        result = your_function()

        # Assert
        assert result == expected
```

3. Run Quality Checks

```bash
# Format code
black src/ tests/

# Sort imports
isort src/ tests/

# Run linting
pylint src/ tests/ --exit-zero

# Type checking
mypy src/

# Run tests with coverage
pytest --cov=src --cov-report=term-missing
```

4. Commit Changes

```bash
git add .
git commit -m "feat: Add new feature
```

```
Detailed description of the changes.
Explain the motivation and any tradeoffs."
```

Use conventional commit prefixes:

- `feat:` - New feature
- `fix:` - Bug fix
- `docs:` - Documentation changes
- `refactor:` - Code refactoring without feature changes
- `test:` - Adding or updating tests
- `chore:` - Build, dependencies, or tooling

5. Push and Create Pull Request

```
git push origin feature/your-feature-name
```

Create a pull request on GitHub with:

- Clear description of changes
- Reference to related issues
- Test evidence (test results, coverage)

Code Standards

Python Style

- Follow PEP 8
- Maximum line length: 88 characters (Black)
- Use 4 spaces for indentation

Type Hints

Add type hints to all function signatures:

```python
def process_file(filename: str) -> dict[str, int]:
    """Process a file and return statistics."""
    pass
```

Documentation

Write clear docstrings using Google style:

```python
def calculate_total(items: list[float]) -> float:
    """Calculate the sum of items.
    Args:
        items: List of numeric values to sum

    Returns:
        The sum of all items

    Raises:
        ValueError: If items list is empty
    """
    if not items:
        raise ValueError("Items list cannot be empty")
    return sum(items)
```

Testing Requirements

- Minimum 80% code coverage
- All new features must include tests
- All bug fixes must include regression tests
- Use the Arrange-Act-Assert (AAA) pattern

Documentation

Update documentation for:

- New features
- API changes
- Breaking changes
- Configuration options

Update relevant files in `docs/`:

- `usage.md` for usage changes
- `api.md` for API changes
- `installation.md` for setup changes

Review Process

A maintainer will review your pull request for:

- Code quality and style compliance
- Test coverage and quality
- Documentation completeness
- Alignment with project goals

Address feedback promptly and update your PR accordingly.

Release Process

Releases follow semantic versioning:

- `MAJOR.MINOR.PATCH`
- `v0.1.0`, `v0.2.1`, etc.

The maintainers handle releasing to PyPI and creating GitHub releases.

Questions?

- Open a discussion on GitHub
- Check existing issues and pull requests
- Review the Usage Guide

Thank you for contributing!