Python/Flask Coding Standards for Personal Projects

Table of Contents

Code Structure

Project Organization

```
my_flask_app/
├── app/
│   ├── __init__.py          # Flask application factory
│   ├── config.py            # Configuration settings
│   ├── models/              # Database models
│   ├── routes/              # Route definitions
│   ├── services/            # Business logic
│   ├── static/              # CSS, JS, images
│   └── templates/           # Jinja2 templates
├── tests/                   # Test files
├── venv/                    # Virtual environment
├── .gitignore
├── requirements.txt         # Dependencies
└── wsgi.py                  # WSGI entry point
```

Code Layout

- Limit lines to 79-88 characters
- Use 4 spaces for indentation (no tabs)
- Separate top-level functions and classes with two blank lines
- Separate methods within classes with one blank line
- Use blank lines sparingly inside functions to indicate logical sections

Naming Conventions

- **Packages/Modules**: `lowercase` (e.g., `models`, `utils`)
- **Classes**: `CamelCase` (e.g., `UserProfile`, `PaymentProcessor`)
- **Functions/Methods**: `snake_case` (e.g., `get_user_data`, `validate_form`)
- **Variables**: `snake_case` (e.g., `user_id`, `total_count`)
- **Constants**: `UPPER_CASE` (e.g., `MAX_CONNECTIONS`, `DEFAULT_TIMEOUT`)
- **Flask Blueprints**: `snake_case` (e.g., `auth_bp`, `admin_bp`)

Endpoint URLs

- Use lowercase with hyphens for readability:
  - Good: `/api/user-profiles`
  - Avoid: `/api/userProfiles` or `/api/user_profiles`

Documentation

Docstrings

Use Google style docstrings:

```python
def get_user_by_email(email):
    """Retrieve a user by their email address.

    Args:
        email (str): The email address to search for

    Returns:
        User: The user object if found

    Raises:
        UserNotFoundError: If no user with that email exists
    """
```

Comments

- Focus on why, not what (the code should be self-explanatory)
- Keep comments updated when changing code
- Use TODO comments for temporary fixes or planned improvements:

```python
# TODO: Replace with database lookup when implemented
```

README

Maintain a clear README.md with:

- Project description
- Setup instructions
- Dependencies
- Configuration steps
- Basic usage examples

Version Control

Git Practices

- Use semantic commit messages:

  ```
  feat: add user authentication
  fix: resolve login redirect issue
  docs: update API documentation
  refactor: simplify data processing logic
  ```

- Commit frequently with focused changes
- Use feature branches for new development
- Avoid committing sensitive data (credentials, API keys)

.gitignore

Include a comprehensive `.gitignore` file:

```
# Python
__pycache__/
*.py[cod]
*$py.class
venv/
env/
.env

# Flask
instance/
.webassets-cache

# Testing
.coverage
htmlcov/

# Editor files
.vscode/
.idea/
*.swp
```

Testing

Testing Framework

Use pytest for testing:

```
pytest tests/
```

Test Structure

```python
# tests/test_user_service.py
def test_create_user_success():
    """Test that users can be created successfully."""
    # Setup
    user_data = {"username": "testuser", "email": "test@example.com"}

    # Execute
    user = create_user(user_data)

    # Verify
    assert user.username == "testuser"
    assert user.email == "test@example.com"
```

Coverage

- Aim for at least 70% code coverage for hobby projects
- Prioritize testing critical components (authentication, data processing)

Security

Input Validation

- Never trust user input
- Use Flask-WTF for form validation
- Validate and sanitize all data from clients

Authentication

- Use a proven library like Flask-Login
- Store passwords with strong hashing (bcrypt, Argon2)
- Implement proper session management

Environment Variables

Use environment variables for sensitive values:

```python
# config.py
import os

SECRET_KEY = os.environ.get('SECRET_KEY', 'dev-key-never-use-in-production')
DATABASE_URI = os.environ.get('DATABASE_URI', 'sqlite:///dev.db')
```

HTTPS

Always configure HTTPS, even for hobby projects. Let's Encrypt provides free certificates.

Performance

Database

- Use SQLAlchemy ORM with care
- Index columns used in frequent queries
- Avoid N+1 query problems with eager loading
- Use query pagination for large datasets

Caching

- Consider Flask-Caching for expensive operations:

```python
from flask_caching import Cache
cache = Cache(app)

@cache.cached(timeout=60)
def get_weather_data():
    # Expensive API call
    return data
```

Resource Loading

- Use a CDN for static assets when possible
- Minify CSS/JS files for production

Deployment

Ubuntu VPS Setup

1. Create a dedicated user for deployment:

```bash
sudo adduser flaskapp
sudo usermod -aG sudo flaskapp
```

2. Set up Python environment:

```bash
sudo apt update
sudo apt install python3-pip python3-venv nginx
```

3. Configure Gunicorn for production:

```bash
pip install gunicorn
```

Service Setup

Create a systemd service file `/etc/systemd/system/flaskapp.service`:

```
[Unit]
Description=Flask Application
After=network.target

[Service]
User=flaskapp
WorkingDirectory=/home/flaskapp/myapp
ExecStart=/home/flaskapp/myapp/venv/bin/gunicorn -w 3 -b 127.0.0.1:8000 wsgi:app
Restart=always

[Install]
WantedBy=multi-user.target
```

Nginx Configuration

```
server {
    listen 80;
    server_name yourdomain.com;

    location / {
        proxy_pass http://127.0.0.1:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }

    location /static {
        alias /home/flaskapp/myapp/app/static;
    }
}
```

CI/CD Considerations

For a hobby project, a simple deployment script:

bash

```bash
#!/bin/bash
cd /home/flaskapp/myapp
git pull
source venv/bin/activate
pip install -r requirements.txt
sudo systemctl restart flaskapp
```

Resources

Tools

- Black: Code formatter
- Flake8: Linter
- isort: Import sorter
- Pre-commit: Git hooks

Configuration

Add a `setup.cfg` file:

```
[flake8]
max-line-length = 88
extend-ignore = E203
exclude = venv/*,migrations/*

[isort]
profile = black
```

Helpful Commands

bash

```bash
# Format code with Black
black app/ tests/

# Check imports
isort app/ tests/ --check

# Run linter
flake8 app/ tests/

# All checks at once (with pre-commit)
pre-commit run --all-files
```