Swift Signals

Coding Standards and Practices

Contents

Overview

This document defines the coding standards, conventions, and practices for this repository. Our project is a **microservice-based system** comprising:

- **Frontend**: React + TypeScript
- **API Gateways**: Go + gRPC
- **Simulation Services**: Python
- **Databases**: PostgreSQL C MongoDB

Maintaining clean, consistent, and reliable code across all services is critical for scalability, maintainability, and team collaboration.

General Principles

- **Consistency Over Cleverness**: Prioritize readability and maintainability over fancy constructs.
- **Fail Fast**: Validate inputs early; handle errors explicitly.
- **Tests Are Non-Negotiable**: All new code must include tests where applicable.
- **Documentation First**: Comment complex logic, provide clear README updates when interfaces change.
- **Security and Performance Mindset**: Apply least privilege principles, sanitize inputs, avoid premature optimization, but flag performance concerns.

General Guidelines

- **Indentation:**
    - **Python: 4 spaces**
    - **Go: tabs**
    - **JS/TS: 2 spaces**
- **Line length:** Max 100 characters
- **Naming:**
    - **Variables and functions:**
        - Python: snake_case
        - Go: MixedCaps
        - JS/TS: camelCase
    - **Constants:** ALL_CAPS
    - **Classes/Structs:** PascalCase
- **Comments:** Explain *why*, not *what/how*. Prefer self-documenting code.

- **Note:** Go uses uppercase/lowercase symbols for visibility, which may differ from naming rules above.

Frontend (React + TypeScript)

Code Style

- Follow **Airbnb Style Guide** for React with TypeScript.
- Enforce via **ESLint**, **Prettier**, and **Husky** pre-commit hooks.

Key Conventions:

Aspect	Standard
File Structure	Feature-based folders, index.ts for exports
Components	Functional Components, Hooks preferred over classes
Type Usage	Strict typing, use interface for public shapes
Props/State	Typed explicitly, avoid any unless unavoidable
Testing	**Jest + React Testing Library** for unit/integration tests

Example:

```tsx
interface UserCardProps {
  name: string;
  age: number;
}

const UserCard: React.FC<UserCardProps> = ({ name, age }) => (
  <div>{`${name} is ${age} years old`}</div>
);
```

API Gateways (Go + gRPC)

Code Style

- Follow **Effective Go** guidelines.
- Use golangci-lint for linting.
- Protobufs version-locked via .proto files with clear versioning (v1, v2).

Key Conventions:

Aspect	Standard
Project Layout	Follow Go Project Layout
Error Handling	Explicit error checks, wrap errors with context
gRPC Practices	Avoid large payloads; version services with care
Dependency Management	Use Go Modules (go.mod) with version pinning
Testing	**Go Test,** table-driven tests, use mocking dependencies

Example:

```go
if err := db.Save(item); err != nil {
    return fmt.Errorf("failed to save item: %w", err)
}
```

Simulation Service (Python)

Code Style

- Follow **PEP8**, enforced via flake8.
- Type hints are mandatory for all new code.
- Use black and isort for formatting.

Key Conventions:

Aspect	Standard
Project Layout	Source in src/, tests in tests/
Type Hints	Use Python 3.10+ type annotations
Testing	**pytest**, ensure coverage for critical paths
Virtual Envs	Use venv or poetry for environment isolation

Example:

```python
def simulate(event: str, duration: float) -> dict:
    """Run a simulation for the given event."""
    return {"event": event, "duration": duration}
```

Database Standards

PostgreSQL

- SQL scripts version-controlled in /migrations/postgres.
- Naming: lowercase with underscores (snake_case).
- Use parameterized queries to prevent SQL injection.
- Schema changes reviewed via pull requests.

MongoDB

- Collections and fields use camelCase.
- No dynamic field insertion at runtime; define expected schemas where possible (e.g., via Pydantic or similar validators).

- Indexes optimized for frequent queries.

Testing G CI/CD

Area	Tool/Standard
Linting	ESLint (TS), golangci-lint (Go), flake8 (Py)
Formatting	Prettier (TS), gofmt (Go), black (Py)
Tests	Jest, Go Test, pytest
Coverage	Minimum 80% enforced per service
CI/CD	GitHub Actions, pipelines fail in lint or test errors

Security Best Practices

- No hard-coded secrets; use environment variables or secret managers.
- Sanitize all external inputs.
- Use HTTPS for all external communication.
- Regular dependency audits (e.g., npm audit, go mod tidy, pip check).

Documentation G Comments

- All public interfaces documented.
- Complex logic receives inline comments.
- API contracts defined via Protobufs and updated as part of version-controlled artifacts.

Versioning

- Semantic Versioning (MAJOR.MINOR.PATCH) for APIs and services.
- Breaking changes require MAJOR version bumps and clear migration paths.

Final Notes

- Pull Requests require at least **1 approved reviewer**.
- All code must pass linters, formatters, and tests before merging.
- This document evolves—team contributions and improvements are welcome.

End of Standards