Coding Standards document

AT-AT Coding Standards Document

Introduction

This document defines the coding conventions and standards used across all three layers of the **AT-AT (API Threat Assessment Tool)** project to ensure our code is uniform, clear, reliable and efficient, which includes:

- A **JavaScript-based REST API** (Node.js + Express)
- A **Python engine** for API threat analysis
- A **React-based frontend** user interface

---

1. JavaScript API (Node.js + Express)

Code Style

- **Indentation:** 2 spaces
- **Semicolons:** Always use
- **Quotes:** Single quotes preferred
- **Trailing commas:** Required in multiline constructs
- **Brace style:** Always use braces for blocks

```
// Good
if (user) {
  return true;
}
```

Naming Conventions

- `camelCase` for variables, functions and filenames
- UPPER_SNAKE_CASE for environment variables

Comments

Single-Line Comments: Use ' // ' for single-line commments Multi-Line Comments: Use ' /* ... */' for multi-line comments

Example of Comments from coverage.js

```
// scripts/coverage.js - Coverage Analysis Helper

/**
 * Parse coverage summary and display formatted results
 */
function displayCoverageSummary() { ... }
```

Error Handling

For error handling, try-catch statements are used to ensure errors are properly caught and handled.

---

2. Python Engine

Code Style

- Follow **PEP8** standards
- Indentation: 4 spaces

Naming Conventions

- `snake_case` for variables, functions, and filenames
- `PascalCase` for class names

Logging

Use the `logging` module instead of `print()` for production logs:

```
import logging
logging.info("Spec uploaded successfully")
```

Imports

- Standard imports first, third-party next, local modules last

```
import os
import yaml
from engine.scanner import ThreatScanner
```

---

3. React Frontend

Folder Structure

Organize by feature, not file type. Example:

```
/src
  /components
    Navbar.jsx
    TagEditor.jsx
  /pages
    UploadPage.jsx
    ReportPage.jsx
  /assets
    logo.svg
  /styles
    global.css
```

JavaScript & JSX

- Use `camelCase` for functions and variables
- Use `PascalCase` for components
- Use arrow functions for components
- Keep components focused and reusable

```
const UploadForm = () => {
  return <form>...</form>;
};
```

CSS

- Use CSS Modules or component-level styles
- Use `kebab-case` for class names
- Example: `tag-list.module.css`

State & Props

- Use `useState`, `useEffect`, `useContext` appropriately
- Validate props with `PropTypes` or TypeScript if used

API Integration

Use a dedicated `api.js` or `services/` folder for API calls:

```
export const fetchEndpoints = async (apiId) => {
  const res = await fetch(`/api/endpoints`, { ... });
  return res.json();
};
```

---

4. Git & Workflow Standards

Branch Naming

- `main` → Stable code
- `dev` → Development integration
- `feature/<name>` → Feature branches
- `fix/<name>` → Bug fixes

Commits

Use clear, human-readable commit messages that briefly describe the change

Pull Requests

- Use descriptive titles

- Link related issues if tracked
- Ensure all tests/linting pass before merging

Enforcement & Verification (Change4)

Area	Where it applies	Tool / Config	How to run (local)	Status
Formatting (JS)	API	Prettier (default config)	`npm run format`	✅ in repo (`/api`)
Lint (JS)	API	ESLint	`npm run lint` / `npm run lint:fix`	✅ in repo (`/api`)
Lint (JS)	Frontend	CRA built-in ESLint (`react-app` preset)	via `react-scripts` (start/build)	✅ implicit (no separate script)
Tests (JS)	API	Jest	`npm run test` / `npm run test:unit`	✅
Tests (JS)	Frontend	React Testing Library via CRA	`npm test -- --coverage`	✅
Tests (Python)	Backend/Engine	pytest	`pytest -q`	✅ (`requirements.txt`)
Static analysis	Repo	GitHub **CodeQL**	—	✅ workflow present

By adhering to these conventions, all contributors to the AT-AT project ensure a cohesive, readable, and professional-grade codebase for API threat assessment.