Testing Strategy

Overview

At present, the ACT application does not include automated testing. However, introducing a comprehensive test suite would help ensure stability, reduce regressions, and simplify onboarding for future contributors.

Below is a recommended **testing strategy** covering unit tests, integration tests, and potential end-to-end (E2E) flows.

Suggested Testing Layers

Type	Description
Unit Tests	Test individual functions (e.g., translateCode, get_embedding) for correctness and error handling.
Integration Tests	Test API endpoints (/translate, /save-translation) to ensure full flow correctness.
End-to-End Tests (E2E)	Simulate real user behavior in the browser (e.g., logging in, submitting code, copying output). Useful for validating critical paths.

Recommended Tools

Layer	Tool / Framework
Unit	jest (JavaScript), pytest (Python)
Integration	supertest (Node.js), pytest + requests
E2E	Playwright or Cypress
Test Runner	GitHub Actions, Docker test service setup (optional)

How to Run Tests

Once testing is implemented, test execution can follow these patterns:

🧪 Node.js Backend (API + Express)

```
cd ACT-app
npm install
npm test
```

🧪 Python Translator API

```
cd ACT-ml
pip install -r requirements.txt
pytest
```

For integration tests, mocks can be used for Authentik and OpenAI endpoints. Consider using VCR.py to record HTTP interactions.

🧪 Sample Test Flow Suggestions

- **Unit Test** translateDelphiToCSharp() – ensure proper API call and error catching
- **Unit Test** the /translate route with mocked FAISS and OpenAI response
- **Integration Test** login + translation + save flow using an authenticated token
- **E2E Test** – simulate login and translation in a browser using Playwright

🧾 Test Coverage Reporting

Currently, there are **no automated test coverage reports**. When implemented, the following tools can help:

Language	Tool	Output

JS	jest --coverage	HTML summary
Python	coverage.py	Terminal + HTML

Major Gaps (as of now):

- No testing around login/authentication behavior
- No test coverage on CodeMirror logic or drag-and-drop interface
- No failover tests for external dependencies (OpenAI, Authentik)

✅ Future Action Plan

1. Introduce unit tests for translation and database logic.
2. Add integration tests with mock Authentik login.
3. Write at least one E2E test simulating user flow using Playwright or Cypress.
4. Add GitHub Actions job for CI test verification.
5. Measure code coverage and prioritize test cases accordingly.