Janeeb
Solutions

TESTING POLICY DOCUMENT

Extended Planning Instrument for Unpredictable Spaces and Environments

University Of Pretoria
janeeb.solutions@gmail.com

Introduction

Testing is a crucial part of ensuring the quality, performance, and reliability of the logistics optimization system. This system, designed to improve the placement of goods in logistics trucks, leverages machine learning and dynamic algorithms. The following guide provides an overview of the testing procedures, including unit tests using Vitest and integration tests using Postman API.

Testing Overview

The logistics optimization system uses a combination of layered architecture, API Gateway, and service-oriented architecture (SOA). The backend is developed using TypeScript with Supabase for database interactions, while the frontend is built with Vue and Three.js for visualization.

The two primary types of testing performed in this system are:

- **Unit Tests**: Ensure the accuracy of isolated functions, algorithms, and components.
- **Integration Tests**: Ensure the proper interaction between system components, especially API functionality and data flow.

Unit Testing

Setup

Unit tests verify the correctness of individual components such as algorithms, database queries, and business logic. For unit testing, we use Vitest.

Ensure the following packages are installed:

```
npm install vitest --save-dev
npm install @types/node --save-dev
```

Writing Unit Tests

1. Create a test file in the `__tests__` directory, with a `_test.ts` suffix. Example: `packing_algorithm_test.ts`.
2. Use Vitest to mock external services and verify that components like the dynamic packing algorithm work correctly.

Example test with Vitest:

```
import { describe, expect, test } from 'vitest';
import { calculateOptimalPacking } from '../src/packing_algorithm';

describe('Packing Algorithm', () => {
  test('calculates optimal packing correctly', () => {
    const input = {/* mock input data */};
    const result = calculateOptimalPacking(input);
    expect(result).toEqual(expectedOutput);
  });
});
```

Running Unit Tests

To run the unit tests using Vitest, use the following command:

```
npx vitest
```

Integration Testing

Setup

Integration tests ensure the proper functioning of multiple components together, focusing on APIs and workflows. These tests are executed using Postman, which tests API routes by simulating real-world scenarios.

1. Create a collection in Postman that includes all relevant API endpoints (e.g., `POST /pack-items`, `GET /status`).
2. Set up different test cases for each endpoint, testing both normal and edge cases for requests and responses.

Writing Integration Tests

- In Postman, create requests for each API endpoint.
- Use the built-in "Tests" tab in Postman to write assertions. For example, verify response codes, data formats, and values.

Example in Postman:

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.test("Response contains correct packing results", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData).to.have.property('packedItems');
    pm.expect(jsonData.packedItems).to.be.an('array');
});
```

Running Integration Tests

- Run individual tests or collections using Postman's interface or via the **Newman CLI** (Postman's command-line companion) to automate test execution.

To run using Newman:

```
newman run <your_postman_collection.json>
```

Conclusion

The logistics optimization system uses Vitest for unit tests to verify the accuracy of individual components and Postman API for integration tests to ensure the system's components work together as expected. Both types of tests will run automatically using GitHub Actions upon code commits and pull requests, ensuring the system remains robust and scalable.