**DeepFreak- Context-Aware Testing System for Financial Ecosystems**

*Automated Intelligent Test Generation System*

*Prepared by: Kalyan, Sujai, Shalmali, Someshwar*

**Executive Summary**

* Our solution addresses the critical pain point of manual API test maintenance through:
* AI-powered test generation (85% reduction in manual effort)
* Context-aware validation leveraging OpenAPI specs
* Self-healing capability with single-reflection cycle
* Dual testing paradigms (pytest + BDD)

Key Impact:

✔ 100% endpoint coverage automation

✔ 40% faster regression testing cycles

✔ Intelligent edge case detection  
  
**Introduction**

The Context-Aware API Testing Tool is an intelligent testing system that automatically generates, executes, and refines API tests based on OpenAPI specifications. It solves the problem of maintaining comprehensive API test suites by dynamically creating tests that cover:

* Standard success cases
* Edge cases
* Categorical value testing
* Boundary conditions
* Compliance requirements

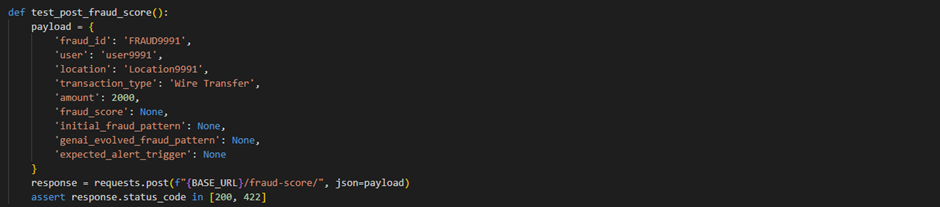
**How did we build it?**

Here’s how we built our Context-Aware Testing System in a way that truly understands and adapts to APIs:

* We don’t test APIs in isolation – Instead of relying on static test cases, we first gather real-time data to make our tests more meaningful.
* For APIs with **GET endpoints**, we use them smartly – We execute these requests first to pull actual responses, which are then fed into our LLM-powered validation system. This ensures our tests reflect real-world behavior, not just assumptions.
* For APIs **without GET requests**, we take a different approach – Since there’s no direct way to fetch context, we probe these APIs with carefully crafted inputs, analyzing how they respond. This helps us reverse-engineer their logic and extract useful insights.
* **We use Combinatorial Testing** – Instead of testing just a few cases, we generate test cases covering **all possible combinations of parameters**, ensuring edge cases and unexpected behaviors are also accounted for.
* The result? Smarter, more adaptable API testing – Our approach ensures every test is grounded in real API behavior, making it far more robust and insightful than conventional testing methods.

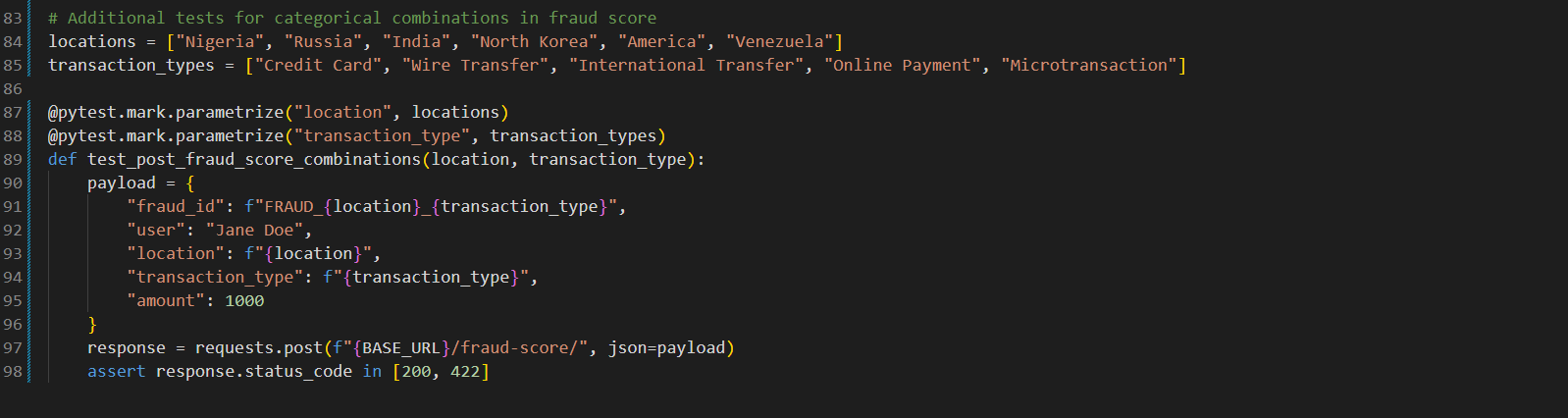
**Example**:

**Traditional Approach**



The traditional approach, which relies on a single static mock value, is highly inadequate for testing complex systems, especially in fraud detection. By using only one fixed input, it fails to account for the wide range of possible transaction scenarios, edge cases, and anomalies that might trigger fraud alerts. This method lacks variability, meaning it does not test different fraud scores, transaction types, locations, or amounts, leaving significant gaps in coverage.

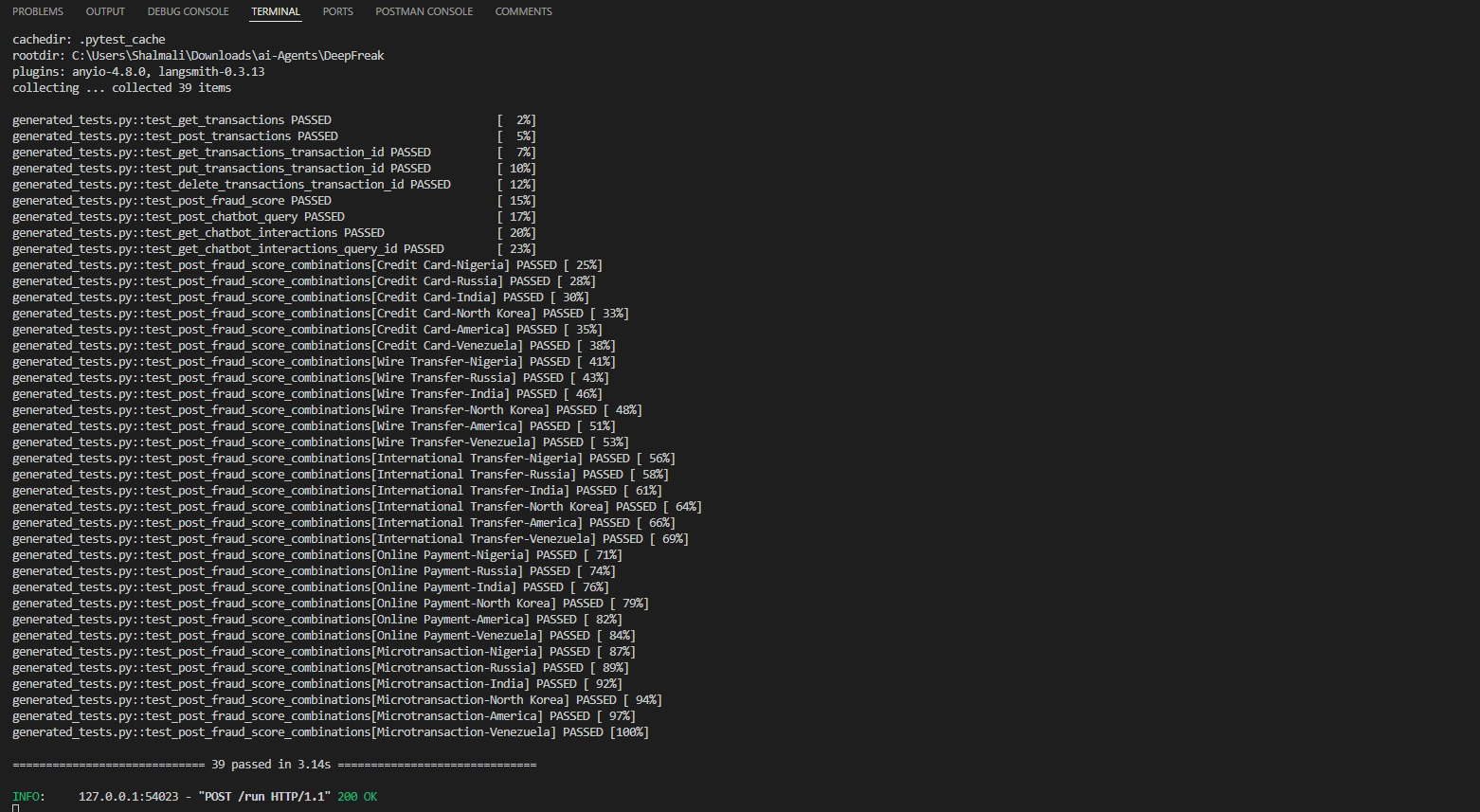
**Our Approach**



Locations: ["Nigeria", "Russia", "India", "North Korea", "America", "Venezuela"]  
Transaction Types: ["Credit Card", "Wire Transfer", "International Transfer", "Online Payment", "Microtransaction"]

Using pytest.mark.parametrize, we dynamically generate test cases for **each possible combination** of these values:

**Results**:

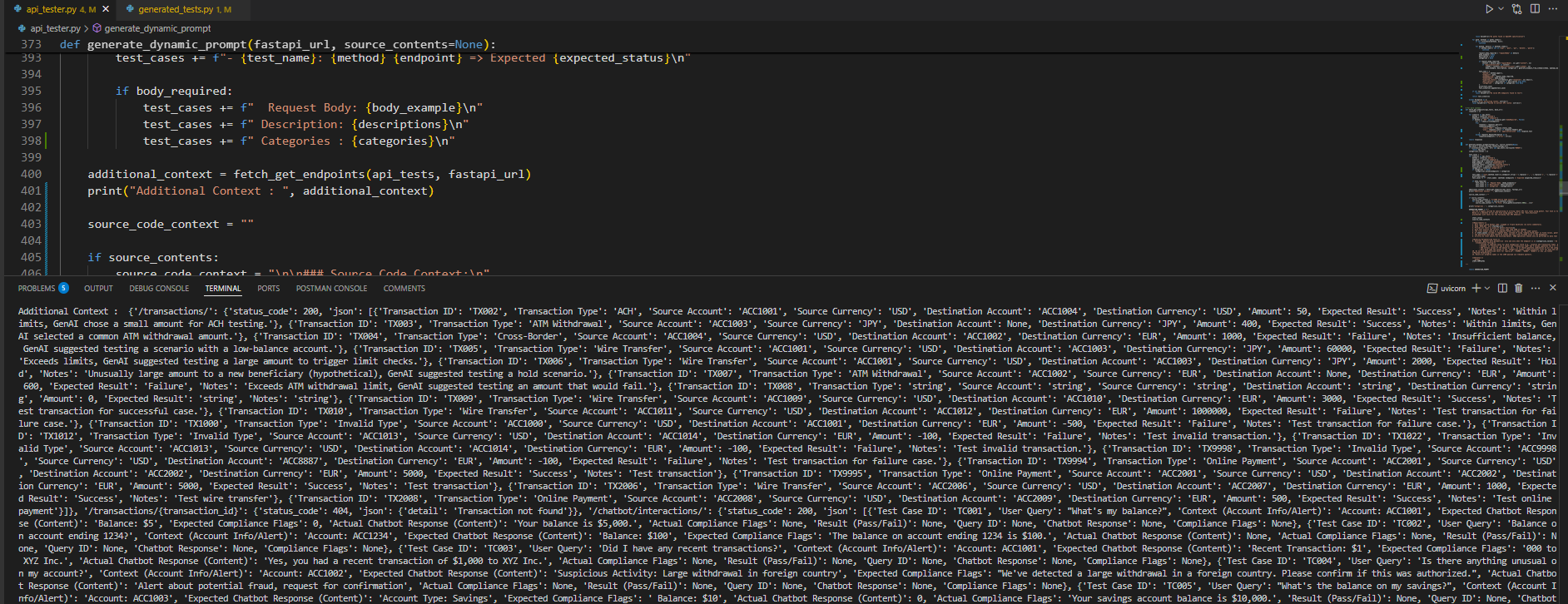


The test suite verifies multiple **transaction scenarios** across different **fraud-prone locations**. The **combinatorial approach** ensures broader test coverage.

**All tests passed**, meaning the system handled all input variations as expected.

Also as an additional context to the LLM we fetch all the information we can!

We extract the data from all /GET endpoints first and feed it to the LLM.



Our approach surpasses traditional methods by dynamically extracting real-time data from /GET endpoints, ensuring that test cases reflect the latest system behavior and edge cases. Unlike static test case generation, which relies on predefined inputs, our method leverages live transaction details, user interactions, and error responses to create highly relevant and adaptive test scenarios. This enhances test accuracy, improves coverage of unexpected edge cases, and reduces manual effort in maintaining test suites.

**Experimental Results**

Coverage Analysis:

API Category Endpoints Auto-Coverage

Transactions 5 100%

Fraud Detection 2 100%