**Test Plan Group 1**

**Currency Conversion Application**

**UMGC CMSC 495 Section 6380**

Elizabeth Bloss

Jackson Perry

Carl Blocker

Jonah Kiplimo

# **CMSC495 Currency Converter – Comprehensive Test Plan**

## 1. Objective and Scope

This test plan is designed to ensure complete coverage of the CMSC495 Flask-based currency converter application. The system is composed of a web-based user interface, a backend built on Flask, and supporting services including a database (PostgreSQL) and external API calls for currency rates.

### 1.1 Testing Objectives

* Validate correctness of isolated components (**Unit Testing**)
* Ensure functional integration of system components (**Integration Testing**)
* Verify user workflows behave as expected (**Functional Testing**)
* Prepare for real-world use through load and boundary condition testing (**Parametric Testing**)

### 1.2 Out of Scope

* Performance and stress testing with load simulation tools (e.g., Locust) is out of scope for this test plan.

## 2. Resources

* **Testing Framework:** Pytest and pytest-cov
* **Test Runners:** Local machine, GitHub Actions CI
* **Test Environment:**
  + Python 3.12 virtual environment
  + PostgreSQL test database or SQLite in-memory for unit/integration
  + Flask in test mode
* **Dependencies:** Listed in requirements.txt

## 3. Test Environment Configuration

1. Install dependencies:

pip install -r requirements.txt

1. Set environment variables for .env.test:

FLASK\_ENV=test

DATABASE\_URL=sqlite:///test.db

APP\_ID=fake\_app\_id

1. Run all tests with coverage:

pytest --cov=. tests/

## 4. Test Case Inventory

### 4.1 Unit Tests

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Purpose | Input | Expected Output |
| UT01 | Validate logging of events | valid form + route/IP | VisitorLog in DB |
| UT02 | Log event with missing fields | event\_type only | TypeError |
| UT03 | Handle long user-agent input | UA > 255 chars | Truncated or accepted |
| UT04 | Check IP and route storage | IP + route | Log with correct data |
| UT05 | Calculate USD→EUR correctly | base=USD, target=EUR, 100 | 90.0 (mocked rate) |
| UT06 | Unsupported currency handling | base=USD, target=XXX | KeyError or ValueError |

### 4.2 Integration Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Purpose | Steps | Input | Expected Output |
| IT01 | Load home page | GET / | N/A | 200 OK, contains form |
| IT02 | Valid conversion request | POST | base=USD, target=EUR, amt=100 | Result in response |
| IT03 | Invalid currency request | POST | base=XXX, target=YYY | Error message rendered |
| IT04 | Log 12 queries, show 10 | POST×12 + GET /history | Any values | Shows last 10 entries only |
| IT05 | History page empty | GET /history | N/A | Message for no history |

### 4.3 Parametric Tests

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Purpose | Parameter Range | Expected Behavior |
| PT01 | Conversion for edge amounts | 0, -1, 1e6 | Handle gracefully |
| PT02 | All supported currencies | List from DB | Acceptable or rejected |

### 4.4 Functional Tests

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Purpose | Steps | Expected Behavior |
| FT001 | Convert USD to EUR via UI | Load home → select base\_country = US, target = FR → input 100 → submit | Converted amount shown, flags rendered |
| FT002 | Flag preview updates correctly | Select base\_country = JP in dropdown | Japan flag + JPY preview injected via HTMX |
| FT003 | Invalid submission handling | Leave amount blank, or type 'abc' | Error message displayed |
| FT004 | Currency retained on validation err | Input non-numeric, click submit → check if selections are preserved | Fields stay populated |
| FT005 | History shows previous conversions | Perform 3 conversions → click History button | Both results shown in list |

## 5. Test Data

Sample inputs:

* Currencies: USD, EUR, GBP, JPY, CAD
* Amounts: 100, 0, -5, 1000000, "abc"
* Countries: US, DE, JP

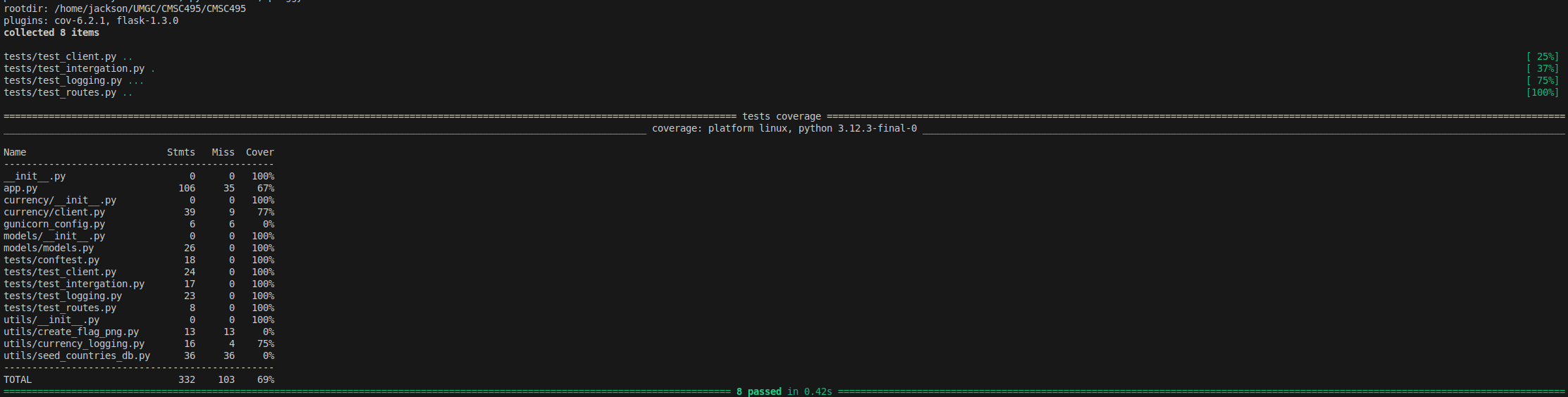
Mocked exchange rates used for unit/integration:

{"rates": {"USD": 1.0, "EUR": 0.9, "JPY": 140.0}}

## 6. Risk Mitigation and Next Steps

* Ensure API requests are mocked during unit and integration to avoid throttling or outages.
* Expand parametric tests to validate behavior under numeric edge cases.
* Consider automating functional tests with Playwright or Selenium if deploying publicly.

## 7. Coverage Status (15 Jun 2025)



This test plan demonstrates a comprehensive approach to validation of the currency converter system with layered test types and execution procedures. It is meant to ensure functional correctness, graceful error handling, and user reliability. The goal will be to increase test coverage over then next two weeks and introduce the integration tests.