# 1. Test Data Upload & Parsing

Test Objective: Ensure the dataset file is uploaded and parsed correctly.

import pandas as pd  
  
def test\_data\_upload\_parsing():  
 # Load the dataset  
 df = pd.read\_csv('Exchange\_Rate\_Final-2013-2024.csv')  
   
 # Check if the Date column is formatted correctly and non-null  
 assert pd.to\_datetime(df['Date'], format='%Y-%m-%d', errors='coerce').notnull().all()  
   
 # Check if all currency columns exist and contain numeric data  
 assert df.select\_dtypes(include=['float', 'int']).shape[1] > 0

# 2. Test Currency and Time Period Selection

Test Objective: Verify if the backend responds correctly to currency and time duration input.

from fastapi.testclient import TestClient  
from app import app # Your FastAPI app  
  
client = TestClient(app)  
  
def test\_currency\_time\_period\_selection():  
 response = client.get('/api/exchange\_rates', params={'currency': 'USD/EUR', 'period': 'monthly'})  
 assert response.status\_code == 200  
 data = response.json()  
   
 # Check if data contains correct time period and currency pair  
 assert 'USD/EUR' in data['currency']  
 assert data['time\_period'] == 'monthly'

# 3. Test MongoDB Data Fetching

Test Objective: Ensure the backend fetches correct data from MongoDB.

from pymongo import MongoClient  
  
def test\_mongodb\_fetching():  
 client = MongoClient('mongodb://localhost:27017')  
 db = client['exchange\_rates\_db']  
 collection = db['rates']  
  
 query = {'currency': 'USD/GBP', 'date': {'$gte': '2023-01-01', '$lte': '2023-12-31'}}  
 result = collection.find(query)  
   
 # Check if data is returned for the given period  
 assert result.count() > 0

# 4. Test Chart Data Display

Test Objective: Check that the fetched data displays correctly in the frontend (mocking data for test).

import matplotlib.pyplot as plt  
  
def test\_chart\_data\_display():  
 # Mock data for testing  
 dates = ['2023-01-01', '2023-02-01', '2023-03-01']  
 rates = [1.1, 1.2, 1.15]  
  
 plt.plot(dates, rates)  
 plt.title('USD/EUR Exchange Rate')  
 plt.xlabel('Date')  
 plt.ylabel('Rate')  
   
 # Check if the plot is correctly formed  
 assert len(plt.gca().lines) == 1  
 plt.close()

# 5. Test High and Low Points Display

Test Objective: Ensure the highest and lowest rates are identified correctly.

def test\_highest\_lowest\_points():  
 # Mock data  
 rates = [1.10, 1.25, 1.15, 1.30, 1.05]  
   
 # Test highest and lowest  
 highest = max(rates)  
 lowest = min(rates)  
   
 assert highest == 1.30  
 assert lowest == 1.05

# 6. Test Print Functionality (Simulated)

Test Objective: Simulate printing functionality (e.g., saving a chart as PDF).

def test\_print\_chart\_functionality():  
 # Mock chart  
 plt.plot([1, 2, 3], [1.1, 1.2, 1.3])  
   
 # Save chart as a PDF (simulating print)  
 plt.savefig('/tmp/chart.pdf')  
   
 # Check if file exists and is non-empty  
 import os  
 assert os.path.exists('/tmp/chart.pdf')  
 assert os.path.getsize('/tmp/chart.pdf') > 0  
 plt.close()

# 7. Test Custom Currency Basket Calculation

Test Objective: Ensure custom currency basket is calculated correctly.

def test\_custom\_currency\_basket():  
 # Mock rates and weights  
 rates = {'USD/INR': 74.50, 'USD/GBP': 0.85}  
 weights = {'USD/INR': 0.6, 'USD/GBP': 0.4}  
   
 # Calculate weighted average  
 basket\_value = sum(rates[currency] \* weight for currency, weight in weights.items())  
   
 # Expected result  
 expected\_basket\_value = 74.50 \* 0.6 + 0.85 \* 0.4  
 assert basket\_value == expected\_basket\_value

# 8. Test Risk Indicator Calculation

Test Objective: Ensure the risk indicator correctly calculates volatility.

import numpy as np  
  
def test\_risk\_indicator():  
 # Mock data for exchange rates  
 rates = [1.1, 1.2, 1.15, 1.3, 1.05]  
   
 # Calculate standard deviation for volatility  
 std\_dev = np.std(rates)  
   
 # Set thresholds for risk indicator  
 if std\_dev < 0.05:  
 risk\_level = 'low'  
 elif std\_dev < 0.10:  
 risk\_level = 'medium'  
 else:  
 risk\_level = 'high'  
   
 assert risk\_level == 'medium' # Based on the mock data

# 9. Test Handling of Missing Data

Test Objective: Ensure missing data is filled correctly using interpolation.

import pandas as pd  
  
def test\_missing\_data\_handling():  
 # Mock data with missing values  
 data = {'Date': ['2023-01-01', '2023-02-01', '2023-03-01'],  
 'Rate': [1.1, None, 1.3]}  
 df = pd.DataFrame(data)  
   
 # Fill missing values with interpolation  
 df['Rate'] = df['Rate'].interpolate()  
   
 # Check if missing value is filled correctly  
 assert df['Rate'].isnull().sum() == 0  
 assert df['Rate'][1] == 1.2 # Interpolated value between 1.1 and 1.3

# 10. Test Dynamic UI Updates (Backend Response)

Test Objective: Verify that backend API responses are correctly updated on currency change.

def test\_dynamic\_ui\_update():  
 response = client.get('/api/exchange\_rates', params={'currency': 'USD/INR'})  
 data\_usd\_inr = response.json()  
  
 response = client.get('/api/exchange\_rates', params={'currency': 'USD/GBP'})  
 data\_usd\_gbp = response.json()  
  
 # Check if the API returns different data for different currencies  
 assert data\_usd\_inr != data\_usd\_gbp

# 11. Test API Response Time

Test Objective: Ensure that the backend API response time is within limits.

import time  
  
def test\_api\_response\_time():  
 start\_time = time.time()  
   
 response = client.get('/api/exchange\_rates', params={'currency': 'USD/GBP', 'period': 'yearly'})  
 assert response.status\_code == 200  
   
 end\_time = time.time()  
   
 # Check if response time is under 2 seconds  
 assert end\_time - start\_time < 2