Test Plan for Tocos API 📝

A t**est plan** outlines the approach, objectives, and scope of testing for a specific project or system. The following Test Plan focuses on testing the API endpoints (/transactions, /buy, and /sell) and includes the following details:

✍️ Test Objectives

- Ensure the API endpoints (/transactions, /buy, and /sell) function correctly.
- Validate both success and error scenarios.
- Test transaction limits and security considerations.
- Ensure the transaction integrity and consistency..

🔮 Test Scope

This test plan covers three critical API endpoints:

- **POST /buy**: Buying Tocos using fiat currency.
- **POST /sell**: Selling Tocos for fiat currency.
- **POST /transactions**: Transferring Tocos between users.
- **GET /transactions/{userId}**: Retrieving a user's transaction history.

🔭 Testing Approach

Manual Testing:

- Initially, use Postman to manually test API endpoints for quick checks and understanding of API behavior.

Automated Testing:

- Automated testing using Playwright and TypeScript.
- Create mock data for users, transactions, Tocos balances, and fiat currency.
- Simulate various scenarios including edge cases and error handling.

🛠️ Resources Needed

- **Tools**:
  - Postman for manual API exploration and testing.
  - Visual Studio Code as an IDE for writing TypeScript scripts.
  - Playwright Test Runner for automation.
  - A Git repository for version control.
- **Environment**:
  - Local development environment or a dedicated testing environment that mirrors production.
- **Data**:
  - Mock data for users, Toco balances, fiat currency amounts, and transaction histories.

🔍 Test Cases (to be executed)

1. **POST /buy Endpoint**:
   a. **Buy Tocos Success**:
      - **Objective**: Ensure users can successfully buy Tocos with sufficient fiat currency.
      - **Steps**: Create a user with a fiat currency balance, perform a /buy operation within the balance limit, and verify the user's Toco and fiat balances are correctly updated.
      - **Expected Outcome**: The transaction completes successfully, Tocos are added to the user's account, and the fiat balance is deducted accordingly.
   b. **Buy Tocos With Insufficient Fiat Currency**:
      - **Objective**: Test the system's handling of buy attempts exceeding the user's fiat currency balance.
      - **Steps**: Attempt a /buy operation that exceeds the user's available fiat currency.

- **Expected Outcome**: The API returns an error indicating insufficient funds, and the user's balances remain unchanged.

c. **Buy Tocos Daily Limit Exceeded**:

- **Objective**: Verify the API enforces daily Toco purchase limits.
- **Steps**: Execute multiple /buy operations in quick succession to surpass the daily Toco purchase limit.
- **Expected Outcome**: The API returns an error after the limit is exceeded, preventing further transactions.

2. **POST /sell Endpoint**:

a. **Sell Tocos Success**:

- **Objective**: Confirm users can sell Tocos for fiat currency within their Toco balance.
- **Steps**: Perform a /sell operation with Tocos within the user's balance and verify both Toco and fiat balances are updated accurately.
- **Expected Outcome**: The transaction completes successfully, reducing the Toco balance and increasing fiat currency.

b. **Attempt to Sell More Tocos Than Owned**:

- **Objective**: Test the API's response to selling more Tocos than the user owns.
- **Steps**: Attempt a /sell operation that exceeds the user's Toco balance.
- **Expected Outcome**: The API returns an error for insufficient Toco balance, and no changes are made to the user's account.

c. **Sell Tocos Beyond Daily Limit**:

- **Objective**: Ensure daily limits for Toco sales are enforced.
- **Steps**: Conduct /sell operations to exceed the user's daily Toco sale limit.
- **Expected Outcome**: The API restricts transactions beyond the daily limit, signaling an error for subsequent sell attempts.

3. **POST /transactions Endpoint**:

a. **Successful Toco Transaction Between Users**:

- **Objective**: Verify a user can transfer Tocos to another user within their balance limits.
- **Steps**: Initiate a /transactions operation transferring Tocos from one user to another and validate both users' Toco balances post-transaction.
- **Expected Outcome**: The transaction is processed successfully, deducting Tocos from the sender's account and crediting them to the recipient's account.

b. **Unauthorized Transaction Attempt**:

- **Objective**: Examine the API's security measures against unauthorized transaction attempts.
- **Steps**: Attempt a /transactions operation without proper authentication or on behalf of another user without authorization.
- **Expected Outcome**: The API denies the request, returning an error message indicating the lack of permissions or authentication failure.

c. **Insufficient Toco Balance for Transaction**:

- **Objective**: Test the transaction failure scenario due to insufficient Toco balance.
- **Steps**: Try to transfer more Tocos from a user's account than available.
- **Expected Outcome**: The API should return an error message about insufficient Toco balance, preventing the transaction.

4. **GET /transactions/{userId} Endpoint**:

a. **Retrieve Transaction History Success**:

- **Objective**: Ensure that a user can successfully retrieve their transaction history.
- **Steps**:
  1. Create or use an existing user with a known userId.
  2. Perform several transactions (buy, sell, transfer) to populate the user's transaction history.
  3. Send a GET request to /transactions/{userId} using the userId.
- **Expected Outcome**: The API returns a 200 OK status with a JSON payload containing the user's transaction history. Each transaction

record should accurately reflect the transactions performed in step 2.

b. **Unauthorized Access to Transaction History**:
   - **Objective**: Confirm that users cannot access the transaction history of other users without proper authorization.
   - **Steps**:
     1. Create two users, User A and User B, each with their own transaction history.
     2. Attempt to retrieve User B's transaction history using User A's credentials or without authentication.
   - **Expected Outcome**: The API returns a 401 Unauthorized or 403 Forbidden status, preventing unauthorized access to another user's transaction history.

c. **Transaction History of a New User**:
   - **Objective**: Verify that the API correctly handles requests for transaction histories of users with no transactions.
   - **Steps**:
     1. Create a new user with no transaction history.
     2. Send a GET request to /transactions/{userId} using the new user's userId.
   - **Expected Outcome**: The API returns a 200 OK status with an empty array or a relevant message indicating that no transaction history is available for the user.

🔐 Security Considerations:

- Implement authorization tests to ensure only authenticated users can initiate transactions.
- Test for SQL Injection vulnerabilities in input fields.
- Validate input sanitization to prevent Cross-Site Scripting (XSS).

🏃‍♂️Performance Considerations:

- Conduct load testing to assess API response times under heavy use.

- Test for endpoint scalability by simulating concurrent transactions.

📉 Reporting:

- Document test cases, results, and anomalies in a test management tool.
- Use Git for code and document version control, ensuring traceability and collaboration.

🪛 Review and Adjust:

- Regularly review test results and adjust test cases as needed based on findings and evolving requirements.