Mocking Strategy Documentation

Mocking Strategy for Testing Environment

To ensure reliable, isolated, and repeatable tests for our Top-Up API functionality, we utilize a mocking strategy. This enables us to simulate various external service behaviors and control edge cases without relying on actual production systems.

1. Mocking Tool Used: MockServer MockServer is an open-source tool that allows us to mock HTTP and HTTPS requests. We configure it in our test environment to simulate backend services such as user balance retrieval, top-up confirmation, etc.

2. How Mocking Works in Our Tests - When `useMock = true`, the test suite routes requests to a local MockServer instance instead of hitting the real backend. - The MockServer is pre-configured with expectations and responses for specific endpoints. - For example, the `/topup` endpoint returns a fixed successful response including a new balance and message.

3. When Real API is Used - When `useMock = false`, the test suite makes actual requests to the testing environment's backend. - The test checks real logic flow including database updates and business rule validations.

4. Reflection on User Balance - **Mock Mode**: The balance is a stubbed value defined in the test or in MockServer configuration. It does not reflect actual changes in a persistent store. - **Real API Mode**: The balance is retrieved and updated in the testing environment database, providing an end-to-end validation of the top-up process.

5. Benefits of This Strategy - Enables testing of both success and failure cases easily. - Allows isolated testing without reliance on external systems. - Supports CI/CD pipelines with fast, reliable test runs.

6. Example In `createValidTopUp_ShouldSucceed()`: - If `useMock = true`, we simulate the balance and return a fixed success message. - If `useMock = false`, the test uses actual logic to validate balance changes in the testing environment.