Mocking things in Async Unit Tests

Orion Edwards

October 2022

Background - Kinds of test

- End to End Test
  - [Approximately] All parts of the system are running in their ‘production’ state and interacting as they would in a real environment.
- Integration Test
  - Multiple parts of the system are running in a test host environment and may or may not interact as they would in a real environment.
- Unit Test
  - [Approximately] One piece of code is running in a test host environment and does not [usually] interact with other components.

These definitions are fuzzy and many tests don’t fit cleanly into any one of those buckets.

Background - Kinds of test

Unit Test

- FAST + ISOLATED
- Doesn’t talk to the outside world. No sockets, files, databases, etc
- Nothing moves unless you tell it to

E2E Test

- SLOW + NOT ISOLATED
- Has real databases, network calls, etc
- Things may move in the background of their own accord

Integration Test

- “When *I* use a word… it means just what I choose it to mean – neither more nor less.”

Background - Mocks and Stubs

Common Wisdom

- Stub
  - Placeholder for an interface/object/etc
  - No logic
- Mock
  - “Mocks verify the behaviour of the code you’re testing”
  - Most articles online suggest that a mock is a “call recorder” and then you assert that X methods were called with Y parameters.

Background - Mocks and Stubs

More practically speaking, everyone just refers to everything as “Mocks”

- Stub/Fake
  - Returns fixed values or throws exceptions

Good! Simple to work with, but limited and don’t scale

- Mock
  - Returns conditional values
  - Call Recorder / Verifier

More powerful than a stub, cheaper than a simulator, but they are brittle and difficult to work with. I do not like these

- Simulator
  - Contains actual logic and internal state

Good! Very powerful and can have spinoff benefits, but are a lot of work

What am I going to show you today?

A useful alternative to a traditional mock. Works best with async code.

This is very narrow set of criteria, but turns out there’s a lot of code which fits it.

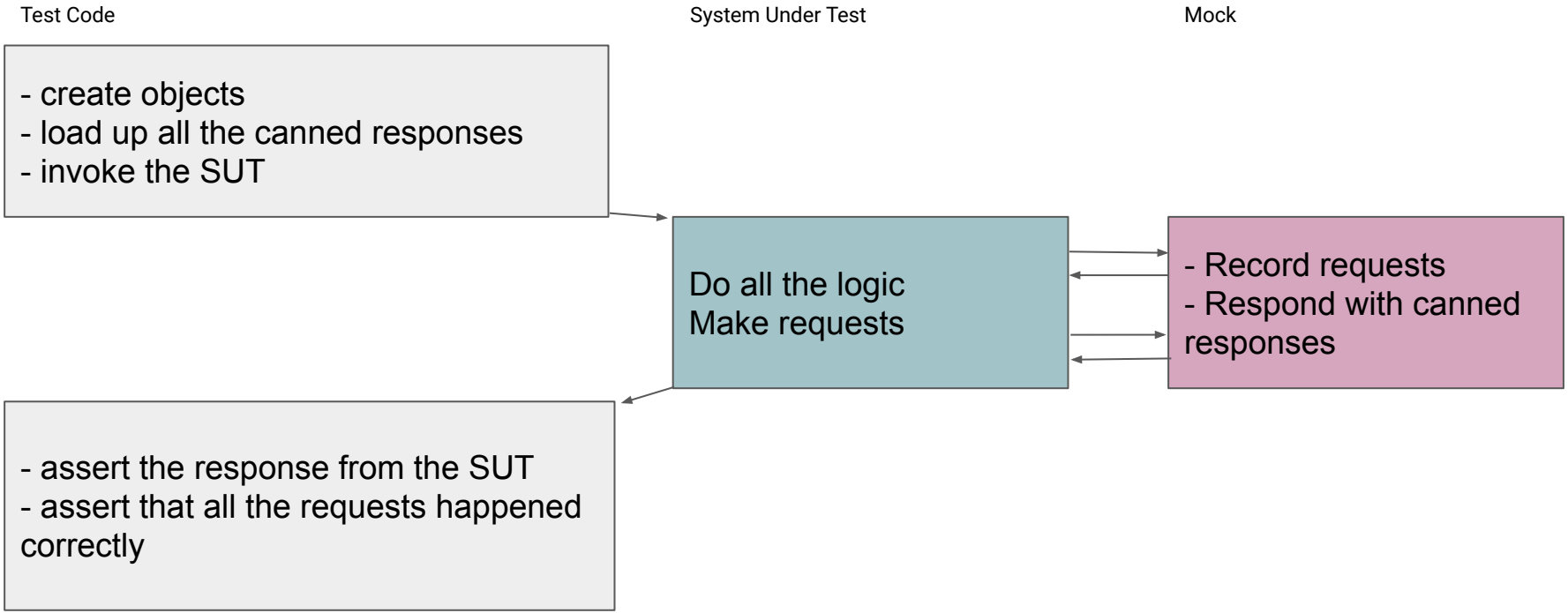
Use-case: Talking to someone else's network server

Contrived Scenario: A basic item-sync protocol

POST `/api/session` { "username", "password" } => 201
GET `/api/items` => 200 [`"Item1.txt"`, …]
POST `/api/items?name=Item1.txt` [binary] => 200

Switch to demo here and show baseline + traditional approaches

Traditional Http Mock - Control Flow

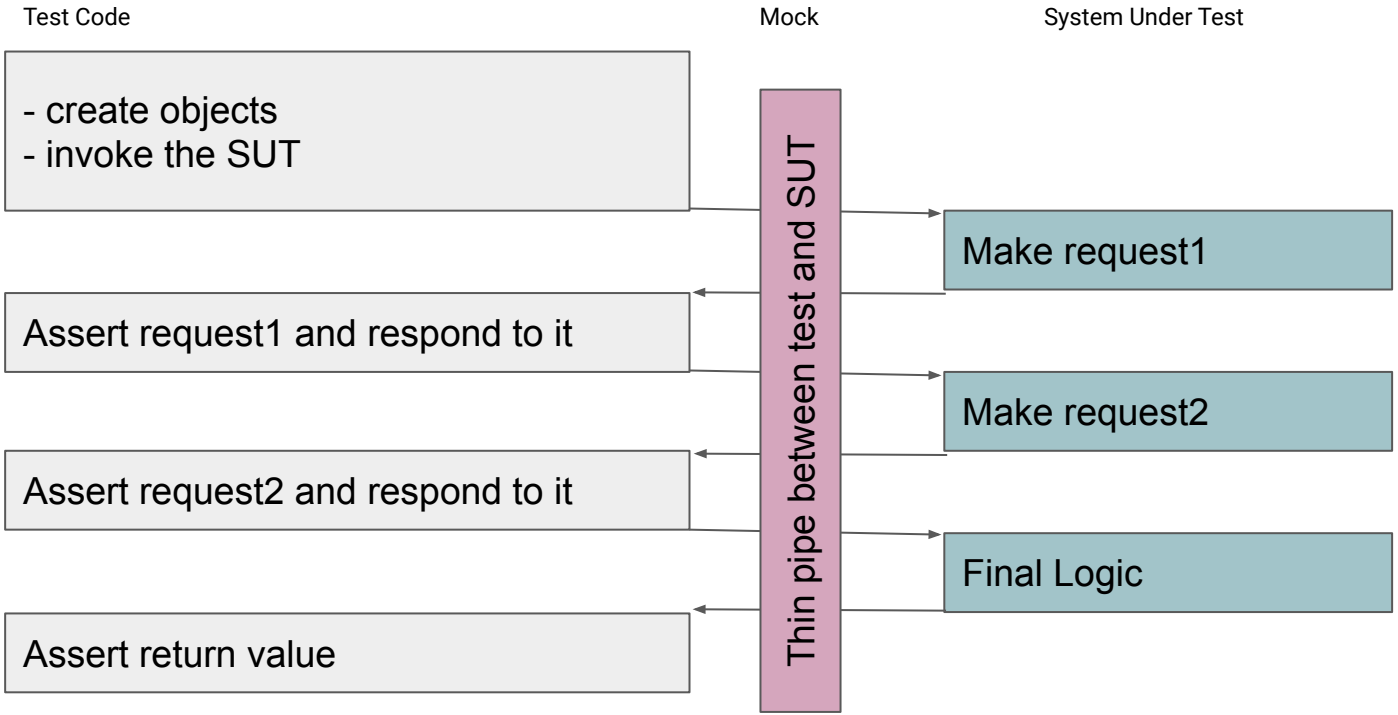


Problems with traditional mocks

- **Large batch size**: We make N requests, then we look at them all at the end
- **Hard to hold in your head:** Stuff you have to think about is spread across all of Test, SUT, and Mock
- Request order and double-calls are usually not tracked
- “Request Matching” logic can end up being complex and brittle

Switch to demo here and show new approach

Async Mock - Control Flow



Why is this approach worse?

- Order-significance can be annoying
  - Reordering requests in your code will break the tests
  - People may assume order is not significant and be confused
  - Doesn't handle requests getting made in nondeterministic order

- It can deadlock if the number of requests doesn't line up on either side
  (can be resolved with some timeouts in the Mock)

- The async plumbing in the mock is fiddly.
  Generally it's "write once then ignore" style framework code, but if you do have to touch it, it can be tricky

Why is this approach better?

- Easier to “hold in your head”, The test itself is much more readable
  All the things you care about are right there in the test, not somewhere else
- Easier to reason about due to smaller batch size at each point.

- More debuggable - you can set breakpoints in the middle of the flow, rather than waiting for it to finish and picking up the pieces

- Asserting that requests are a) in the right order, b) aren’t duplicated, c) you’ve handled them all, comes for free.
- Gets rid of “request matching” logic in mocks

Done!

Questions?

Remember, this doesn't just apply to HTTP.
Anywhere you have a request/response flow where there is more than one request, it can be valuable.