Rails encourages developers to use a database in integrated acceptance tests, and we've gone with the flow on that front.

However, like many rails apps we also integrate with a number of external services that are more difficult to deal with in tests - stripe , campaign monitor , akismet and slack are good examples. In most cases these services are accessed over the internet and operated by external companies, so to keep our tests fast and reliable we've opted to stub them rather than change any remote state.

To begin with, we were inconsistent with where and how we stubbed these services, exploring solutions like rspec mocks , vcr and plain webmock .

As our product grew and we integrated new services, we started to look for a pattern we could standardise on. Our goals for the new pattern were:

- Avoid all network activity to external services during tests
- Encourage happy path acceptance tests to avoid excessive build times
- Rely on unit tests to thoroughly test code that calls the external service, including error cases and intermittent network issues
- Avoid excessive setup for acceptance tests
- If we're using an external gem, encapsulate it with code we own to simplify future refactoring

## Gateway Objects

We settled on a pattern we called "Gateway Objects", inspired by a Martin Fowler refactoring article .

We store them in app/gateways/ and have added the following line to config/application.rb so rails can find them:

```
config.autoload_paths<<Rails.root.join("gateways").to_s
```

In most cases we use a fraction of the external service's functionality, so our gateway classes have public methods for just the bits we need. Each method has extensive unit specs, usually relying on webmock to avoid hitting a live HTTP API.

Where possible, our gateways return objects that are either from ruby core (Integers, Strings, Hashes, Arrays, etc) or immutable value objects that we control.

Here's a simplified example that adds subscribers to a Campaign Monitor list. It relies on the createsend gem, and uses webmock for unit specs.

```
require'createsend'classCampaignMonitorListGatewaydefinitialize(list_id,api_key: nil)@list_id=
list_id
@api_key
=
api_key
||
ENV
.
fetch
(
"CAMPAIGN_MONITOR_KEY"
)
end
# add a new email address to this list
#
def
subscribe
(
email
)
CreateSend
::
Subscriber
.
add
(
{
api_key:
@api_key
},
@list_id
```

,
email
,
# email address
email
,
# name
[],
# custom fields
false
# resubscribe
)
==
email
end
end

Exhaustive unit specs allow us to focus our integrated acceptance tests on a happy path , so we usually stub the gateway to return a consistent result across all tests with the some configuration in spec/rails_helper.rb:

RSpec.configuredo|config|config.before:each,type: :feature,stub_cm_list: truedoallow(CampaignMonitorListGateway
).
to
receive
(
:new
)
{
instance_double
(
CampaignMonitorListGateway
,
subscribe:
true
)
}
end
end

With that, any acceptance test can use metadata to declare a need for stubbing.

feature'A reader subscribing to our daily newsletter'doscenario"from a topbar link",:js,:stub_cm_list do#
some assertions

```
end
end
```

Most method calls on a gateway class will result in network activity and lots can go wrong when networks are involved. We avoid silently swallowing exceptions, and either:

- catch the exception, notify our error tracker , and return a Null object ; or
- allow the exception to bubble out so the caller deals with it

We're pretty happy with the final result – our test setup is cleaner, the tests are acceptably fast and pass without internet access, and the shared vocabulary helps us maintain team code in a consistent style over time.

Authors: James Healy, Technical Director, The Conversation

**Read more** http://theconversation.com/gateway-classes-a-pattern-for-interacting-with-external-services-65633