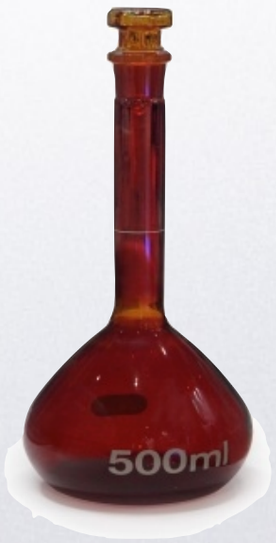
Rails Testing

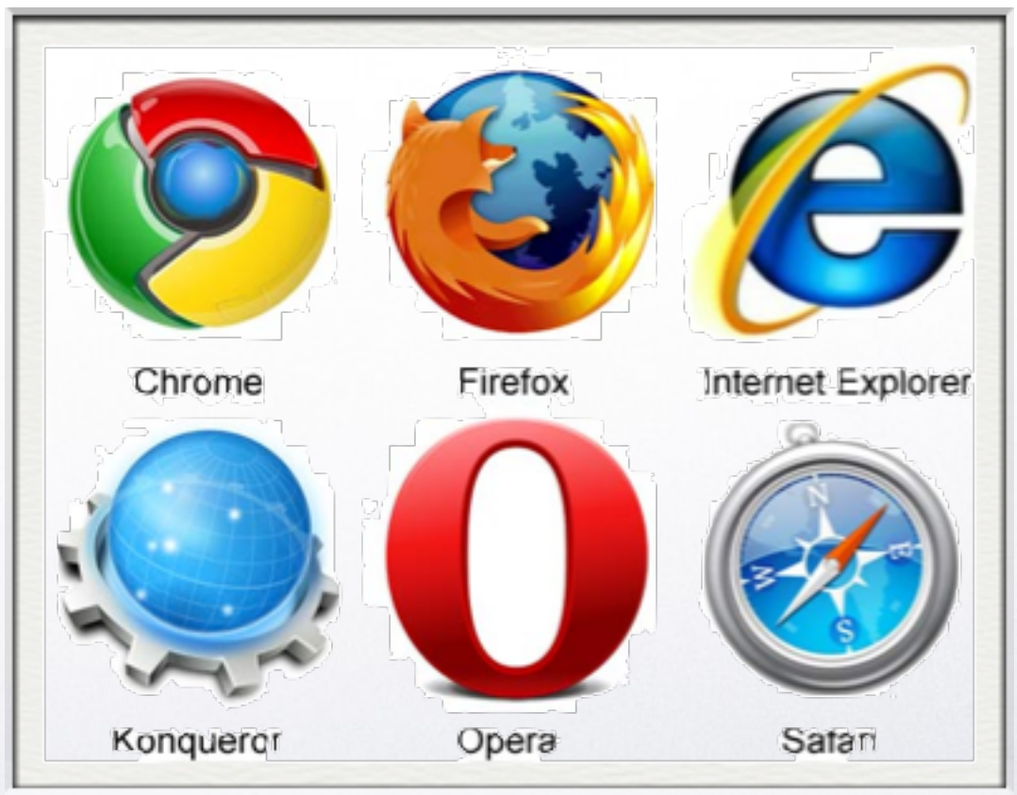
Ynon Perek
ynon@ynonperek.com
http://ynonperek.com



Agenda

- What do you test?
- Rails Test World
- Mocks - How to use / How to avoid

Why is it hard?



Types of Tests

Unit tests

System tests

Testing Goals

- We write acceptance tests to make sure a piece of code works
- We write unit tests to easily find bugs before they happen

Demo - Unit Test

```
describe 'cart#price' do
  it 'is the sum of all its products' do
    b1 = create(:product, price: 10)
    b2 = create(:product, price: 20)

    @cart << b1
    @cart << b2

    expect(@cart.price).to eq(30)
  end
end
```


Demo - System Test

```
it 'allows user to change password' do
  visit edit_user_registration_path
  fill_in 'user[password]', with: 'hello'
  fill_in 'user[password_confirmation]', with: 'hello'
  fill_in 'user[current_password]', with: '10203040'
  click_on I18n.t('users.edit-profile')

  @user.reload
  expect(@user.valid_password?('hello'))
end
```


Differences

```
describe 'cart#price' do
  it 'is the sum of all its
products' do
    b1 = create(:product,
price: 10)
    b2 = create(:product,
price: 20)

    @cart << b1
    @cart << b2

    expect(@cart.price).to eq()
  end
end
```

```
it 'allows user to change password' do
  visit edit_user_registration_path
  fill_in 'user[password]', with:
'hello'
  fill_in
'user[password_confirmation]', with:
'hello'
  fill_in 'user[current_password]',
with: '10203040'
  click_on I18n.t('users.edit-profile')

  @user.reload
  expect(@user.valid_password?
('hello'))
end
```

Now You:

- In your project, find one unit test. What's its scope?
- In your project, find one system test. What process does it verify?

Goals - Unit Tests

- Short feedback loop
- Find bugs before they happen
- Run in development on your box
- Run after each “save”
- Easy to maintain, safe to delete


Goals - System Tests

- Verify the process as realistically as possible
- OK to be slow
- Run automatically before deployment
- Maintenance is hard

What Should You Test?

- Past bugs
- Intended behaviour
- Edge cases

Balancing Tests

- Few system / scenario tests
- Many unit tests

Rails Testing Landscape

Agenda

- Testing frameworks: rspec / minitest
- Fixtures / Factories
- Model / Controller / System tests
- Code sharing between tests

Minitest

- Default rails testing framework
- Tons of extensions
- Uses XUnit syntax by default
- Feature rich
- Previously named Test::Unit

Minitest

```
require 'test_helper'

class CouponTest < ActiveSupport::TestCase
  include FactoryBot::Syntax::Methods

  setup do
    @teacher = create(:teacher)
    @school  = create(:school, owner: @teacher)
  end

  test 'coupons are invalid when remaining == 0' do
    c = create(:valid_coupon, remaining: 0, school: @school)
    assert(c.expired?)
  end
end
```

RSpec

- Behaviour Driven Development for Ruby
- Making BDD Productive and Fun


RSpec

```
require 'rails_helper'

RSpec.describe Tag, type: :model do
  context 'tag priority level' do
    it 'is marked important when name ends in !' do
      t = Tag.new(name: 'demo!')
      expect(t.important?).to be true
    end

    it 'is not important when name ends in other things' do
      t = Tag.new(name: 'demo')
      expect(t.important?).to be false
    end
  end
end
```

RSpec vs. Minitest

- It doesn't really matter…
- Both have the same features / capabilities

Q & A

Creating Data

Fixtures

- Fixed dataset created in yml files
- Saved into DB without validation
- Reset between tests

Fixtures RSpec Setup

- Create a directory spec/fixtures
- Add a line in rails_helper.rb:
  config.global_fixtures = :all

Fixtures File

```yaml
one:
  text: hello world
  priority: 3
  user: admin
```

Fixture Labels

```
require 'active_record/fixtures'
ActiveRecord::FixtureSet.identify(:admin)
 => 135138680
```

Factories

- Quick object creation
- Passed validation
- Needs to be explicitly created in tests
- gem ‘factory_bot’

Setup

- Add gem factory_bot_rails
- Setup rspec:
  config.include FactoryBot::Syntax::Methods

A First Factory

```
FactoryBot.define do
  sequence :email do |n|
    "person#{n}@example.com"
  end

  factory :user do
    email
    password { '10203040' }
  end
end
```


Using The Factory

```
require 'rails_helper'

RSpec.describe User, type: :model do
  describe '#tickets' do
    it 'should show all the tickets created by this user' do
      u = build(:user)
      u.tickets.build([
        { text: 'hello world' },
        { text: 'bye bye'}
      ])

      expect(u.tickets.size).to eq(2)
    end
  end
end
```

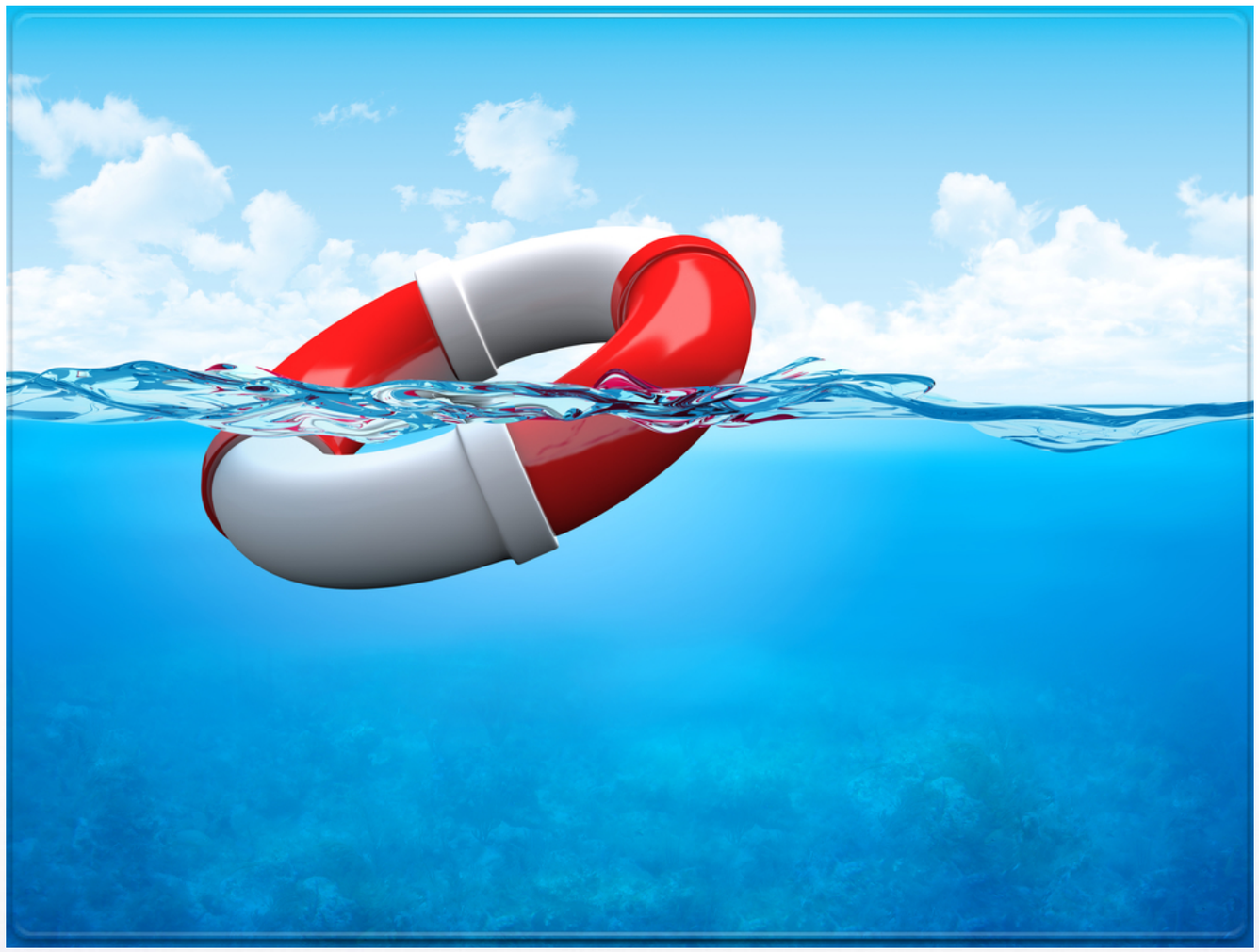
Better Factories

```
FactoryBot.define do
 factory :user do
    email
    password { '10203040' }

    factory :user_with_tickets do
      transient do
        tickets_count { 5 }
      end

      after(:build) do |user, evaluator|
        user.tickets = build_list(:ticket, evaluator.tickets_count, user: user)
      end
    end
  end
end
```

When To Use Each

- Use fixtures for “fixed” test data
- Use factories for dynamic data relevant for specific tests

Q & A



Model Tests

What To Test

- Business Logic (any method in the model)
- Custom or complex validations


What not to test

- Don't test THAT a validation or hook exists
- Don't test that AR methods work

Tip: skip DB

```
require 'rails_helper'

RSpec.describe Tag, type: :model do
  context 'tag priority level' do
    it 'is marked important when name ends in !' do
      t = Tag.new(name: 'demo!')
      expect(t.important?).to be true
    end

    it 'is not important when name ends in other things' do
      t = Tag.new(name: 'demo')
      expect(t.important?).to be false
    end
  end
end
```

Tip: Disable Hooks

```
context 'after save' do
  before do
    Tag.skip_callback(:save, :after, :sync_with_remote_api)
  end

  after do
    User.set_callback(:save, :after, :sync_with_remote_api)
  end

  it 'has #id' do
    t = Tag.create(name: 'demo')
    expect(t.id).not_to be_nil
  end
end
```

Tip: Count DB Queries

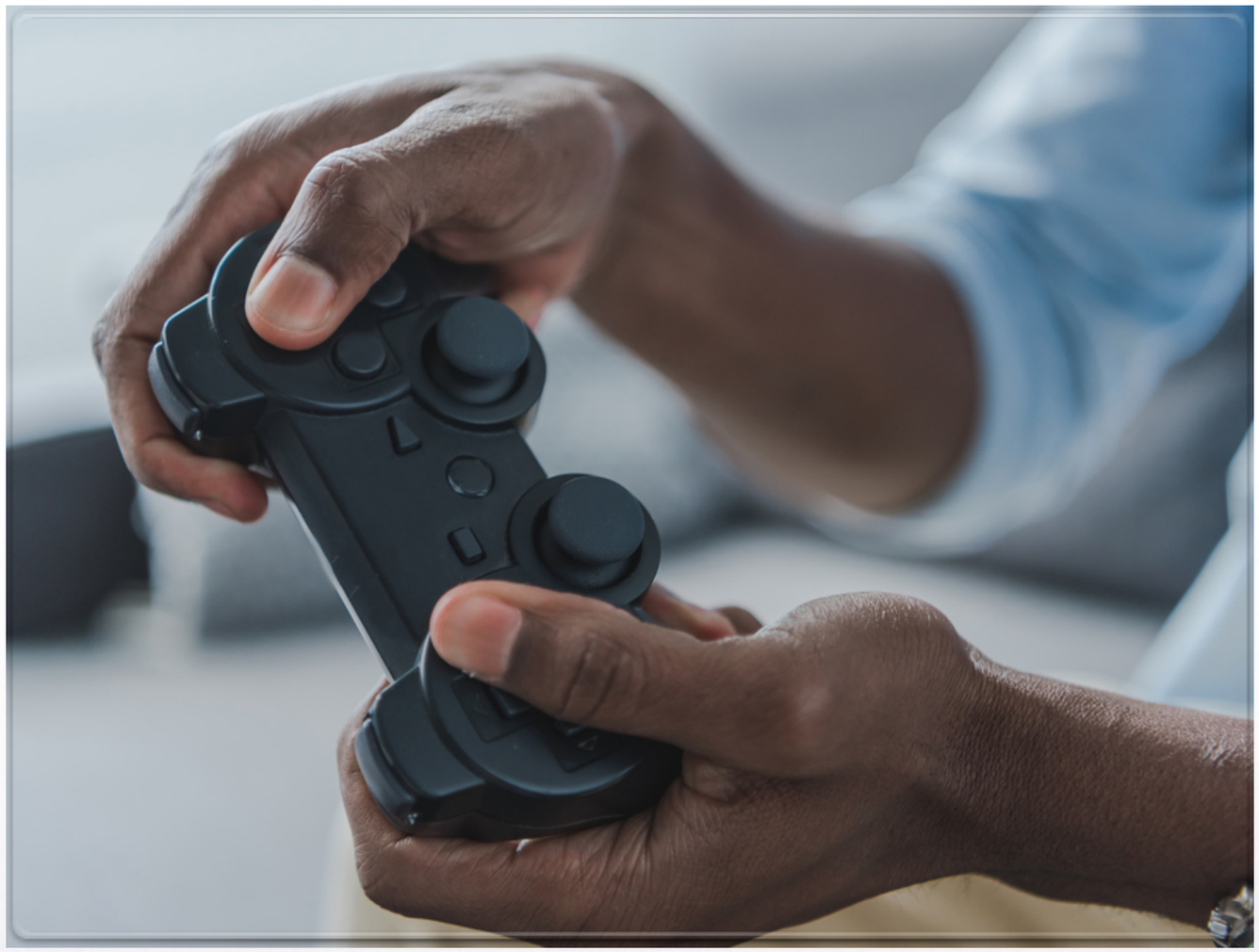
- Use gem db-query-matchers to count SQL queries in model calls or scopes
- https://github.com/civiccc/db-query-matchers

Tip: Count DB Queries

```
describe '.tagged_with' do
  it 'should perform only 1 query' do
    ticket = Ticket.new(text: 'hello')
    tag = ticket.tags.build(name: 'one')

    expect {
      Ticket.tagged_with(tag).load
    }.to make_database_queries(count: 1)
  end
end
```

Your Turn

- Write 3 model tests in your system
- Consider: what to test? what not to test?



Controller Tests

What

- Mainly to test:
- HTTP status codes
- Authorization
- JSON APIs

How

- gem 'rails-controller-testing'
- rails g rspec:controller <controller_name>

How

- For devise to work we need to add this line to spec/rails_helper.rb:

```
config.include Devise::Test::ControllerHelpers, type: :controller
```

Why

- Test controller code in isolation
- Relatively fast

Why Not

- Stuck in the middle
- Not as accurate as model tests
- Not as reliable as system tests

How

```
describe '#index' do
  it 'should return a list of tickets' do
    get :index
    expect(assigns(:tickets).size).to eq(tickets.length)
  end
end
```

How

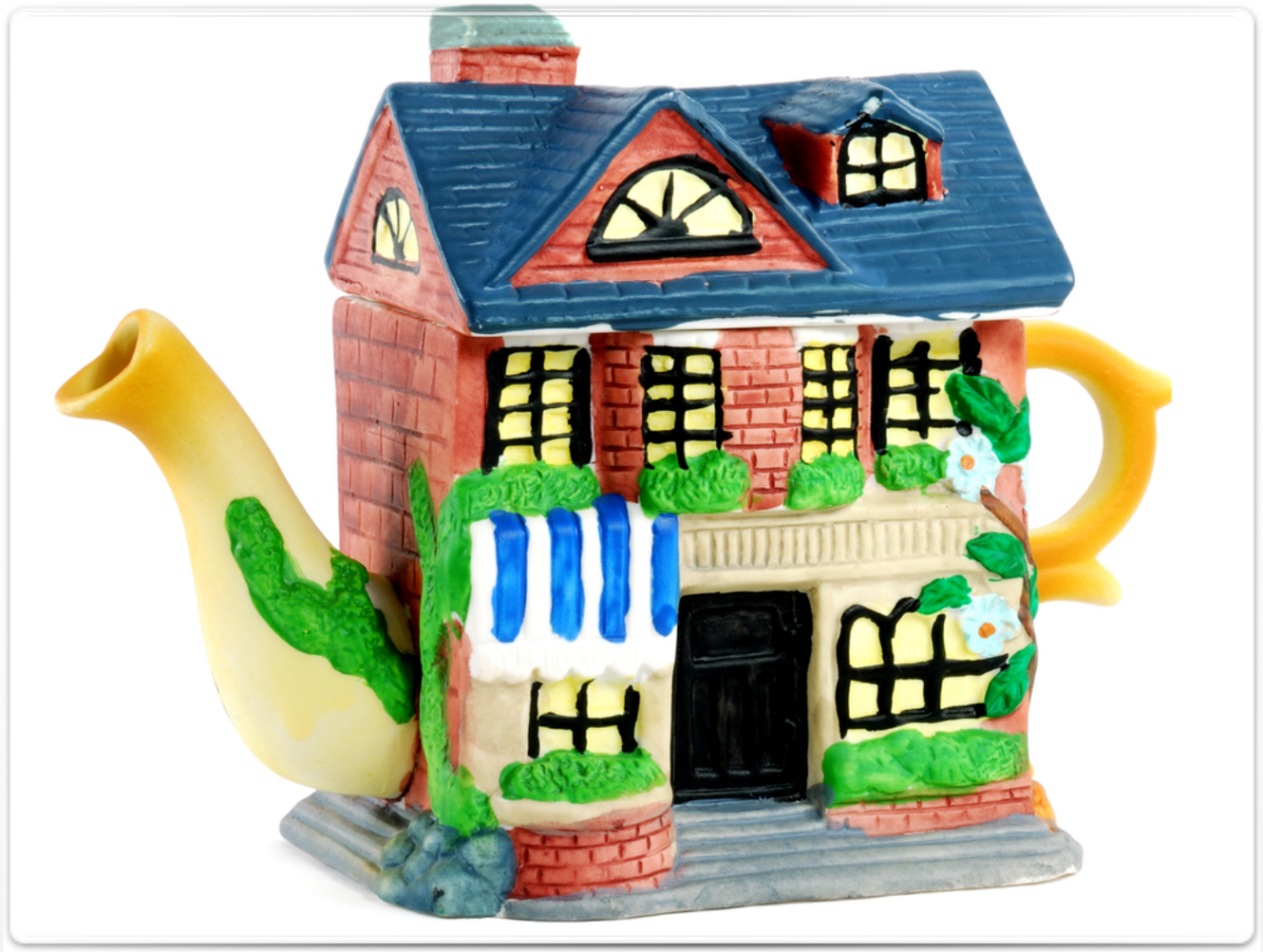
```
context 'with views' do
  render_views

  it 'should render a json of the result' do
    get :index, format: :json
    json_response = JSON.parse(response.body)
    expect(json_response.size).to eq(tickets.length)
  end
end
```

Tip

- Use browser to get the query

Tip

- Only render views if that's what you're testing (slow tests are the enemy)



System Tests

What

- Real browser, Fake data
- Most effective
- Slowest

How

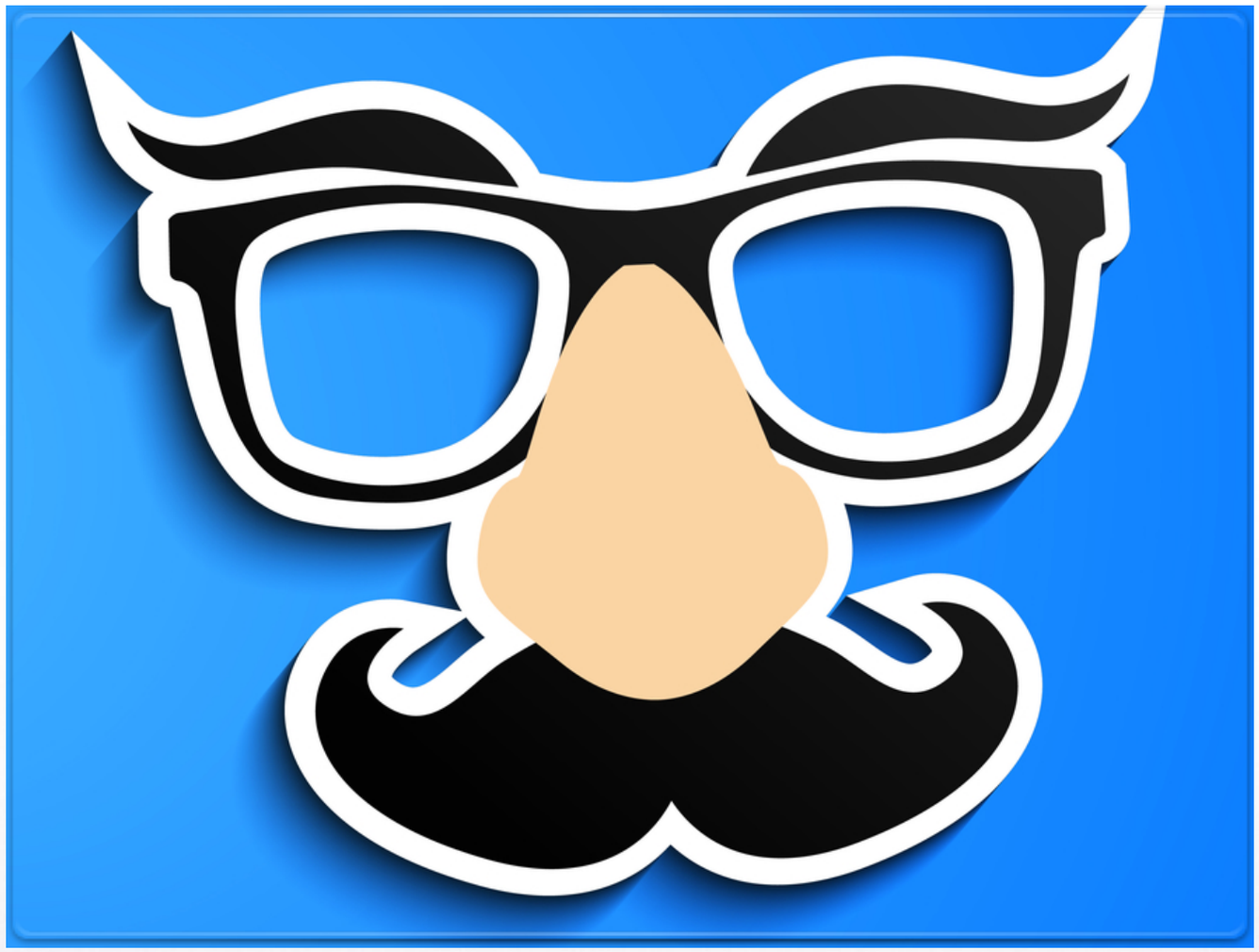
```
require "rails_helper"

RSpec.describe 'Tickets Management', :type => :system do
  before do
    driven_by :selenium, using: :chrome, screen_size: [1400, 1400]
  end

  it 'enables me to create widgets' do
    visit '/'
    expect(page).to have_selector('tbody tr', count: tickets.length)
  end
end
```


Why

- Easy to describe
- Uses fixtures / factories we already have
- Integrate server/client tests
- Provides “peace of mind”

Why Not

- Slow
- Test Fails -> Debugging Starts



Mocking

What

- Replace existing class/object with fake

Why

- Skip DB / Network calls in tests
- Test edge cases in external services
- Avoid unwanted side effects


How

```
RSpec.describe Tag, type: :model do
  context 'fake remote api' do
    it 'should not notify the remote api' do
      t = Tag.new
      allow(t).to receive(:sync_with_remote_api)

      expect {
        t.save
      }.to perform_under(1).sec
    end
  end
end
```

Why

- In system tests - when the real API doesn't provide "test mode"
- In unit tests - easy way out


Why Not

- System tests usually have test modes, which is more reliable
- Unit tests can be refactored

Refactoring

- Check existence of API Token before trying to connect to external service

```
def sync_with_remote_api
  api_key = Rails.application.secrets[:zoom_api_key]
  return if api_key.nil?

  puts "syncing the tag #{name} with a remote API.."
  sleep 3
end
```

Refactoring

- Separate API methods from parsing methods

```
def reload_repos
  list_of_repos = Github.repositories('ynonp')
  parse_data(list_of_repos)
end
```

Q & A

Thanks For Listening

- Ynon Perek
- ynon@ynonperek.com
- www.tocode.co.il