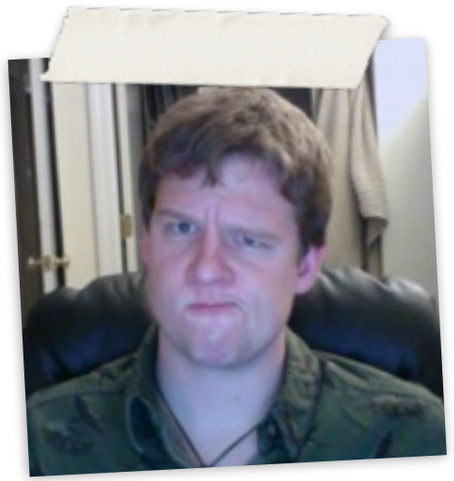
BACKFORTY

A RAILS PRIME PRIMER

How most of the pros do it



HI! I'M FLIP

@flipsasser

https://github.com/flipsasser

flip@inthebackforty.com

AGENDA

1. Intro to Rails Prime
2. In The Land of MySQL, Postgres is King
3. RSpec for Unit Tests
4. RSpec for Integration Tests
5. Building a Service Layer
6. Presenters and Decorators
7. HAML & Slim

WE'LL BE CONVERTING BETASTORE TO RAILS PRIME

This is the store app used in the Betamore intro to Rails class

HTTPS://GITHUB.COM/BACKFORTY/BETASTORE-SPRING-2014

INTRO TO RAILS PRIME

As coined by Steve Klabnik

BORN OF PRAGMATISTS AND SCOFFLAWS

THE BASICS

* Use Postgres instead of MySQL
* Use RSpec for testing
* Use service objects to simplify models
* Use a confusing template language

THE SUBTEXT

- Use one database for most services
- Write easy-to-read tests
- Use dependency injection to simplify code
- Writing HTML is for suckers

BACKFORTY

IN THE LAND OF MYSQL, POSTGRES IS KING

BENEFITS OF POSTGRES

1. It's pretty fast
2. It reduces your dependence on other services (and therefore your points of failure)
3. It tells you what's wrong when something's wrong
4. It's older and wiser
5. Also, it's pretty fast

BEFORE POSTGRES

* Use MySQL for relational data
* Use Memcached or Redis for a key-value store
* Use client-side encryption for data security
* Use Mongo or Couch for schema-less data
* Use Sphinx or SOLR for fulltext search
* Use backticks everywhere

WITH POSTGRES

- Use Postgres for relational data
- Use Hstore extension for a key-value store
- Use PGCrypto for data security
- Use JSON columns for schema-less data
- Use built-in fulltext indexing
- Never use a backtick again

BEFORE THE WORKSHOP

* Remove “mysql2” gem
* Add “pg” gem
* Update “config/database.yml”

BACKFORTY

WORKSHOP TIME

Upgrade Betastore to Postgres

BACKFORTY

RSPEC FOR UNIT TESTS

RSPEC IS LIKE TEST/UNIT'S COOLER LITTLE BROTHER

GOOFY ASSERTIONS

```
expect(value).to eq(something)
```

BLOCK-BASED TEST DEFINITIONS

```
1 require 'rails_helper'
2
3 describe Address do
4   before :each do
5     @valid_attributes = {
6       street: "555 S Logan St\nApartment 2",
7       city: "Denver",
8       state: "CO",
9       postal_code: "80209"
10    }
11  end
12
13  it "creates a new instance given valid attributes" do
14    FactoryGirl.create(:address, @valid_attributes)
15  end
16 end
```

INCREDIBLE HOOKS

```
1 RSpec.configure do |config|
2
3   config.before do |example|
4     if example.metadata[:js]
5       example.metadata[:use_truncation] = true
6     end
7     if example.metadata[:use_truncation]
8       DatabaseCleaner.strategy = :truncation
9     else
10      DatabaseCleaner.strategy = :transaction
11    end
12    DatabaseCleaner.start
13  end
14
15 end
```

INCREDIBLE HOOKS

```
1 require 'rails_helper'
2
3 feature "Adding a payment account" do
4
5   context "when signed in" do
6     let(:address) { FactoryGirl.create(:address, user: current_user) }
7
                                                    …
15
16     it "lets a user create credit cards", js: true do
17       stub_gateway_for_success
18       click_link "Add a Credit Card"
19       fill_form(:credit_card, {
20         number: '4111 1111 1111 1111',
21         expiration_2i: '9 - Sep',
22         expiration_1i: (Date.today.year + 3).to_s
23       })
24       click_button 'Add Credit Card'
25       expect_path(payment_accounts_path)
26       expect(current_user.credit_cards.count).to eq(1)
27     end
```

BEFORE THE WORKSHOP

* Add “rspec-rails” to your Gemfile’s test group
* Run the “rspec:install” generator

BACKFORTY

WORKSHOP TIME

Upgrade Betastore to RSpec unit tests

RSPEC FOR INTEGRATION TESTS

ALL THE FUN OF RSPEC

```
1 require 'rails_helper'
2
3 feature "Adding a payment account" do
4
5   context "when signed in" do
6     let(:address) { FactoryGirl.create(:address, user: current_user) }
7
                                        …
15
16     it "lets a user create credit cards", js: true do
17       stub_gateway_for_success
18       click_link "Add a Credit Card"
19       fill_form(:credit_card, {
20         number: '4111 1111 1111 1111',
21         expiration_2i: '9 - Sep',
22         expiration_1i: (Date.today.year + 3).to_s
23       })
24       click_button 'Add Credit Card'
25       expect_path(payment_accounts_path)
26       expect(current_user.credit_cards.count).to eq(1)
27     end
```

BEFORE THE WORKSHOP

* Configure Capybara to use Poltergeist in spec/support

WORKSHOP TIME

Upgrade Betastore to RSpec integration tests

BACKFORTY

BUILDING A SERVICE LAYER

OMAKASE

- Skinny controllers
- Fat models

RAILS PRIME

- Skinny controllers
- Skinny models
- Service objects handle non-ORM stuff

BACKFORTY

ACTIVERECORD: BEING AN ORM IS ENOUGH

BEFORE

```ruby
class Transaction < ActiveRecord::Base

  before_create :process

  belongs_to :parent, class_name: name
  belongs_to :payment

  maintain :status do
    state :clean, 0
    state :voided, 1
  end

  def process
    transaction = Braintree::Payment.new(amount: self.amount)
    transaction.process!(with_some_options: true)
    if transaction.success?
      self.transaction_hash = transaction.to_unique_id
    else
      errors.add(:transaction, "failed for some reason")
    end
  end

end
```

RAILS PRIME TENET

The model should store the details of the work

The work should be done elsewhere

AFTER

```ruby
class Transaction < ActiveRecord::Base

  belongs_to :parent, class_name: name
  belongs_to :payment

  maintain :status do
    state :clean, 0
    state :voided, 1
  end

end
```

APP/SERVICES/BRAINTREE_SERVICE.RB

```ruby
require 'braintree'

class BraintreeService

  def initialize(credentials = nil)
    credentials ||= {
      user: ENV['BRAINTREE_USER'],
      pass: ENV['BRAINTREE_PASS']
    }
  end

  def create_transaction(amount)
    transaction = Braintree::Payment.new(amount: amount)
    # Raise an error if something goes wrong
    transaction.process!(with_some_options: true)
    if transaction.success?
      Transaction.create({
        amount: amount,
        transaction_hash: transaction_hash.to_unique_id
      })
    end
  end
end
```

BENEFITS TO THIS APPROACH

1. Finite, easily test-able classes
2. Changes to a service's responsibilities will not break your ORM
3. Simpler classes responsible for single activity

WORKSHOP TIME

Add checkout to Betastore using a service object

PRESENTERS AND DECORATORS

PRESENTERS

- Move presentation logic from models to views
- Explicitly separate objects

DECORATORS

- Move presentation logic from models to views
- Masquerade as your models

APP/PRESENTERS/USERPRESENTER.RB

```
1 class UserPresenter
2
3   def initialize(user)
4     @user = user
5   end
6
7   def full_name
8     [
9       @user.first_name,
10      @user.last_name
11    ].compact.join(' ')
12  end
13 end
```

PROVIDE IT AS A HELPER...

```
def current_user_presenter
  @current_user_presenter ||=
    UserPresenter.new(current_user)
end
```

...AND USE IT DIRECTLY

```
<%= current_user_presenter.full_name %>
```

DECORATORS

```ruby
class UserDecorator < SimpleDelegator

  def full_name
    [first_name, last_name].join(' ')
  end

end
```

DECORATORS: PRETTIER, BUT DANGEROUS

BEFORE THE WORKSHOP

* Add “draper” gem

WORKSHOP TIME

Add presentation logic to decorators

BACKFORTY

HAML & SLIM

WHERE ONCE THERE WAS TERROR...

```
<div class="<%= "class-name" if user.class_name? %>”>
</div>
```

THERE CAME A GREAT CALM

```
div{class: user.class_name? && "class-name"}
```

THE GOAL IS TO MAKE HTML EASIER TO WRITE

The accident is increasingly semantic web interfaces

BEFORE THE WORKSHOP

* Add “slim” gem

WORKSHOP TIME

Start moving erb templates over to slim

IF YOU EVER HAVE ANY QUESTIONS,

PLEASE EMAIL ME:

flip@inthebackforty.com

Thanks!