MODERN RAILS

(solutions used building docsrc.com)

Rails makes it dead easy to shoot yourself in the foot.
(Not surprising, since it's from the US *cue laughing track*)

PROBLEM ONE

Complex behaviour in views is hard and / or slow to test!

- Lots of smaller cases in views multiply quickly
- two or three status paired with account roles make it hard to test for every case
- Link texts and buttons usually only (and partially) tested in slow feature specs

EXAMPLE

```
.image
  - unless @property.images.empty?
    = link_to property_medium(@property), '#photos'
    - if @property.images.size > 1
      %a{:href => '#photos'} more photos
  - else
    = link_to image_tag("default_medium.png")

.details.features
  .bedrooms
    = pluralize(@property.bedrooms, 'Bedroom', 'Bedrooms')
  - if @property.bathrooms?
    .bathrooms
      = pluralize(@property.bathrooms, 'Bathroom', 'Bathrooms')
  - if @property.garden?
    .garden
      = @property.garden
  - if @property.property_type?
    .grouped.property_type
      = @property.property_type
  - if @property.let_type?
    .let_type
      = @property.let_type
  - if @property.council_tax_band?
    .tax_band
      Tax Band
      = @property.council_tax_band
  %p.terms
    Available on
    = l @property.available_on
    %br
    = @property.minimum_tenancy_term
    Months
    Minimum Let
```

- a HAML view lots of ifs and buts, but no pots and pans

PRESENTER OBJECTS

Solution: Presenters

PRESENTERS

```ruby
class TemplatePresenter
  attr_reader :template

  delegate :file_type,
           :created_at,
           :is_enabled,
           to: :template

  def initialize(template, view_context)
    @template = template
    @view     = view_context
  end

  def field_name_list
    h.truncate(template.field_config.keys.join(", "))
  end

  def active_mark
    if is_enabled
      h.content_tag :span, "\u2713", class: "text-success"
    else
      nil
    end
  end

  def config_link
    h.link_to "Config",
              h.edit_template_path(template.id),
              class: "btn btn-outline-primary btn-sm"
  end

  def h
    @view
  end
end
```

```erb
<tr>
  <td><%= template.active_mark %></td>
  <td><%= template.field_name_list %></td>
  <td><%= template.created_at.to_s(:db) %></td>
  <td>
   <%= template.config_link  %>
  </td>
</tr>
```

The trick: Inject view context so you can do everything you would normally do in a partial
Caveat: When stuff gets too complex, use more sub-presenters and partials for them
You can now test all those things as a plain unit test (injecting a mock view context)

PROBLEM TWO

An HTML form on a page
does not necessarily map
to the ActiveRecord object!

- Rails pretty much assumes this, but sometimes you are editing or creating multiple objects or even parts of them
- Stuff gets really tricky when different validations need to be triggered in varying cases

EXAMPLE

Entering data for a new account and an organisation at the same time.

Also: Maybe you need different validation in different situations.

- Maybe even adding payment details
- Better examples?!?

FORM OBJECTS

FORM OBJECTS

```ruby
module Forms
  class PasswordReset
    include ActiveModel::Model

    attr_accessor :password, :password_confirmation
    attr_reader :account

    validates :password, length: { minimum: 6 }
    validates :password, confirmation: true
    validates :password_confirmation, presence: true

    def initialize(account, attributes = {})
      @account = account
      super(attributes)
    end

    def save
      return false unless valid?
      account.password = password
      account.change_password!(password)
    end
  end
end
```

- Create an object backing every field of your form.
- Good type deserialisation missing in Rails (though simple hack with the type registry)
- Validate per form
- Different validations possible
- (Also validate in your backend logic, but only for the really important bits.)

PROBLEM THREE

Controller logic gets
complex and out of hand!

- Controllers with complex logic (different submit cases, multiple error and success cases) are tricky to handle
- State too complicated to store to second step

EXAMPLE

```ruby
def create
  session[:tenant_registration_params] ||= {}

  session[:tenant_registration_params].deep_merge!(params[:tenant_registration]) if params[:tenant_registration]
  @registration = TenantRegistration.new(session[:tenant_registration_params])
  set_current_step_to_session_step

  if session[:registration_step].nil?
    @registration.current_step = @registration.steps.first
    set_session_step_to_current_step
  end

  # if user presses 'back button'
  if params[:back_button]
    @registration.previous_step
    set_session_step_to_current_step
    render 'tenant_registrations2step_alt/new'

  # else if user is on step 1 and has entered valid details
  elsif session[:registration_step] == "step_1" && @registration.valid?
    change_step
    render 'tenant_registrations2step_alt/new'

  # else if user is on step 2 and has entered valid details but there are no agents available in their chosen areas at that budget
  elsif @registration.valid? && @registration.all_branches_in_area_at_budget.empty? && @registration.current_step == @registration.steps.second
    if params[:tenant_registration][:tenancy_term] == "less than 6 months"
      redirect_to short_term_rentals_path
    elsif params[:tenant_registration][:housing_benefit] == "true"
      redirect_to housing_benefit_tenants_path
    else
      set_session_step_to_other_options
      set_current_step_to_session_step
      other_possible_options
      render 'tenant_registrations2step_alt/new'
    end

  # else if user is on step 2 and has entered valid details but is looking for a short-term rental
  elsif session[:registration_step] == "step_2" && @registration.valid?
    if params[:tenant_registration][:tenancy_term] == "less than 6 months"
      redirect_to short_term_rentals_path
    else
      change_step
      load_branches
      if @branches.nil? || @branches.empty?
        render 'tenant_registrations/branch_notifications/no_agents'
      else
        render 'tenant_registrations2step_alt/new'
      end
    end

  # else if user is on the other options screen and has chosen a valid other option
  elsif session[:registration_step] == "other_options"
    set_session_step_to_step_3
    set_current_step_to_session_step
    load_branches
    render 'tenant_registrations2step_alt/new'

  # else if user is on step 2 and has entered valid details and agents have successfully been found for the user's requirements
  elsif session[:registration_step] == "step_3" && @registration.valid?
    @branch_ids = params[:branch_id]
    if @branch_ids.nil? || @branch_ids.empty?
      flash[:error] = "You didn't choose any agents to register with"
      load_branches
      render 'tenant_registrations2step_alt/new'
    else
      @registration.save
      if @registration.move_in_date > TimingSettings::LATEST_ALLOWABLE_MOVE_IN_DATE
        @branch_ids.each do |b|
          BranchNotification.create(branch_id: b, tenant_registration_id: @registration.id, agent_notified_at: TimingSettings::MOVE_IN_DATE_TOO_LATE)
        end
        BranchNotificationMailer.admin_notification(@registration).deliver
        UserMailer.t_reg_too_early(@registration).deliver
      else
        @branch_ids.each do |b|
          BranchNotification.create(branch_id: b, tenant_registration_id: @registration.id)
        end
        BranchNotificationMailer.admin_notification(@registration).deliver
        UserMailer.t_reg_success(@registration).deliver
      end
      session[:tenant_registration_id] = @registration.id
      if more_info_needed?
        redirect_to add_optional_info_tenant_registration_path(@registration)
      else
        redirect_to congratulations_path
      end
      reset_registration_session_data
    end

  # else if none of the above
  else
    set_current_step_to_session_step
    render 'tenant_registrations2step_alt/new'
  end
end
```

RUNNER OBJECTS

RUNNER OBJECTS

```ruby
class HandleComplexAction
  def initialize(data, controller)
    @data = data
    @controller = controller
  end

  def call
    case data.status
    when case_1:
       do_something
       controller.render_success
    when case_2:
      do_something_else
      controller.render_success(with_extra_config: true)
    when case_3:
      controller.render_error
    default:
      controller.redirect_back
  end

  private

  attr_reader :data, :controller
end
```

Some people call them actions, some service objects. In the controller case I call them “Runners”, because they “run” the controller. It’s like a remote control for the controller.
Again you can test the logic in isolation with out setting up all the controller state.

PROBLEM FOUR

You need to save a record in the database in order to have an ID for it!

- This sucks for distributed systems (centralized ID creation)
- Also difficult when creating object graphs
- Sequential IDs can also easily be guessed

EXAMPLE

```
2.4.1 :048 > t = AR::Template.new.id
 => nil
```

You don’t know what the ID is going to be, and you cannot set it!

UUIDS

UUIDS

```
2.4.1 :049 > SecureRandom.uuid
 => "443a80dc-4d7e-487a-b51d-2aa215a73899"
```

```ruby
class CreateTemplates < ActiveRecord::Migration[5.1]
  def change
    create_table :templates, id: :uuid do |t|
      t.references :organisation, type: :uuid, index: true

      …

      t.timestamps null: false
    end
  end
end
```

- Extremely unlikely to have collisions.
- You can generate IDs everywhere (even in the frontend if you like) and pass them
- Works great with PostgreSQL

TO SUMMARIZE:

DON’t BE AFRAID TO USE OBJECTS TO ENCAPSULATE THINGS.

USE UUIDS.

T. HANKS!