FORMS ON RAILS

Vladimir Dementyev

Evil Martians

SF Bay Area Ruby #7


WINS: 10
99
UI FORM
RUBY ON RAILS
FIGHT!
RAILS

Cables / Create new cable

Name

amazing-cable

Region

sea

Your backend

Ruby on Rails | JavaScript | Hotwire | Any Backend

Using Ruby on Rails? Continue with this option whether you want to power-up your Action Cable, speed up Hotwire streams or use AnyCable pub/sub.

RPC URL

https://myapp.example.dev/api/anycable

anycable-saaqs spike/wizard-demo

_wizard.html.erb

anycable-saaqs/app/views/cables/_wizard.html.erb

```erb
<%= form_for @cable do |f| %>
  <%= f.text_field :name, required: true %>

  <%= f.text_field :region, required: true %>

  <%= f.radio_button :framework, "rails" %>
  <%= f.radio_button :framework, "js" %>
  <%= f.radio_button :framework, "hotwire" %>
  <%= f.radio_button :framework, "default" %>

  <%= f.text_field :rpc_host %>

  <%= f.text_field :rpc_secret %>

  <%= f.text_field :secret %>
  <%= f.text_field :turbo_secret %>
  <%= f.text_field :jwt_secret %>

  <%= f.submit "Create" %>
<% end %>
```

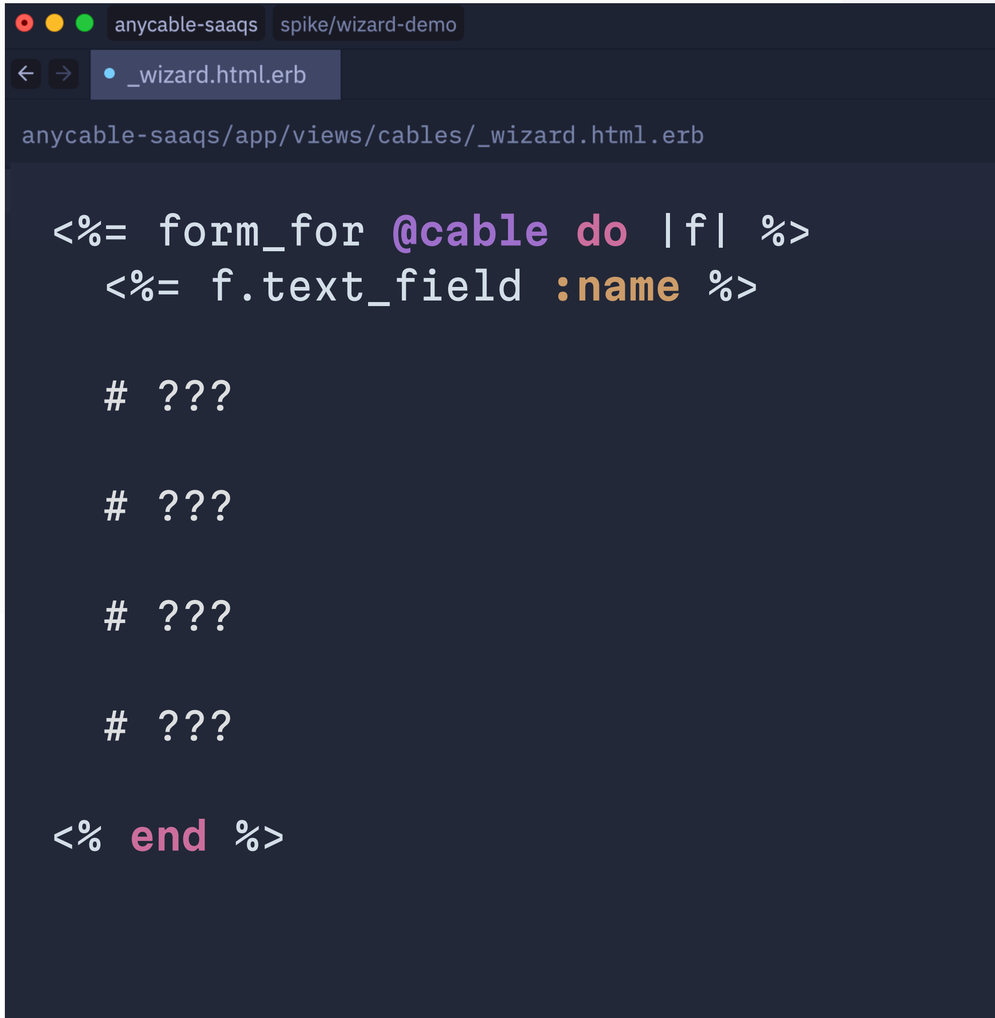
Cables / Create new cable

Name

amazing-cable

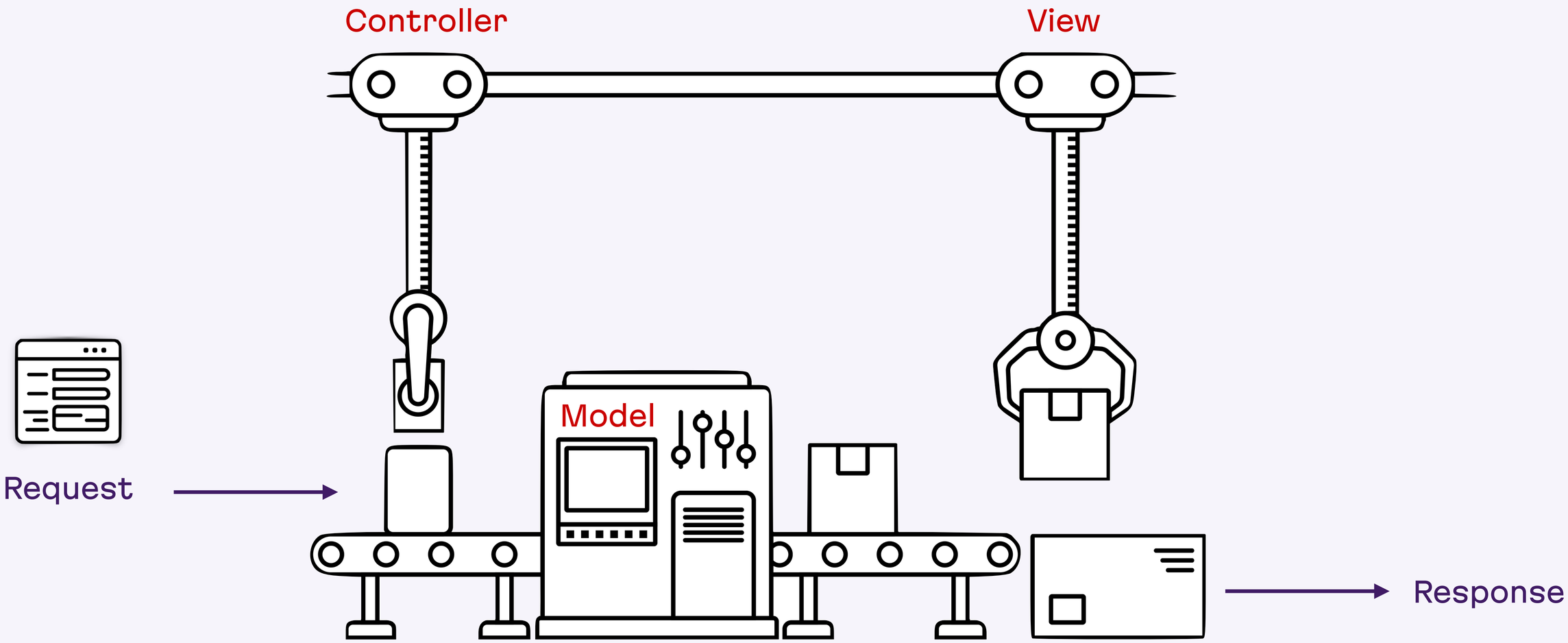
Pick a name for your AnyCable installation!

Next

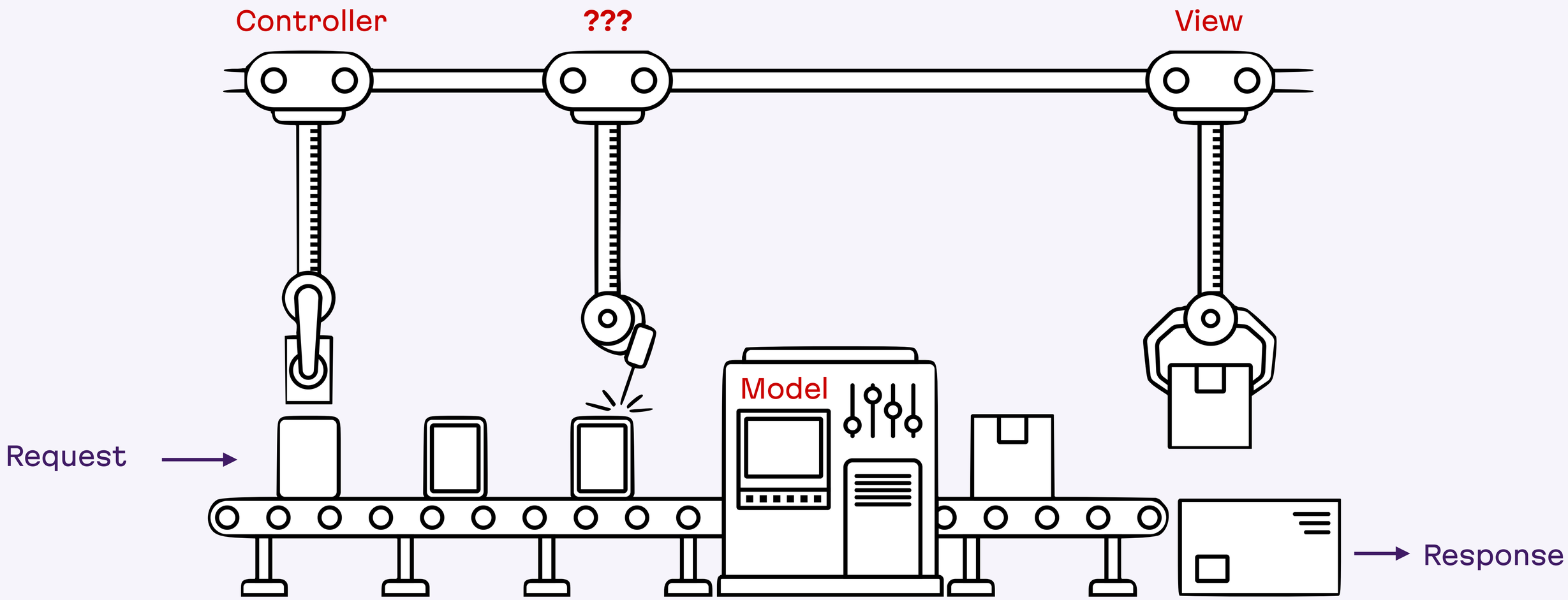



RAILS WAY

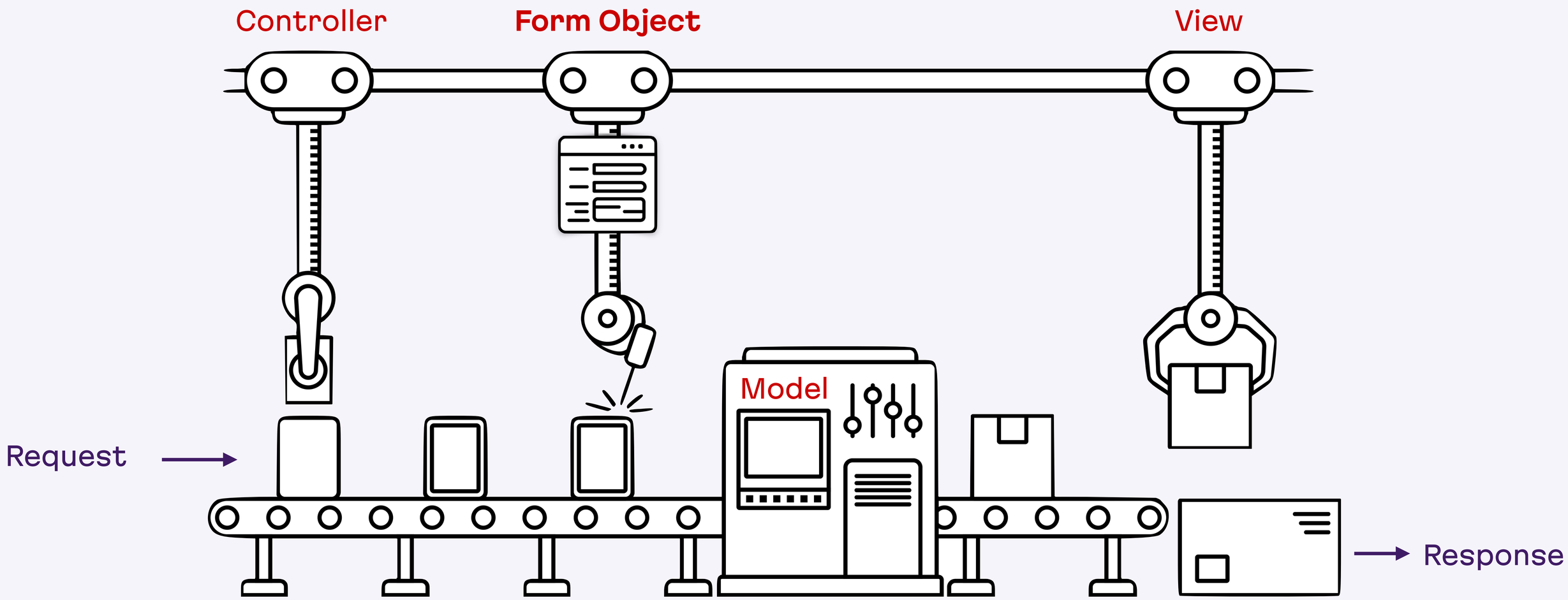
Rails Way



Rails Way+



Rails Way+




SEPARATE
CONCERNS
* without using Rails Concerns 😉

Form Object: concerns

- Context-specific validations
- User input transformation
- User feedback
- Custom UI-driven logic (like wizards)

FORMS FROM THE PAST

dhh / aggregator.rb

Created 9 years ago

May 5, 2014 at 5:46 PM GMT+4

Star 5 Fork 0

Code Revisions 1 Stars 5 Download ZIP

aggregator.rb Raw

```ruby
class Profile < ActiveModel::Aggregator
  aggregate :user
  aggregate :email

  def combine
    user.save
    user.emails.build(email_attributes).save
  end
end

class ProfileController < ApplicationController
  def create
    @profile = Profile.new(profile_attributes)

    if @profile.save
      redirect_to somewhere
    else
      render :new
    end
  end
end
```

actionform Public

Watch 26 | Fork 42 | Star 296

forked from m-Peter/activeform

rafaelfranca Update the travis badge — fe9ed88 on Jul 17, 2015 — 13[illegible] commits

Action Form

DISCLAIMER: This project is an experiment and should not be used in applications.

Consider an example where you want to create/update a conference that can have many speakers which can present a single presentation with one form submission. You start by defining a form to represent the root model, `Conference`:

```
class ConferenceForm < ActionForm::Base
  self.main_model = :conference

  attributes :name, :city

  validates :name, :city, presence: true
end
```

Your form object has to subclass `ActionForm::Base` in order to gain the necessary API. When defining the form, you have to specify the main_model the form represents with the following line:

```
self.main_model = :conference
```

Reform

```ruby
class AlbumForm < Reform::Form
  property :title

  validates :title, presence: true

  property :artist do
    property :name

    validates :name, presence: true
  end
end
```

Reform

```
class AlbumsController < ApplicationController
  def new
    @form = AlbumForm.new(Album.new)
  end
end
```

Introduce `ActiveModel::Base` class #49647

Closed **seanpdoyle** wants to merge 1 commit into `rails:main` from `seanpdoyle:am-base`

Conversation 32 | Commits 1 | Checks 3 | Files changed 8

seanpdoyle commented on Oct 16 • edited

Contributor

Motivation / Background

Database-backed Active Record models are Rails's de-facto classes for business logic, which makes sense. Reading fr
and writing to the database can encompass the majority of most application's logic.

However, there are circumstances where Active Record classes either aren't a good fit, have grown large enough to ma
their maintenance burdens outweigh the benefits of new code, or are tedious to special-case for context-specific
integrations with other parts of Rails like Action Pack routing or Action View form building.

This is not a new perspective. There are various philosophies on how to categorize and bucket business logic when Active Record isn't the best candidate.

The de-facto alternative is plain-old Ruby classes, or Ruby classes that mix-in variations of what the `ActiveModel` module has to offer (e.g. `ActiveModel::API`, `ActiveModel::Model`, `ActiveModel::Attributes`, etc.).

Introduce `ActiveModel::Base` class #49647

Closed **seanpdoyle** wants to merge 1 commit into `rails:main` from `seanpdoyle:am-base`

rafaelfranca commented on Oct 18 — Member

I think this is one of the cases of "We still didn't arrive to a solution that the Rails Core is happy with". We already tried form objects and we weren't happy with were we arrived. That is why I think people experimenting with objects like that in libraries is a good thing. Rails doesn't need to included every single solution.

1

Record isn't the best candidate.

The de-facto alternative is plain-old Ruby classes, or Ruby classes that mix-in variations of what the `ActiveModel` module has to offer (e.g. `ActiveModel::API`, `ActiveModel::Model`, `ActiveModel::Attributes`, etc.).

LIGHTS!
ACTION!
FORM!

* or Action Form I'm happy with 🙂


packt
1ST EDITION
Layered Design for Ruby on Rails Applications
Practical design patterns for maintainable Ruby on Rails web applications
VLADIMIR DEMENTYEV

Handling User Input outside of Models

In this chapter, we'll continue discussing abstractions related to Rails models. This time, we will talk about user-driven operations and the corresponding design patterns. First, we'll talk about modifying operations (creating or updating models) and introduce the concept of a **form object**, which is an object representing a user interface form in the code base. Then, we'll discuss how to read (or filter) data based on user-provided parameters with the help of **filter objects**.

You will learn how to identify functionality that can be moved to form and filter objects, so you can introduce new abstraction layers and reduce the responsibility of the existing ones (especially the model layer).

We will cover the following topics:

- Form objects – closer to the UI, farther from persistence
- Filter objects or user-driven query building

This chapter aims to give you practice in extracting the presentation layer abstractions from Rails models so you learn which properties they have in common and how and when to use them in your code base.

Technical requirements

In this chapter, and all chapters of this book, the code given in code blocks is designed to be executed on Ruby 3.2 and Rails 7. Many of the code examples will work on earlier versions of the afore mentioned software.

You will find the code files on GitHub at https://github.com/PacktPublishing/Layered-Design-for-Ruby-on-Rails-Applications/tree/main/Chapter07.

Form objects – closer to the UI, farther from persistence

From a user interaction point of view, using almost any web application could be seen as a sequence of interleaving form submissions and link clicks. Although this is a very rough definition, it's good enough to demonstrate that there are two primary ways an application deals with user-provided

Cables / Create new cable

Name

amazing-cable

Region

sea

Your backend

Ruby on Rails | JavaScript | Hotwire | Any Backend

Using Ruby on Rails? Continue with this option whether you want to power-up your Action Cable, speed up Hotwire streams or use AnyCable pub/sub.

RPC URL

https://myapp.example.dev/api/anycable

anycable-saaqs spike/wizard-demo

create_form.rb

anycable-saaqs/app/forms/cable/create_form.rb

```ruby
class Cable
  class CreateForm < ApplicationForm
    self.model_name = "Cable"

    attribute :name
    attribute :region, default: -> { "sea" }
    attribute :framework, default: -> { "rails" }
    attributes :secret, :rpc_host, :rpc_secret,
               :turbo_secret, :jwt_secret

    attr_reader :cable

    def initialize(...)
      super
      @cable = Cable.new(
        name:, region:,
        metadata: {framework:},
        configuration: {
          secret:, rpc_host:, rpc_secret:,
          turbo_secret:, jwt_secret:
        }
      )
    end

    def submit! = cable.save!
  end
end
```

anycable-saaqs spike/wizard-demo

create_form.rb

anycable-saaqs/app/forms/cable/create_form.rb

```ruby
class Cable
  class CreateForm < ApplicationForm
    self.model_name = "Cable"

    attribute :name
    attribute :region, default: -> { "sea" }
    attribute :framework, default: -> { "rails" }
    attributes :secret, :rpc_host, :rpc_secret,
               :turbo_secret, :jwt_secret

    attr_reader :cable

    def initialize(...)
      super
      @cable = Cable.new(
        name:, region:,
        metadata: {framework:},
        configuration: {
          secret:, rpc_host:, rpc_secret:,
          turbo_secret:, jwt_secret:
        }
      )
    end

    def submit! = cable.save!
  end
end
```

UI / schema decoupling

anycable-saaqs spike/wizard-demo

create_form.rb

anycable-saaqs/app/forms/cable/create_form.rb

```ruby
class Cable
  class CreateForm < ApplicationForm
    self.model_name = "Cable"

    attribute :name
    attribute :region, default: -> { "sea" }
    attribute :framework, default: -> { "rails" }
    attributes :secret, :rpc_host, :rpc_secret,
               :turbo_secret, :jwt_secret

    validates :name, :region, presence: true
    validates :rpc_host, format: %r{\Ahttps?://}, allow_blank: true

    attr_reader :cable

    def initialize(...)
      # ...
    end

    def submit! = cable.save!
  end
end
```

anycable-saaqs spike/wizard-demo

create_form.rb

anycable-saaqs/app/forms/cable/create_form.rb

```ruby
class Cable
  class CreateForm < ApplicationForm
    self.model_name = "Cable"

    attribute :name
    attribute :region, default: -> { "sea" }
    attribute :framework, default: -> { "rails" }
    attributes :secret, :rpc_host, :rpc_secret,
               :turbo_secret, :jwt_secret

    validates :cable_is_valid
    validates :rpc_host, format: %r{\Ahttps?://}, allow_blank: true

    attr_reader :cable

    def initialize(...)
      # ...
    end

    def submit! = cable.save!

    private

    def cable_is_valid
      return if cable.valid?
      merge_errors!(cable)
    end
  end
end
```

Validations:

a) model delegation

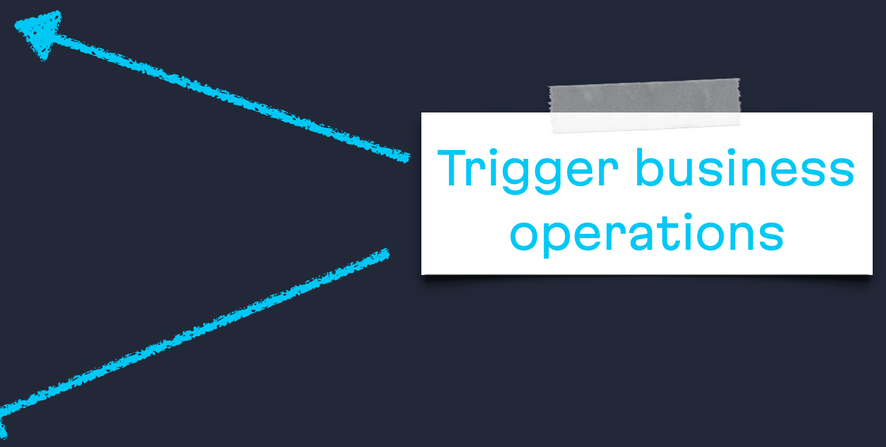
b) context-specific

anycable-saaqs spike/wizard-demo

create_form.rb

anycable-saaqs/app/forms/cable/create_form.rb

```ruby
class Cable
  class CreateForm < ApplicationForm
    self.model_name = "Cable"

    attribute :name
    attribute :region, default: -> { "sea" }
    attribute :framework, default: -> { "rails" }
    attributes :secret, :rpc_host, :rpc_secret,
               :turbo_secret, :jwt_secret

    validates :cable_is_valid
    validates :rpc_host, format: %r{\Ahttps?://}, allow_blank: true

    after_commit :enqueue_provisioning

    attr_reader :cable

    def initialize(...)
      # ...
    end

    def submit! = cable.save!

    private

    def enqueue_provisioning = cable.provision_later
  end
end
```



anycable-saaqs spike/wizard-demo

create_form.rb

anycable-saaqs/app/forms/cable/create_form.rb

```ruby
class Cable
  class CreateForm < ApplicationForm
    self.model_name = "Cable"

    attribute :name
    attribute :region, default: -> { "sea" }
    attribute :framework, default: -> { "rails" }
    attributes :secret, :rpc_host, :rpc_secret,
               :turbo_secret, :jwt_secret

    validates :cable_is_valid
    validates :rpc_host, format: %r{\Ahttps?://}, allow_blank: true

    after_commit :enqueue_provisioning

    attr_reader :cable

    def initialize(...)
      # ...
    end

    def submit! = cable.save!

    private

    def enqueue_provisioning = cable.provision_later
  end
end
```

“...work with the framework, not against it.”

—DHH, The Rails Way, 1st edition

- Learn Rails building blocks and re-use them
- Agree to the overachieving principle of conventions
- Make your code play nicely with other Rails components

anycable-saaqs spike/wizard-demo

application_form.rb

anycable-saaqs/app/forms/application_form.rb

```ruby
class ApplicationForm
  include ActiveModel::API
  include ActiveModel::Attributes

  define_callbacks :save, only: :after
  define_callbacks :commit, only: :after

  def save
    return false unless valid?

    with_transaction do
      AfterCommitEverywhere.after_commit { run_callbacks(:commit) }
      run_callbacks(:save) { submit! }
    end
  end

  def model_name
    ActiveModel::Name.new(nil, nil, self.class.name.sub(/Form$/, ""))
  end

  private

  def with_transaction(&) = ApplicationRecord.transaction(&)

  def submit!
    raise NotImplementedError
  end
end
```

Validations / Types

Callbacks

Transactions awareness

Action View compat

Interface

anycable-saaqs spike/wizard-demo

cables_controller.rb

anycable-saaqs/app/controllers/cables_controller.rb

```ruby
class CablesController < ApplicationController
  def new
    authorize!
    @cable = Cable.new
  end

  def create
    authorize!
    @cable = Cable.new(cable_params)

    if @cable.save
      redirect_to cable_path(@cable), notice: "Success!"
    else
      render :new, status: :unprocessable_entity
    end
  end
end
```

anycable-saaqs spike/wizard-demo

cables_controller.rb

anycable-saaqs/app/controllers/cables_controller.rb

```ruby
class CablesController < ApplicationController
  def new
    authorize!
    @form = Cable::CreateForm.new
  end

  def create
    authorize!
    @form = Cable::CreateForm.from(params.require(:cable))

    if @form.save
      redirect_to cable_path(@form.cable), notice: "Success!"
    else
      render :new, status: :unprocessable_entity
    end
  end
end
```

anycable-saaqs spike/wizard-demo

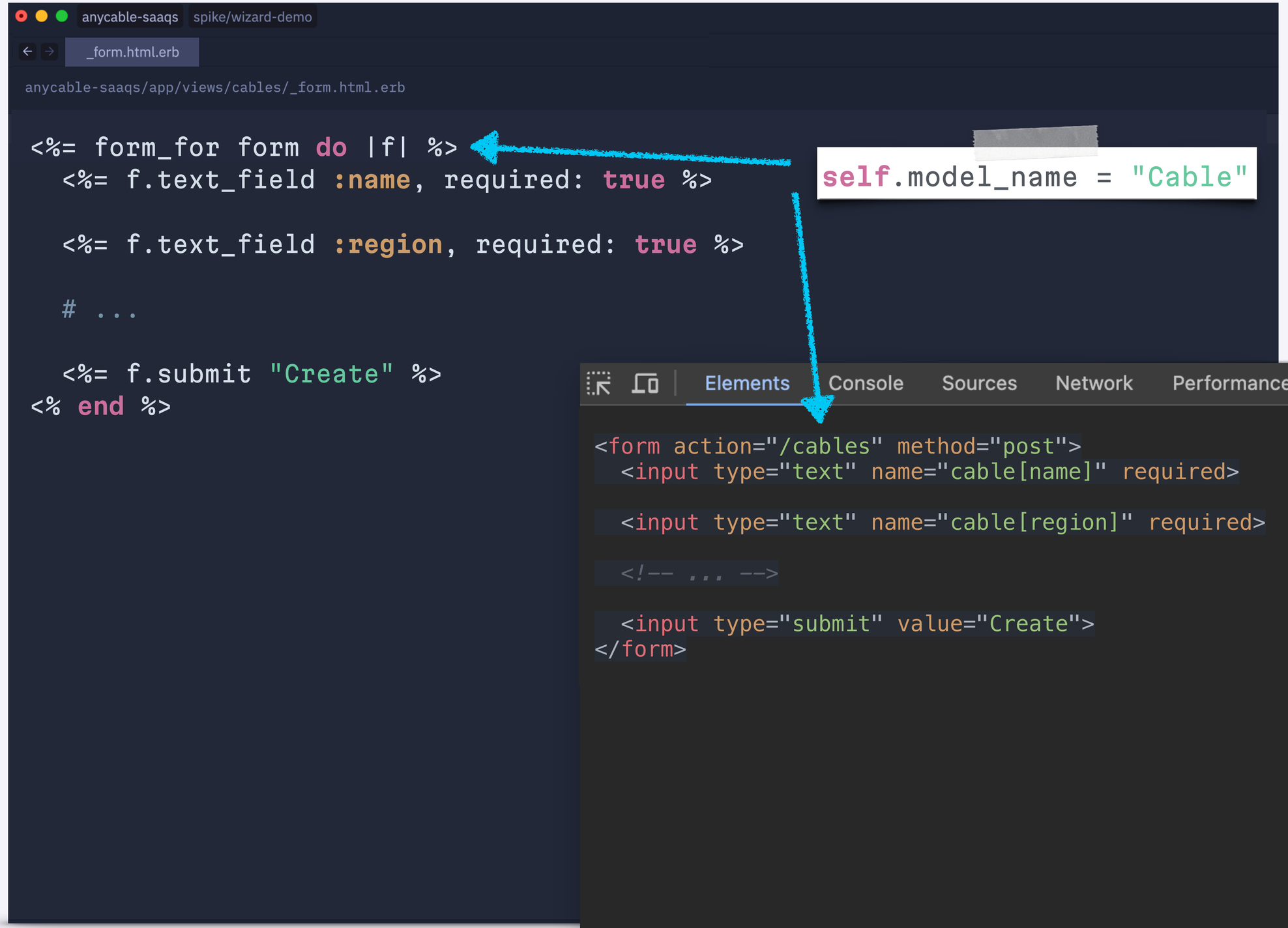
_form.html.erb

anycable-saaqs/app/views/cables/_form.html.erb

```erb
<%= form_for @cable do |f| %>
  <%= f.text_field :name, required: true %>

  <%= f.text_field :region, required: true %>

  # ...

  <%= f.submit "Create" %>
<% end %>
```

Elements Console Sources Network Performance Memory App

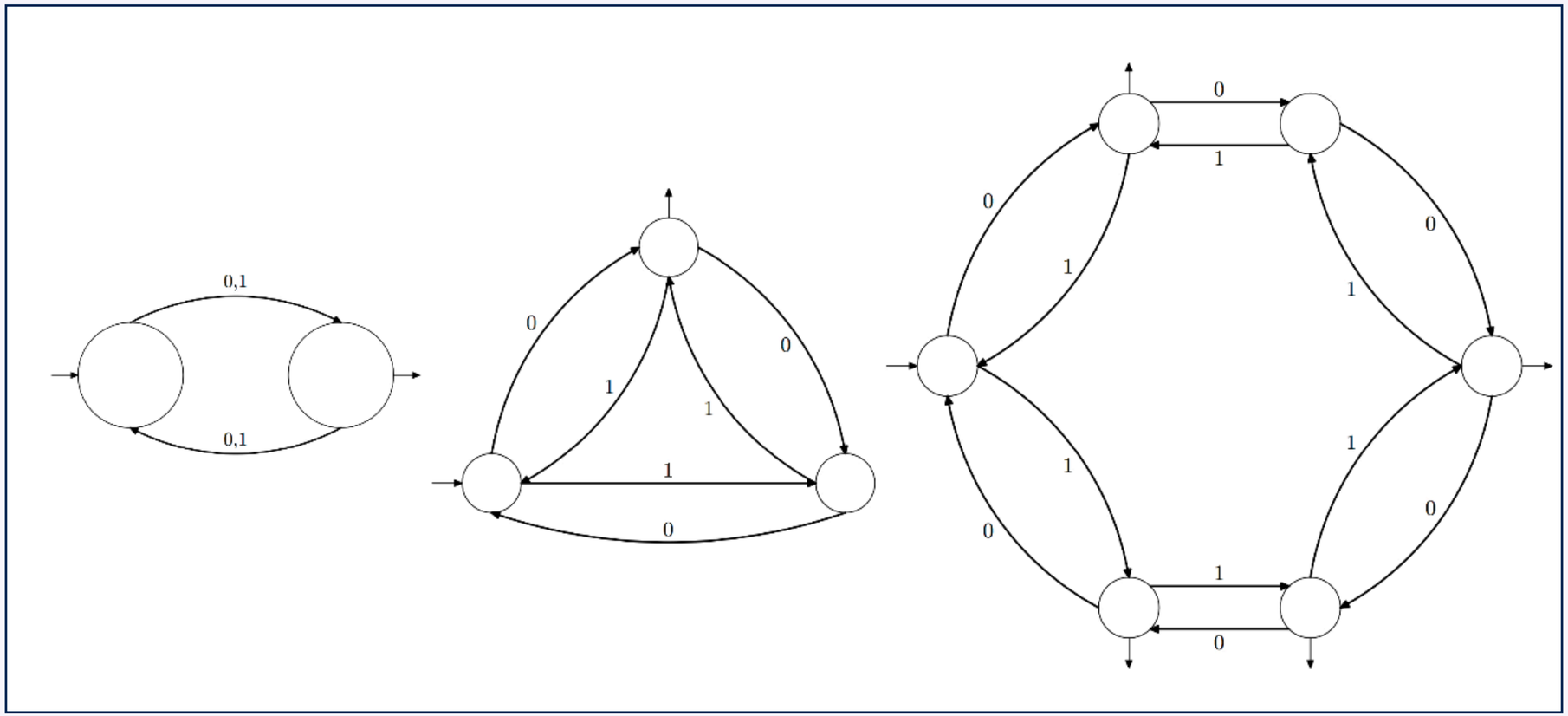
```html
<form action="/cables" method="post">
  <input type="text" name="cable[name]" required>

  <input type="text" name="cable[region]" required>

  <!-- ... -->

  <input type="submit" value="Create">
</form>
```


anycable-saaqs
spike/wizard-demo
_form.html.erb
anycable-saaqs/app/views/cables/_form.html.erb
<%= form_for form do |f| %>
<%= f.text_field :name, required: true %>
<%= f.text_field :region, required: true %>
...
<%= f.submit "Create" %>
<% end %>
self.model_name = "Cable"
Elements
Console
Sources
Network
Performance
Memory
<form action="/cables" method="post">
<input type="text" name="cable[name]" required>
<input type="text" name="cable[region]" required>
<!-- ... -->
<input type="submit" value="Create">
</form>


SEND IN
THE WIZARDS!

- Now the form logic has its own home in the application
- We can localize changes related to this feature and iterate faster

Wizard is a DFA



```
gem "workflow"
```

https://github.com/geekq/workflow

anycable-saaqs spike/wizard-demo

create_form.rb

anycable-saaqs/app/forms/cable/create_form.rb

```ruby
class Cable
  class CreateForm < ApplicationForm
    class Wizard < ApplicationWorkflow
      workflow do
        state :name do
          event :submit, transitions_to: :framework
        end

        state :framework do
          event :submit, transitions_to: :rpc, if: :needs_rpc?
          event :submit, transitions_to: :secrets
          event :back, transitions_to: :name
        end

        state :rpc do
          event :submit, transitions_to: :secrets
          event :back, transitions_to: :framework
        end

        state :secrets do
          event :submit, transitions_to: :region
          event :back, transitions_to: :rpc, if: :needs_rpc?
          event :back, transitions_to: :framework
        end

        state :complete
      end
    end
  end
end
```

anycable-saaqs spike/wizard-demo

create_form.rb

anycable-saaqs/app/forms/cable/create_form.rb

```ruby
class Cable
  class CreateForm < ApplicationForm
    class Wizard < ApplicationWorkflow
      # ...
    end

    attribute :wizard_state, default: -> { "name" }
    attribute :wizard_action

    def submit!
      if wizard_action == "back"
        wizard.back!
      else
        wizard.submit!
      end

      return false unless wizard.complete?

      cable.save!
    end

    def wizard = @wizard ||= Wizard.new(self)
  end
end
```

anycable-saaqs spike/wizard-demo

cables_controller.rb

anycable-saaqs/app/controllers/cables_controller.rb

```ruby
class CablesController < ApplicationController
  def new
    authorize!
    @form = Cable::CreateForm.new
  end

  def create
    authorize!
    @form = Cable::CreateForm.from(params.require(:cable))

    if @form.save
      redirect_to cable_path(@form.cable), notice: "Success!"
    else
      status = @form.valid? ? :created : :unprocessable_entity
      render :new, status:
    end
  end
end
```

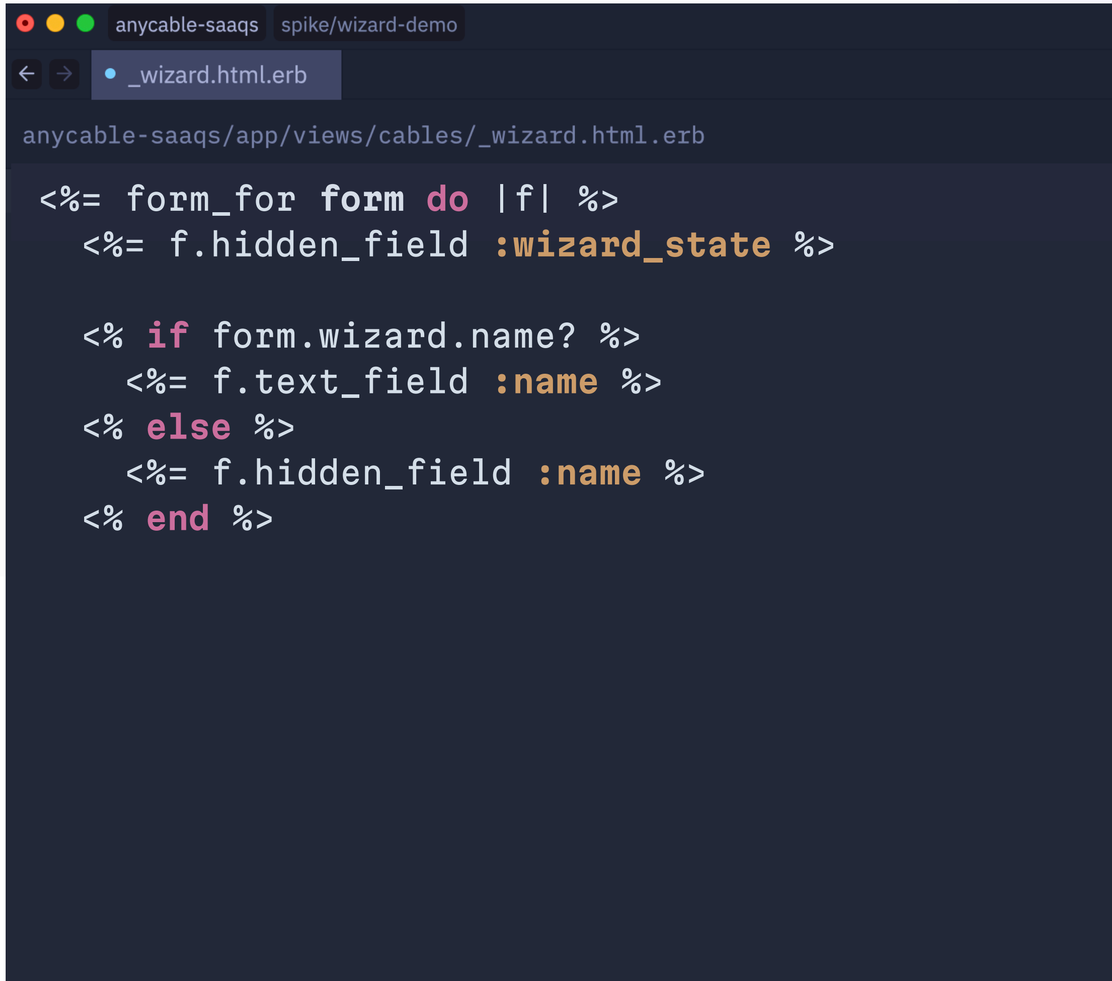
Cables / Create new cable

Name

amazing-cable

Pick a name for your AnyCable installation!

Next



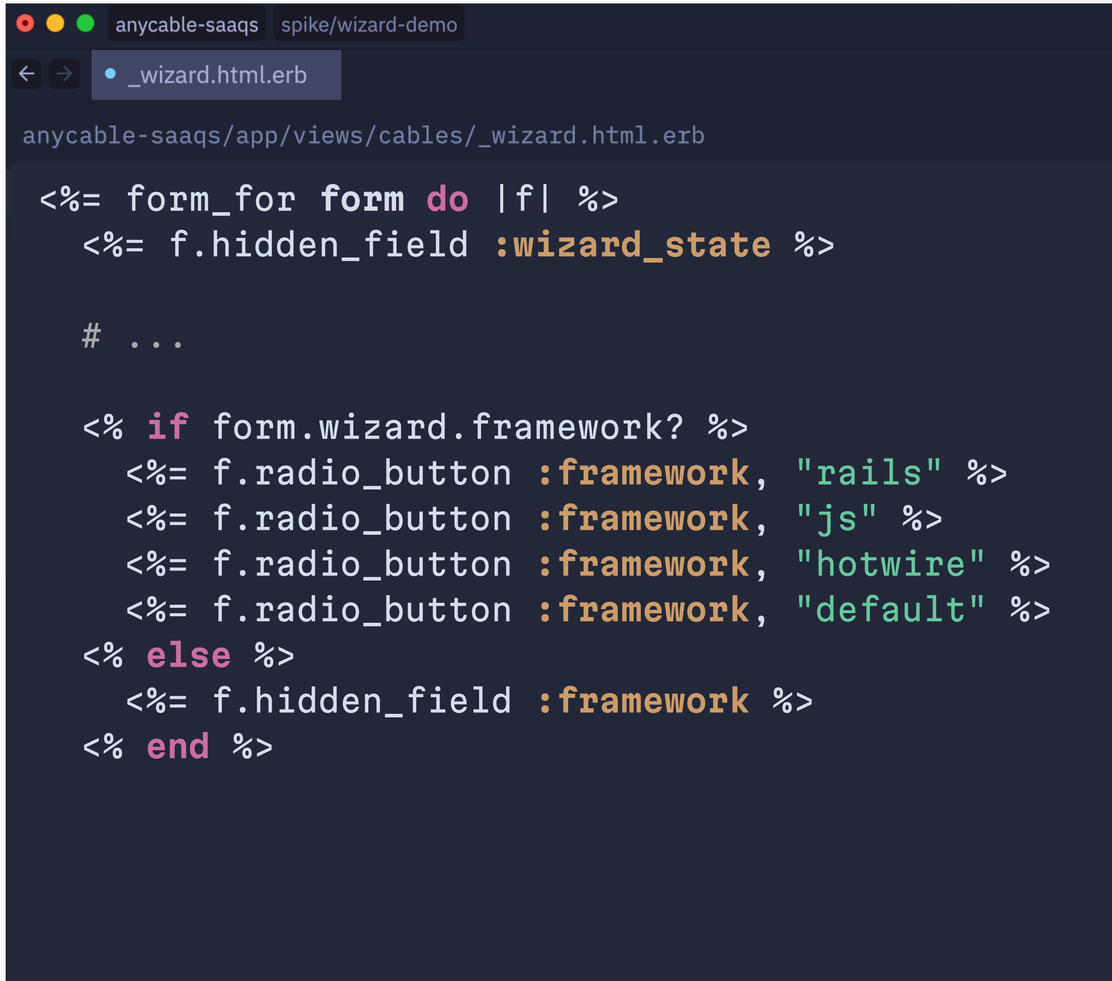
Cables / Create new cable

Name

sf-ruby-cable

Pick a name for your AnyCable installation!

Next



Cables / Create new cable

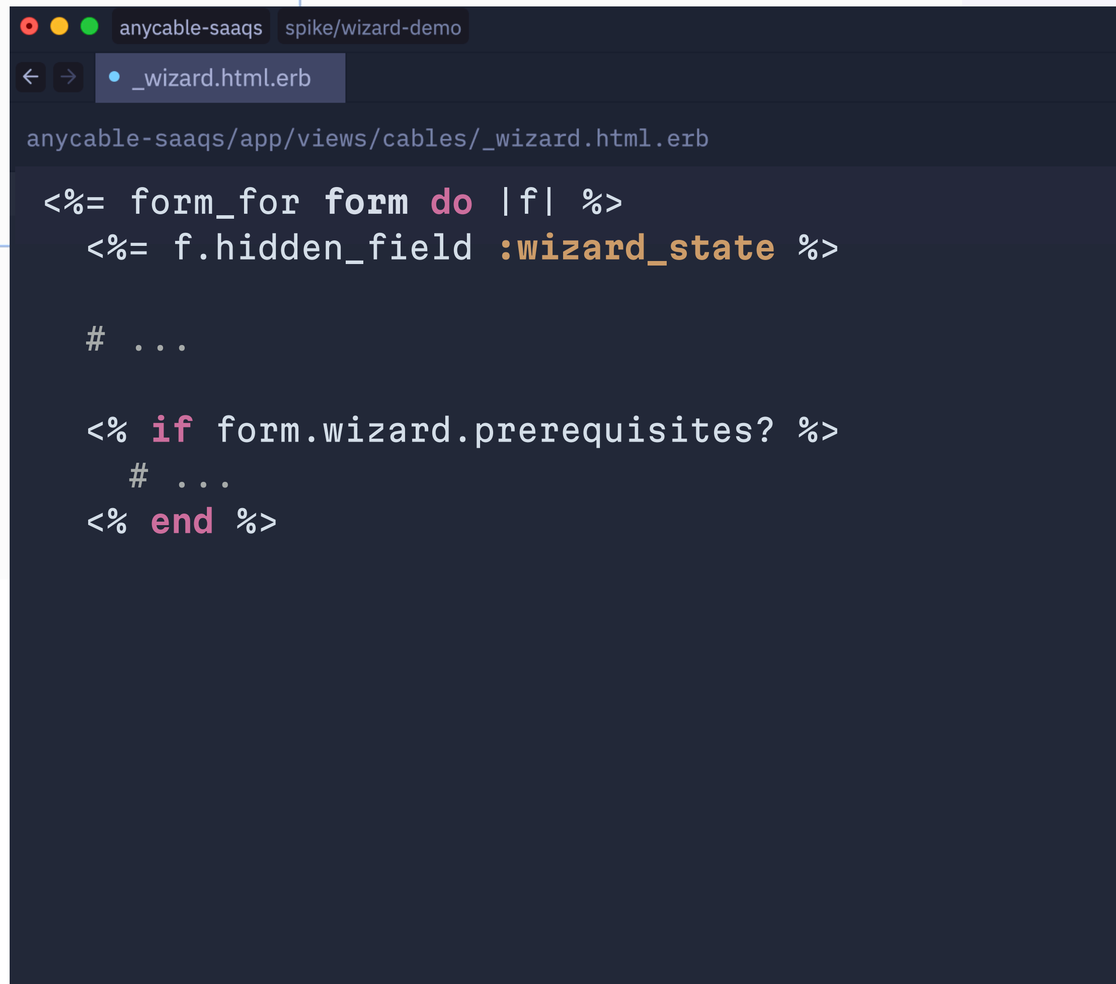
Your backend

Ruby on Rails | JavaScript | Hotwire | Any Backend

Using Ruby on Rails? Continue with this option whether you want to power-up your Action Cable, speed up Hotwire streams or use AnyCable pub/sub.

Back Next

Select the framework/technology you use to build your application (backend), so we can guide you through the setup process.



Cables / Create new cable

To empower your Rails app with AnyCable, you need to install the anycable-rails-core gem. Add it to your Gemfile:

```
gem "anycable-rails-core", "~> 1.5"
```

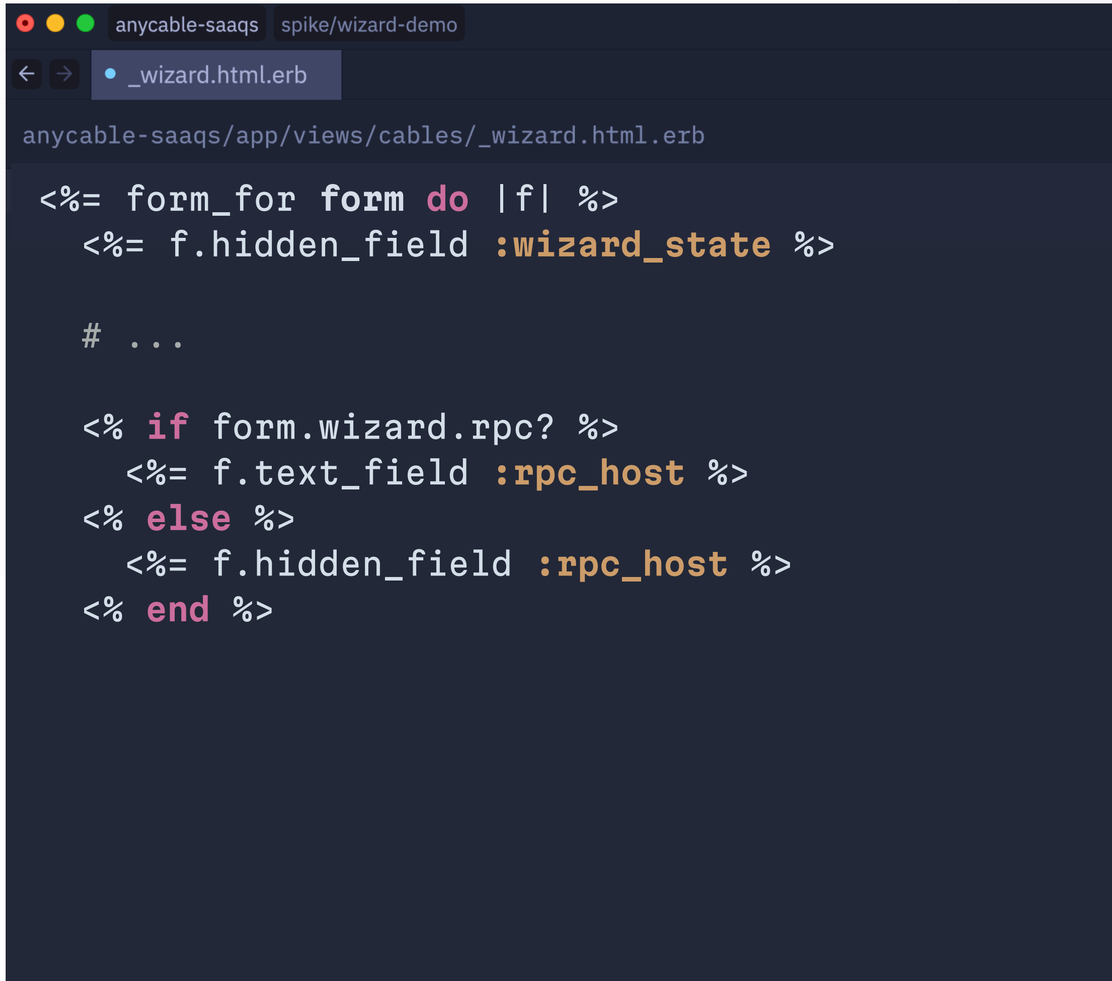
Then, run the installation script as follows:

```
bin/rails g anycable:setup --rpc=http
```

Follow the instructions in the terminal to complete the setup.

See more in the documentation.

Back Next



Cables / Create new cable

RPC URL

https://myapp.example.dev/api/anycable

Specify the AnyCable HTTP RPC endpoint of your Rails application, e.g., `https://<my-app-host>/_anycable`, where the path is the value you provided for the `http_rpc_mount_path` configuration parameter c
`anycable-rails` gem.

Learn more in the docum

NOTE: Feel free to skip t
configure these settings
haven't set up your app y
don't use Action Cable ch

Back Next

anycable-saaqs spike/wizard-demo

_wizard.html.erb

anycable-saaqs/app/views/cables/_wizard.html.erb

```
<%= form_for form do |f| %>
  <%= f.hidden_field :wizard_state %>

  # ...

  <% if form.wizard.secrets? %>
    <%= f.text_field :secret %>
  <% else %>
    <%= f.hidden_field :secret %>
  <% end %>
```

Cables / Create new cable

Application secret

Q-qiGlaxh0

Almost there! Here is your **application secret**.

Put it into your Rails credentials
(`anycable.secret`) or production environment
(`ANYCABLE_SECRET`) (docs).

This secret is used to sig
broadcasting requests an

If you use Hotwire Turbo
as the
`config.turbo.signed_`
value in your application

If you want to use AnyCa
authentication, no stream

Back Next

anycable-saaqs spike/wizard-demo

_wizard.html.erb

anycable-saaqs/app/views/cables/_wizard.html.erb

```erb
<%= form_for form do |f| %>
  <%= f.hidden_field :wizard_state %>

  # ...

  <% if form.wizard.can_complete? %>
   <%= f.submit "Create" %>
  <% else %>
   <%= f.submit "Next",
       formaction: new_cable_path %>
  <% end %>
  <% if form.wizard.can_back? %>
   <%= f.submit "Back",
     formaction: new_cable_path, value: "Back",
     name: "cable[wizard_action]" %>
  <% end %>
<% end %>
```


YOU DON'T NEED
TO BE A WIZARD TO
BUILD A WIZARD

THANK YOU

Vladimir Dementyev

Evil Martians

SF Bay Area Ruby #7