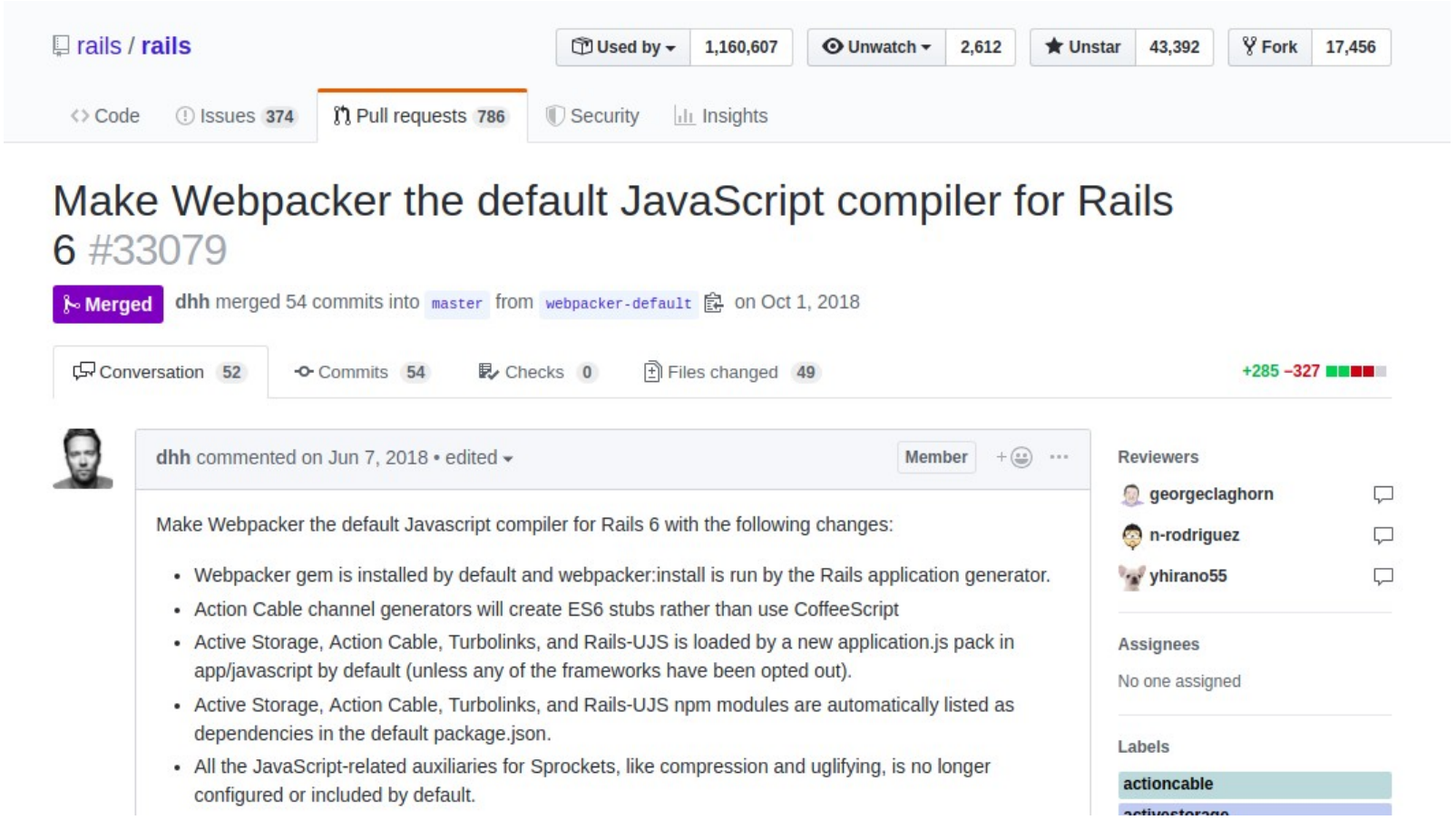
Modern Frontend in Rails

By Philip Lambok

Rails 6 Released!

Webpacker Sebagai default js compiler!



rails / rails

Used by 1,160,607 | Unwatch 2,612 | Unstar 43,392 | Fork 17,456

Code | Issues 374 | Pull requests 786 | Security | Insights

Make Webpacker the default JavaScript compiler for Rails 6 #33079

Merged **dhh** merged 54 commits into `master` from `webpacker-default` on Oct 1, 2018

Conversation 52 | Commits 54 | Checks 0 | Files changed 49 | +285 −327

dhh commented on Jun 7, 2018 • edited — Member

Make Webpacker the default Javascript compiler for Rails 6 with the following changes:

- Webpacker gem is installed by default and webpacker:install is run by the Rails application generator.
- Action Cable channel generators will create ES6 stubs rather than use CoffeeScript
- Active Storage, Action Cable, Turbolinks, and Rails-UJS is loaded by a new application.js pack in app/javascript by default (unless any of the frameworks have been opted out).
- Active Storage, Action Cable, Turbolinks, and Rails-UJS npm modules are automatically listed as dependencies in the default package.json.
- All the JavaScript-related auxiliaries for Sprockets, like compression and uglifying, is no longer configured or included by default.

Reviewers
georgeclaghorn
n-rodriguez
yhirano55

Assignees
No one assigned

Labels
actioncable

Before Webpacker (use assets pipeline)

```
/app
  /assets
    /stylesheets
    /javascripts
      application.js
```

With webpacker

```
/app
  /models
  /controllers
  /javascripts
    /packs
      application.js
    /new_course.vue
  /views
```

Apa itu webpacker?

- Webpacker makes it easy to use **the JavaScript pre-processor** and **bundler webpack** 4.x.x+ to manage application-like JavaScript in Rails.

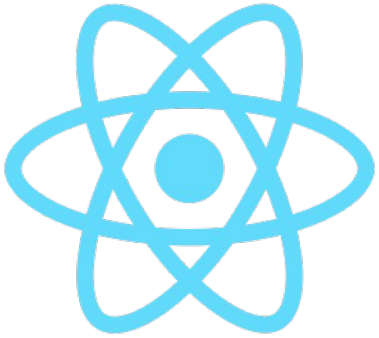
Apa itu webpack?

- Javascript autoloader
- Solusi atas masalah:
  - Ketika projek hanya punya 1 file js saja.
  - Membuat struktur tapi tidak scalling.

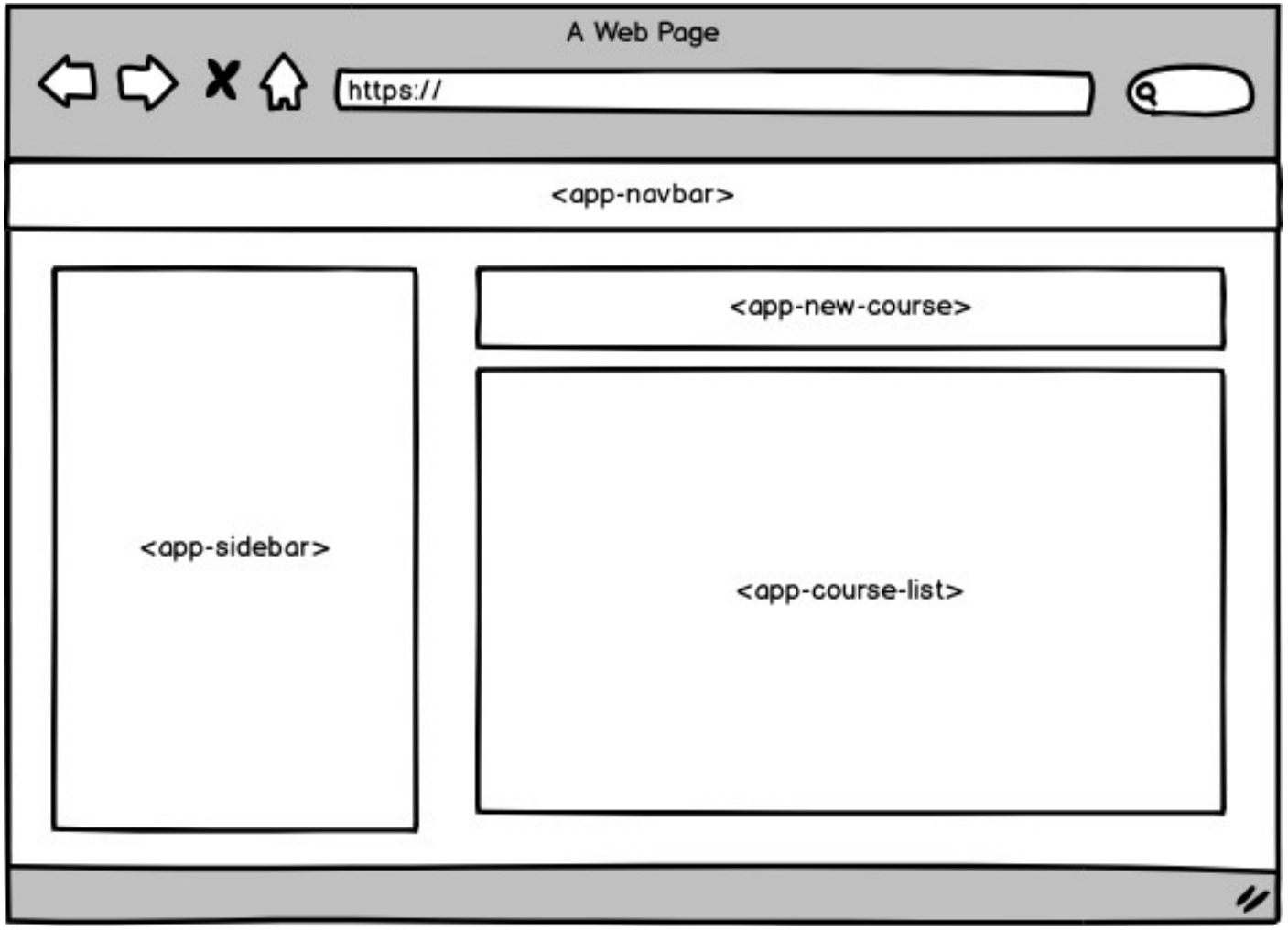
```
<head>
  <script src="dom_declaration.js"></script>
  <script src="course.js"><script>
  // ....
<head>
```

Javascript Framework

- Keuntungan pakai framework
  - Deliver fitur lebih cepat
  - Lebih mudah dalam menulis kode
  - Don’t repeat yourself
- Kerugian tanpa framework
  - Kode menjadi global
  - Sulit menamakan sesuatu
  - Tidak scalling



Web Component



- Kita membuat costum-tag-sendiri.
  - Kode menjadi terisolasi
  - Kode *reuseable* (mudah digunakan kembali)
  - Lebih *scalling*

```
<!-- default tag -->
<div></div>
<a href=""></a>

<!-- Kita buat -->
<new-course-form></new-course-form>
<show-course id="4"></show-course>
```

- Salah satu framework paling popular saat ini yang dibuat Evan You (Former Google Dev)

SPA VS Hybrid

JANUARY 29, 2019

You probably don’t need a single-page application

Hybrid solutions

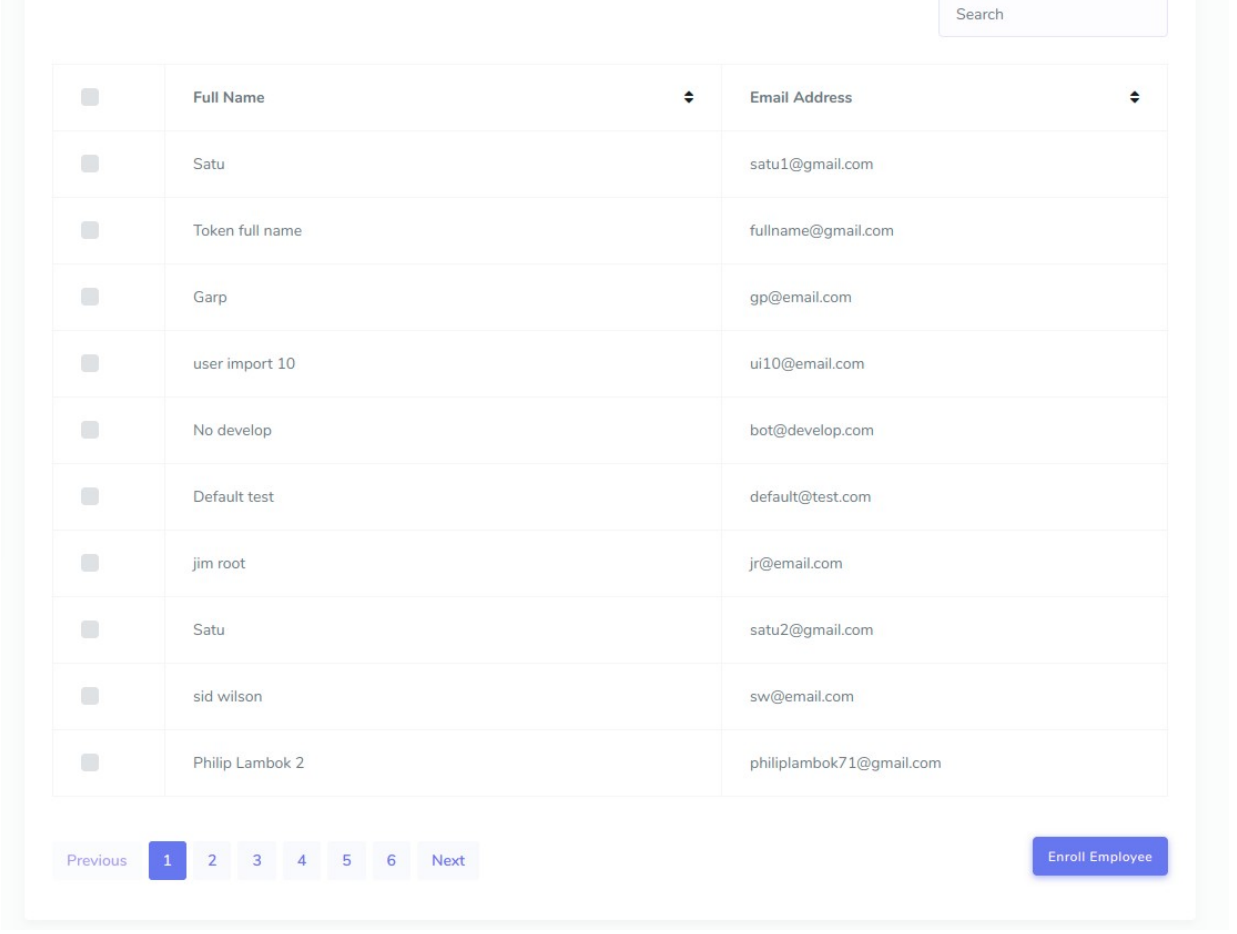
Even if your app needs some real-time capabilities or rich interactions, you don’t need to use the SPA paradigm for your whole app. A great approach is to embed small frontend applications into a traditional architecture.

Github uses this hybrid approach. The backbone of their website is a traditional rails app but some areas, like the projects tab, are built as embedded frontend applications. It’s a beautiful solution that combines the best of both worlds. The great thing is that you can start simple and gradually add more complex UI interactions with this approach.

Kapan kita harus gunakan ini?

- Permintaan client akan:
  - Validasi form ketika user mengetik
  - Bekerja dengan table yang dinamis (*rich feature*)

Kapan kita harus gunakan ini? (Lanj.)



Studi Kasus (CRUD)

- Instalasi

```
$ rails new sample_app --skip-coffee --skip-sprockets --skip-turbolinks --webpack=vue -T
```

```ruby
# frozen_string_literal: true

require 'rails_helper'

RSpec.describe 'Create Article', type: :system, js: true do
  it 'returns success response' do
    visit new_article_path
    fill_in :article_title, with: 'Use Repository Pattern in Active::Record'
    fill_in :article_body, with: 'Just use concern.'
    click_on 'Submit Article'
    expect(page).to have_content 'Article has been published'
  end
end
```

```ruby
# frozen_string_literal: true

Rails.application.routes.draw do
  resources :articles

  namespace :api do
    resources :articles
  end
end
```

```ruby
# frozen_string_literal: true

class ArticlesController < ApplicationController
  def new; end
end
```

```
<!-- /app/views/articles/new.html.erb -->

<div class="pb-2">
  <h5>Create new article</h5>
</div>

<article-form></article-form>
```

```vue
<!-- app/javascript/article-form.vue -->

<template>
  <div id="article-form">
    Hello, {{ message }}
  <div>
</template>

<script>
export default {
  data() {
    return {
      message: "World!"
    };
  }
};
</script>

<style scoped>
 div {
   color: deeppink;
 }
</style>
```

```
<!-- app/javascript/article-form.vue -->

<template>
  <div id="article-form">
    <div v-show="successMessage" class="alert alert-success">{{ successMessage }}</div>
    <div class="form-group row">
      <label for="article_title" class="col-form-label col-sm-2">Title</label>
      <div class="col-md-10">
        <input type="text" id="article_title" class="form-control" name="article[title]"v-model="title">
      </div>
    </div>

    <div class="form-group row">
      <label for="article_body" class="col-form-label col-sm-2">Body</label>
      <div class="col-md-10">
        <textarea id="article_body" class="form-control" cols="30"  rows="10" v-model="body"></textarea>
      </div>
    </div>

    <div class="form-group row">
      <div class="col-md-10 ml-auto">
        <button class="btn btn-primary px-4" @click="submitArticle()">
          Submit Article
        </button>
      </div>
    </div>
  </div>
</template>
```

```
<!-- app/javascript/article-form.vue -->

 <!-- // .... -->
</template>

<script>
export default {
  data() {
    return {
      title: "",
      body: "",
      successMessage: null,
    };
  },
  methods: {
    submitArticle() {
      fetch("/api/articles", {
        method: "POST",
        headers: {
          "Content-Type": "application/json"
        },
        body: JSON.stringify(article: { title: this.title, body: this.body })
      })
        .then(response => response.json())
        .then(data => {
          this.title = "";
          this.body = "";
          this.successMessage = data.message;
        });
    }
  },
};
</script>
```

```ruby
# frozen_string_literal: true

module Api
  class ArticlesController < ApplicationController
    def create
      article = Article.new(article_params)
      article.save
      render json: { message: 'Article has been published' }
    end

    private

    def article_params
      params.require(:article).permit(:title, :body)
    end
  end
end
```

```
import Vue from "vue/dist/vue.esm";
import ArticleForm from "../article_form.vue";

document.addEventListener("DOMContentLoaded", () => {
  const app = new Vue({
    el: "#app",
    data: {},
    components: { ArticleForm }
  });
});
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Sample CRUD Vue-Rails</title>
    <%= csrf_meta_tags %>
    <%= csp_meta_tag %>

    <%= stylesheet_link_tag    'application', media: 'all' %>
    <%= javascript_include_tag 'application' %>
    <%= javascript_pack_tag 'application' %>
  </head>

  <body>
    <div id="app" class="container my-4">
      <%= yield %>
    </div>
  </body>
</html>
```

Back to article lists

Create new article

Title: Title of article

Body: Body of article

Submit Article

```ruby
# frozen_string_literal: true

require 'rails_helper'

RSpec.describe 'Article Index', type: :system, js: true do
  before do
    Article.create(title: 'Intro to Ruby', body: 'body of ruby')
    Article.create(title: 'Intro to Vue', body: 'body of vue')
  end

  it 'returns list of articles' do
    visit articles_path
    expect(page).to have_content 'Intro to Ruby'
    expect(page).to have_content 'body of ruby'
    expect(page).to have_content 'Intro to Vue'
    expect(page).to have_content 'body of vue'
  end
end
```

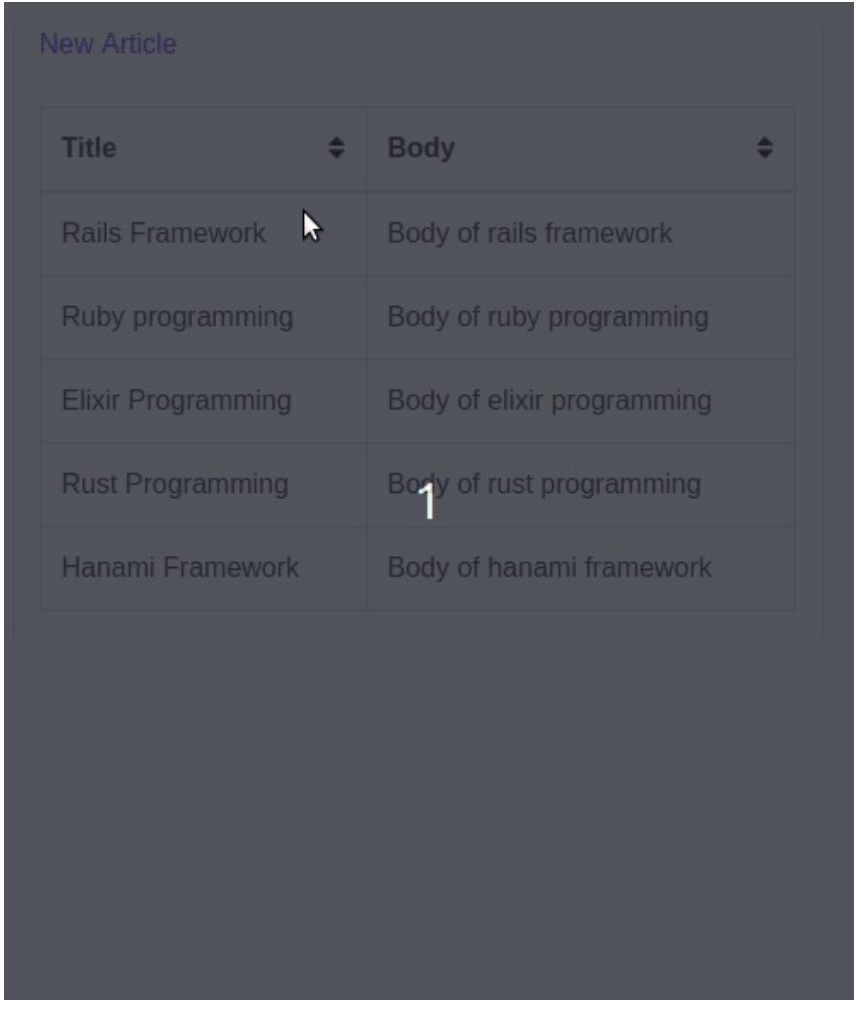
```ruby
# frozen_string_literal: true

class ArticlesController < ApplicationController
  def index; end

  def new; end
end
```

```erb
<!-- app/views/articles/index.html.erb -->

<div class="mb-4">
  <%= link_to 'New Article', new_article_path %>
</div>

<article-list></article-list>
```

```
<!--- app/javascript/article-list.vue -->

<template>
  <div id="article-list">
    <table class="table table-bordered">
      <thead>
        <tr>
          <th>
            <div class="d-flex justify-content-between align-items-center">
              <span>Title</span>
              <a href="#" @click="sortByTitle()">
                <i class="fa fa-sort"></i>
              </a>
            </div>
          </th>
          <th>
            <div class="d-flex justify-content-between align-items-center">
              <span>Body</span>
              <a href="#" @click="sortByBody()">
                <i class="fa fa-sort"></i>
              </a>
            </div>
          </th>
        </tr>
      </thead>
      <tbody>
        <tr :key="article.id" v-for="article in articles" :id="article.id">
          <td>{{ article.title }}</td>
          <td>{{ article.body }}</td>
        </tr>
      </tbody>
    </table>
  </div>
</template>
```

```
<script>
export default {
  data() {
    return {
      articles: [],
    };
  },
  methods: {
    sortByTitle() {
      // ..
    },

    sortByBody() {
     // ..
    },
  },
  mounted() {
    fetch("/api/articles")
      .then(response => response.json())
      .then(data => {
        this.articles = data;
      });
  }
};
</script>
```

```
import Vue from "vue/dist/vue.esm";
import ArticleForm from "../article_form.vue";
import ArticleList from "../article_list.vue";

document.addEventListener("DOMContentLoaded", () => {
  const app = new Vue({
    el: "#app",
    data: {},
    components: { ArticleForm, ArticleList }
  });
});
```


New Article
Title
Body
Rails Framework
Body of rails framework
Ruby programming
Body of ruby programming
Elixir Programming
Body of elixir programming
Rust Programming
Body of rust programming
1
Hanami Framework
Body of hanami framework

```ruby
# frozen_string_literal: true

require 'rails_helper'

RSpec.describe 'Update Article', type: :system, js: true do
  before do
    Article.create(title: 'Ruby Article', body: 'Text body')
  end

  it 'returns success message' do
    article = Article.find_by(title: 'Ruby Article')
    visit edit_article_path(article)
    fill_in :article_title, with: 'Ruby Article updated'
    fill_in :article_body, with: 'Text body updated'
    click_on 'Update Article'
    expect(page).to have_content 'Article has been updated'
  end
end
```

```ruby
# frozen_string_literal: true

class ArticlesController < ApplicationController
  def index; end

  def new; end

  def edit
    @article = Article.find_by(id: params[:id])
  end
end
```

```erb
<!-- app/views/articles/edit.html.erb -->

<div class="mb-4 text-right">
  <%= link_to 'Back to article lists', articles_path %>
</div>

<div class="pb-2">
  <h5>Update Article</h5>
</div>

<article-form id="<%= @article.id %>"></article-form>
```

```
<script>
export default {
  props: ["id"],
  data() {
    // ...
  },
  methods: {
    // ...
  },
  mounted() {
    if (this.id) {
      fetch(`/api/articles/${this.id}`)
        .then(response => response.json())
        .then(data => {
          this.title = data.title;
          this.body = data.body;
        });
    }
  }
};
</script>
```

```ruby
# frozen_string_literal: true

module Api
  class ArticlesController < ApplicationController
    # ..

    def show
      article = find_article(params[:id])
      render json: article
    end

    # ...

    private

    # ...

    def find_article(id)
      Article.find_by(id: id)
    end
  end
end
```

```
<div class="form-group row">
  <div class="col-md-10 ml-auto">
    <button class="btn btn-primary px-4" @click="submitArticle()" v-if="!this.id">Submit Article</button>
    <button class="btn btn-primary px-4" @click="updateArticle()" v-if="this.id">Update Article</button>
  </div>
</div>
```

```
<script>
export default {
  props: ["id"],
  data() {
    // ...
  },
  methods: {
    //...
    updateArticle() {
      fetch(`/api/articles/${this.id}`, {
        method: "PUT",
        headers: {
          "Content-type": "application/json"
        },
        body: JSON.stringify({article: {title: this.title, body: this.body })
      })
        .then(response => response.json())
        .then(data => {
          this.successMessage = data.message;
        });
    },
    // ...
  },
  mounted() {
    // ...
  }
};
</script>
```

```ruby
# frozen_string_literal: true

module Api
  class ArticlesController < ApplicationController
    # ...

    def update
      article = find_article(params[:id])
      article.update(article_params)
      render json: { message: 'Article has been updated' }
    end

    # ..

    private

    def article_params
      params.require(:article).permit(:title, :body)
    end

    # ..
  end
end
```

New Article

Title	Body	Action
Rails Framework	Body of rails framework	Edit
Ruby programming	Body of ruby programming	Edit
Elixir Programming	Body of elixir programming	Edit
Rust Programming	Body of rust programming	Edit
Hanami Framework	Body of hanami framework	Edit

```ruby
require 'rails_helper'

RSpec.describe 'Destroy Article', type: :system, js: true do
  before do
    Article.create(title: 'Sample')
  end

  it 'destroy the article' do
    visit articles_path
    click_on 'Delete'
    expect(page).to_not have_content 'Sample'
  end
end
```

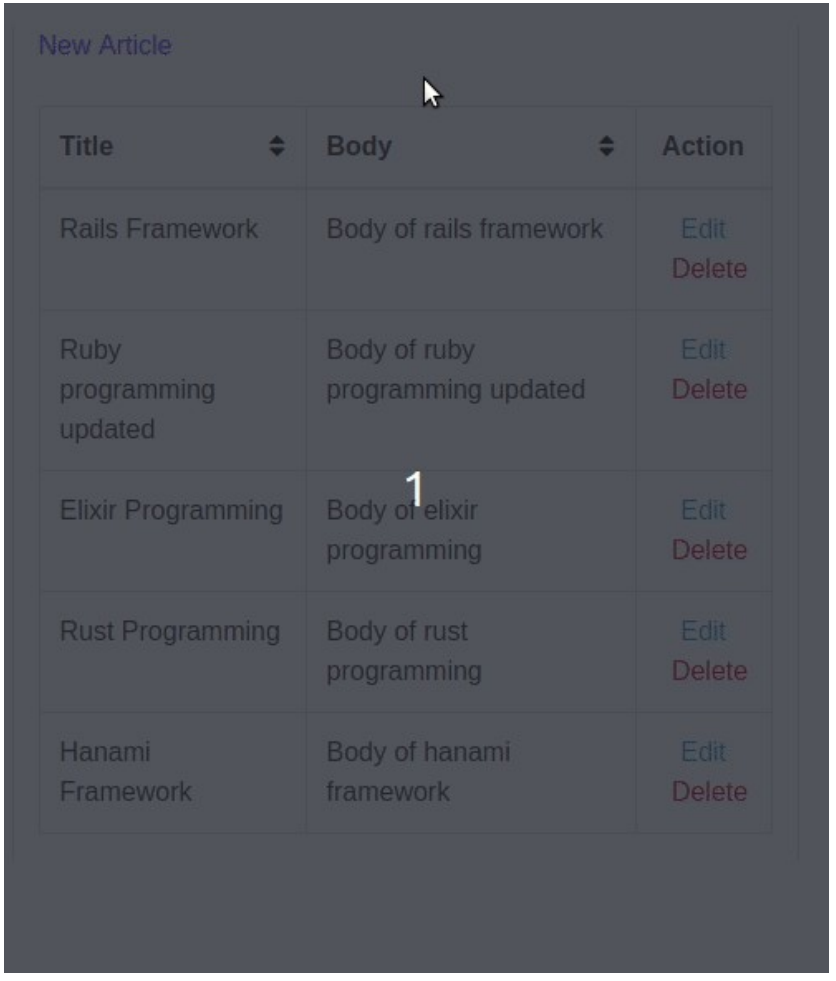
app/javascript/article_list.vue

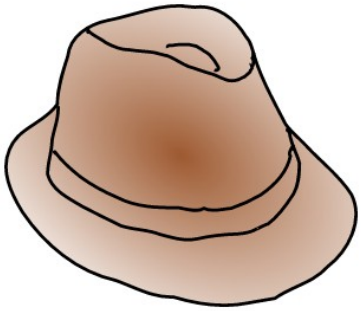
```
<tr :key="article.id" v-for="article in articles" :id="article.id">
  <td>{{ article.title }}</td>
  <td>{{ article.body }}</td>
  <td class="text-center">
    <a :href="editLink(article)" class="text-info">Edit</a>
    <a href="#" class="text-danger pl-2" @click.prevent="deleteArticle(article)">Delete</a>
  </td>
</tr>
```

app/javascript/article_list.vue

```
<script>
export default {
  data() {
   // ...
  },
  methods: {
    // ...
    deleteArticle(article) {
      fetch(`/api/articles/${article.id}`, {
        method: "DELETE",
        headers: {
          "Content-type": "application/json"
        },
        body: null
      })
        .then(response => response.json())
        .then(data => {
          this.articles = data.articles;
        });
    }
  },
  // ...
};
</script>
```

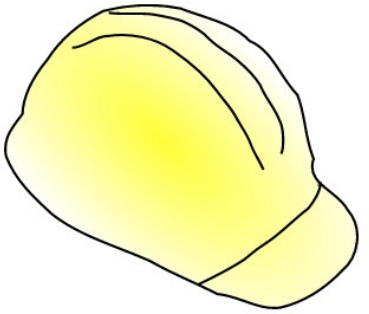
```ruby
# frozen_string_literal: true

module Api
  class ArticlesController < ApplicationController
    # ...
    def destroy
      article = find_article(params[:id])
      article.destroy
      render json: { articles: Article.all }
    end
    # ...
  end
end
```


New Article
Title
Body
Action
Rails Framework
Body of rails framework
Edit
Delete
Ruby programming updated
Body of ruby programming updated
Edit
Delete
Elixir Programming
Body of elixir programming
Edit
Delete
Rust Programming
Body of rust programming
Edit
Delete
Hanami Framework
Body of hanami framework
Edit
Delete
1



Refactoring

When refactoring every change you make is a small behavior-preserving change. You only refactor with green tests, and any test failing indicates a mistake. By stringing together a series of small changes like this you can move more quickly and with less risk because you shouldn't get trapped in debugging.



Adding Function

Any other change to the code is adding function. You will add new tests and break existing tests. You aren't confined to behavior-preserving changes (but it's wise to keep changes small and return to green tests swiftly).

During programming you may swap frequently between hats, perhaps every couple of minutes. But...

You can only wear one hat at a time

app/javascript/models/article.js

```
export default class Article {
  constructor(title, body) {
    this.title = title;
    this.body = body;
  }

  save() {
    // ...
  }

  destroy(){
  }

  // ...
}
```

app/javascript/article-form.vue

```
<script>
import Article from "models/article";
export default {
  props: ["id"],
  data() {
    return {
      article: new Article("", ""),
      successMessage: null,
      errors: []
    };
  },
  methods: {
    submitArticle() {
      if(this.article.save()){
        // ....
      }
    }
  },
  mounted() {
    // ...
  }
};
</script>
```

Don't use javascript if it's not needed

Bonus

Introduce support for ActionView::Component #36388

Merged **tenderlove** merged 2 commits into `rails:master` from `joelhawksley:actionview-component` 9 days ago

Conversation 55 | Commits 2 | Checks 0 | Files changed 4 | +72 −1

joelhawksley commented 19 days ago • edited — Contributor

Introduce support for ActionView::Component

This PR introduces structural support for `ActionView::Component`, a framework for building view components.

Specifically, it modifies `ActionView::RenderingHelper#render` to support passing in an object to `render` that `responds_to` `render_in`, enabling us to build view components as objects in Rails.

We've been running a variant of this patch in production at GitHub since March, and now have about a dozen components used in over a hundred call sites.

The PR includes an example component (`TestComponent`) that closely resembles the base component we're using at GitHub.

Reviewers

- eileencodes
- skyksandr
- tenderlove ✓

Assignees

No one assigned

Labels

actionview

Projects

Example

Given the component `app/components/test_component.rb`:

```
class TestComponent < ActionView::Component
  validates :content, :title, presence: true

  def initialize(title:)
    @title = title
  end

  def self.template
    <<~'erb'
    <span title="<%= title %>"><%= content %></span>
    erb
  end

  private

  attr_reader :title
end
```

We can render it in a view as:

```
<%= render(TestComponent.new(title: "my title")) do %>
  Hello, World!
<% end %>
```

Which returns:

```
<span title="my title">Hello, World!</span>
```

Thanks

Source code: https://github.com/philiplambok/crud-vue-rails