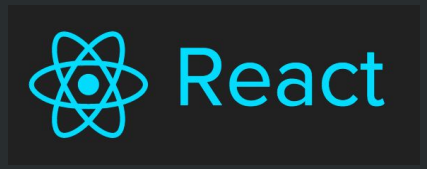
Browser JavaScript Structure

So far, we've been very meticulous with **organizing** our Ruby code.

Rails obviously has plenty of opinions in that regard.

If we wanted to arrange **JavaScript** with the same rigor, what would that look like?

It depends on who you ask...



The JavaScript Framework
I Invented Yesterday

There’s is no right answer. There’s no best way.

What's important is that you **give it some thought**, just like we do with Ruby.

Organizing your JavaScript

Rather than get into a JavaScript framework holy war, let's try to apply a **simple methodology** of organizing our JavaScript.

Organizing your JavaScript

A methodology that will work well for our Rails apps and perhaps other apps as well.

Components

The goal in organizing code is **splitting it across smaller files** in a way that makes sense.

Components

Each file needs to have a small responsibility to keep things organized.

Components

Let’s break up the JavaScript side of our application into small **components**.

Components

A **component** is just a small piece of functionality in our application.

Components

The concept of a **component** is fuzzy, but it's also flexible. We can choose what we want a **component** to be.

Components

It works well for a Rails application because we will mostly be adding **small pieces of JavaScript** to **enhance** our apps.

Components

Most of the code will still be on the Rails side rather than the JavaScript side.

The Pokemon project

Throughout this unit, we will be working on a ready-made Rails application all about Pokemon.

The Pokemon project

We will be making use of the Pokemon API (pokeapi.co) for all the data.

No need for our own database models.

The Pokemon project

Clone this repo

Don’t forget to `bundle install`.

The Pokemon project

This project has a single view in Rails that shows all of the pokemon in a list.

Fun fact: the **plural** of pokemon is **pokemon**.

The Pokemon project

So far this view comes from the backend completely.

Let's implement some **JavaScript features**.

The Pokemon project

The idea is that we use JavaScript for **small interactions** that we wouldn't want to refresh the entire page for.

Rails JavaScript setup

We've got a little bit of setup that we only need to do once though.

With the setup, we will also talk a little bit about how Rails does JavaScript.

Rails JavaScript setup

In Rails, the most important JavaScript file is the `application.js`.

It can be found in: `app/assets/javascripts/`.

Rails JavaScript setup

The Rails asset pipeline uses the `application.js` file to kick off all the JavaScript in the project.

Rails JavaScript setup

The important bit is some special comments:

```
//= require jquery
//= require jquery_ujs
//= require twitter/bootstrap
//= require_tree .
```

Rails JavaScript setup

Notice the equal signs:

```
//= require jquery
//= require jquery_ujs
//= require twitter/bootstrap
//= require_tree .
```

Rails JavaScript setup

The asset pipeline actually uses these to pull code into your application.

```
//= require jquery
//= require jquery_ujs
//= require twitter/bootstrap
//= require_tree .
```

Rails JavaScript setup

The inclusion of jQuery comes with every Rails application.

```
//= require jquery
//= require jquery_ujs
//= require twitter/bootstrap
//= require_tree .
```

Rails JavaScript setup

We've added the Bootstrap JavaScript.

```
//= require jquery
//= require jquery_ujs
//= require twitter/bootstrap
//= require_tree .
```

Rails JavaScript setup

Finally, this magical line includes all the JavaScript files that you add to the project:

```
//= require jquery
//= require jquery_ujs
//= require twitter/bootstrap
//= require_tree .
```

Rails JavaScript setup

The other ones in `app/assets/javascripts/`.

```
//= require jquery
//= require jquery_ujs
//= require twitter/bootstrap
//= require_tree .
```

Rails JavaScript setup

Let's add another now: an `init.js` to set up our application's JavaScript.

```
//= require jquery
//= require jquery_ujs
//= require twitter/bootstrap
//= require init
//= require_tree .
```

Rails JavaScript setup

If it's included automatically, why add it to the `application.js`?

```
//= require jquery
//= require jquery_ujs
//= require twitter/bootstrap
//= require init
//= require_tree .
```

Rails JavaScript setup

Because we want to make sure that it's the **first to run** of all the JavaScript that we write.

```
//= require jquery
//= require jquery_ujs
//= require twitter/bootstrap
//= require init
//= require_tree .
```

IRON HACK

Rails JavaScript setup

Now create `app/assets/javascripts/init.js` with this code:

```
if (window.PokemonApp === undefined) {
  window.PokemonApp = {};
}

PokemonApp.init = function () {
  // 3rd party setup code here
  console.log("Pokemon App ONLINE!");
};

$(document).on("ready", function () {
  PokemonApp.init();
});
```

Rails JavaScript setup

This creates the object that will contain all our components, PokemonApp.

```
if (window.PokemonApp === undefined) {
  window.PokemonApp = {};
}

PokemonApp.init = function () {
  // 3rd party setup code here
  console.log("Pokemon App ONLINE!");
};

$(document).on("ready", function () {
  PokemonApp.init();
});
```

IRON HACK

Rails JavaScript setup

We'll use this object hold all our classes and functions to protect them from other coders.

```
if (window.PokemonApp === undefined) {
  window.PokemonApp = {};
}

PokemonApp.init = function () {
  // 3rd party setup code here
  console.log("Pokemon App ONLINE!");
};

$(document).on("ready", function () {
  PokemonApp.init();
});
```

Rails JavaScript setup

This executes the functionality of our jQuery code only when all elements have been loaded.

```
if (window.PokemonApp === undefined) {
  window.PokemonApp = {};
}

PokemonApp.init = function () {
  // 3rd party setup code here
  console.log("Pokemon App ONLINE!");
};

$(document).on("ready", function () {
  PokemonApp.init();
});
```

Rails JavaScript setup

Most of our components will have a call like this for any DOM setup they need.

```
if (window.PokemonApp === undefined) {
  window.PokemonApp = {};
}

PokemonApp.init = function () {
  // 3rd party setup code here
  console.log("Pokemon App ONLINE!");
};

$(document).on("ready", function () {
  PokemonApp.init();
});
```

Rails JavaScript setup

Things like click handlers will go in here.

```
if (window.PokemonApp === undefined) {
  window.PokemonApp = {};
}

PokemonApp.init = function () {
  // 3rd party setup code here
  console.log("Pokemon App ONLINE!");
};

$(document).on("ready", function () {
  PokemonApp.init();
});
```

Rails JavaScript setup

Finally, we call the `PokemonApp.init()` function we've defined.

```
if (window.PokemonApp === undefined) {
  window.PokemonApp = {};
}

PokemonApp.init = function () {
  // 3rd party setup code here
  console.log("Pokemon App ONLINE!");
};

$(document).on("ready", function () {
  PokemonApp.init();
});
```

Rails JavaScript setup

It isn't doing anything right now but, use it to set up any JavaScript you need on **every page**.

```
if (window.PokemonApp === undefined) {
  window.PokemonApp = {};
}

PokemonApp.init = function () {
  // 3rd party setup code here
  console.log("Pokemon App ONLINE!");
};

$(document).on("ready", function () {
  PokemonApp.init();
});
```

Rails JavaScript setup

Note that all our components are going to have a similar structure to `init.js`.

```
Object or class
definition
```

```
Use of object or class in
$(document).on("ready")
```

JS Feature #1: Pokemon details

First, let's make the names of the pokemon **clickable**.

When you click the name of a pokemon, we should see **a box with more details**.

Pokemon details

Our first component will be the `Pokemon` component.

It will handle the click of the pokemon's name, retrieve the pokemon's information and display it in a Bootstrap modal.

Pokemon details

Add this class definition to `app/assets/javascripts/pokemon.js`:

```
PokemonApp.Pokemon = function (pokemonUri) {
  this.id = PokemonApp.Pokemon.idFromUri(pokemonUri);
};

PokemonApp.Pokemon.prototype.render = function () {
  console.log("Rendering pokemon: #" + this.id);
};

PokemonApp.Pokemon.idFromUri = function (pokemonUri) {
  var uriSegments = pokemonUri.split("/");
  var secondLast = uriSegments.length - 2;
  return uriSegments[secondLast];
};
```

Pokemon details

Remember a “class” in JavaScript is defined with a function.

```
PokemonApp.Pokemon = function (pokemonUri) {
  this.id = PokemonApp.Pokemon.idFromUri(pokemonUri);
};

PokemonApp.Pokemon.prototype.render = function () {
  console.log("Rendering pokemon: #" + this.id);
};

PokemonApp.Pokemon.idFromUri = function (pokemonUri) {
  var uriSegments = pokemonUri.split("/");
  var secondLast = uriSegments.length - 2;
  return uriSegments[secondLast];
};
```

Pokemon details

Notice how we are adding our component class to the PokemonApp object.

```
PokemonApp.Pokemon = function (pokemonUri) {
  this.id = PokemonApp.Pokemon.idFromUri(pokemonUri);
};

PokemonApp.Pokemon.prototype.render = function () {
  console.log("Rendering pokemon: #" + this.id);
};

PokemonApp.Pokemon.idFromUri = function (pokemonUri) {
  var uriSegments = pokemonUri.split("/");
  var secondLast = uriSegments.length - 2;
  return uriSegments[secondLast];
};
```

Pokemon details

The `Pokemon.idFromUri()` function extracts the pokemon's id from the URI.

```
PokemonApp.Pokemon = function (pokemonUri) {
  this.id = PokemonApp.Pokemon.idFromUri(pokemonUri);
};

PokemonApp.Pokemon.prototype.render = function () {
  console.log("Rendering pokemon: #" + this.id);
};

PokemonApp.Pokemon.idFromUri = function (pokemonUri) {
  var uriSegments = pokemonUri.split("/");
  var secondLast = uriSegments.length - 2;
  return uriSegments[secondLast];
};
```

Pokemon details

The `.render()` method will display the pokemon box. We will circle back to it later.

```
PokemonApp.Pokemon = function (pokemonUri) {
  this.id = PokemonApp.Pokemon.idFromUri(pokemonUri);
};

PokemonApp.Pokemon.prototype.render = function () {
  console.log("Rendering pokemon: #" + this.id);
};

PokemonApp.Pokemon.idFromUri = function (pokemonUri) {
  var uriSegments = pokemonUri.split("/");
  var secondLast = uriSegments.length - 2;
  return uriSegments[secondLast];
};
```

Pokemon details

In general, `.render()` is a good name for a component method that displays content.

```
PokemonApp.Pokemon = function (pokemonUri) {
  this.id = PokemonApp.Pokemon.idFromUri(pokemonUri);
};

PokemonApp.Pokemon.prototype.render = function () {
  console.log("Rendering pokemon: #" + this.id);
};

PokemonApp.Pokemon.idFromUri = function (pokemonUri) {
  var uriSegments = pokemonUri.split("/");
  var secondLast = uriSegments.length - 2;
  return uriSegments[secondLast];
};
```

Pokemon details

Remember, in JavaScript methods are defined on the class' prototype.

```
PokemonApp.Pokemon = function (pokemonUri) {
  this.id = PokemonApp.Pokemon.idFromUri(pokemonUri);
};

PokemonApp.Pokemon.prototype.render = function () {
  console.log("Rendering pokemon: #" + this.id);
};

PokemonApp.Pokemon.idFromUri = function (pokemonUri) {
  var uriSegments = pokemonUri.split("/");
  var secondLast = uriSegments.length - 2;
  return uriSegments[secondLast];
};
```

Pokemon details

Now for use of our component class. Add this to `app/assets/javascripts/pokemon.js`:

```
$(document).on("ready", function () {

  $(".js-show-pokemon").on("click", function (event) {
    var $button = $(event.currentTarget);
    var pokemonUri = $button.data("pokemon-uri");

    var pokemon = new PokemonApp.Pokemon(pokemonUri);
    pokemon.render();
  });

});
```

IRON HACK

Pokemon details

Here we make use of our component class.

```
$(document).on("ready", function () {

  $(".js-show-pokemon").on("click", function (event) {
    var $button = $(event.currentTarget);
    var pokemonUri = $button.data("pokemon-uri");

    var pokemon = new PokemonApp.Pokemon(pokemonUri);
    pokemon.render();
  });

});
```

Pokemon details

And we call the `.render()` method on the instance of the component.

```
$(document).on("ready", function () {

  $(".js-show-pokemon").on("click", function (event) {
    var $button = $(event.currentTarget);
    var pokemonUri = $button.data("pokemon-uri");

    var pokemon = new PokemonApp.Pokemon(pokemonUri);
    pokemon.render();
  });

});
```

Pokemon details

Each individual pokemon element also has a data attribute with information specific to it.

```
$(document).on("ready", function () {

  $(".js-show-pokemon").on("click", function (event) {
    var $button = $(event.currentTarget);
    var pokemonUri = $button.data("pokemon-uri");

    var pokemon = new PokemonApp.Pokemon(pokemonUri);
    pokemon.render();
  });

});
```

Pokemon details

Circling back to `.render()` now, we want it to display the pokemon's information.

```
PokemonApp.Pokemon.prototype.render = function () {
  console.log("Rendering pokemon: #" + this.id);
};
```

Pokemon details

All we have right now is the pokemon's id from the data attribute.

```
PokemonApp.Pokemon.prototype.render = function () {
  console.log("Rendering pokemon: #" + this.id);
};
```

Pokemon details

We can use that id to get the rest of the details from the API. AJAX time!

```
PokemonApp.Pokemon.prototype.render = function () {
  console.log("Rendering pokemon: #" + this.id);

  $.ajax({
    url: "/api/pokemon/" + this.id,
    success: function (response) {
      console.log("Pokemon info:");
      console.log(response);
    }
  });
};
```

Pokemon details

We've already got the API set up via Rails.

```
PokemonApp.Pokemon.prototype.render = function () {
  console.log("Rendering pokemon: #" + this.id);

  $.ajax({
    url: "/api/pokemon/" + this.id,
    success: function (response) {
      console.log("Pokemon info:");
      console.log(response);
    }
  });
};
```

Pokemon details

Run it once and see what the structure of the object is in the console.

```
PokemonApp.Pokemon.prototype.render = function () {
  console.log("Rendering pokemon: #" + this.id);

  $.ajax({
    url: "/api/pokemon/" + this.id,
    success: function (response) {
      console.log("Pokemon info:");
      console.log(response);
    }
  });
};
```

Pokemon details

Let's also save the response in the component for future use.

```
PokemonApp.Pokemon.prototype.render = function () {
  console.log("Rendering pokemon: #" + this.id);

  var self = this;

  $.ajax({
    url: "/api/pokemon/" + this.id,
    success: function (response) {
      self.info = response;

      console.log("Pokemon info:");
      console.log(self.info);
    }
  });
};
```

Pokemon details

Now we need to show the box with the pokemon's info. Let's see the box first.

```
PokemonApp.Pokemon.prototype.render = function () {
  console.log("Rendering pokemon: #" + this.id);

  var self = this;

  $.ajax({
    url: "/api/pokemon/" + this.id,
    success: function (response) {
      self.info = response;

      console.log("Pokemon info:");
      console.log(self.info);
    }
  });
};
```

Pokemon details

We've conveniently placed a Bootstrap modal to `app/views/pokemon/index.html.erb`.

```
<div class="modal fade">
  <div class="modal-dialog">
    <div class="modal-content">

      <div class="modal-header">
        <button class="close" data-dismiss="modal">
          <span>&times;</span>
        </button>

        <h3 class="modal-title">
          <span>Feraligtr</span>
          <!-- [...] -->
```

Pokemon details

This line shows the box.

Here's just the AJAX part, for brevity.

```
$.ajax({
   url: "/api/pokemon/" + this.id,
   success: function (response) {
     self.info = response;

     console.log("Pokemon info:");
     console.log(self.info);

     $(".js-pokemon-modal").modal("show");
   }
 });
```

Pokemon details

What do we need to change for this to work?

```
$.ajax({
   url: "/api/pokemon/" + this.id,
   success: function (response) {
     self.info = response;

     console.log("Pokemon info:");
     console.log(self.info);

     $(".js-pokemon-modal").modal("show");
   }
 });
```

Pokemon details

The modal needs to right selector!

```
<div class="modal fade js-pokemon-modal">
  <div class="modal-dialog">
    <div class="modal-content">

      <div class="modal-header">
        <button class="close" data-dismiss="modal">
          <span>&times;</span>
        </button>

        <h3 class="modal-title">
          <span>Feraligtr</span>
          <!-- [...] -->
```

Pokemon details

So we've got 4 properties in the modal: name, number, height and weight.

Pokemon details

After looking at the API response in the console, we can see how to access them.

```
$.ajax({
   url: "/api/pokemon/" + this.id,
   success: function (response) {
     self.info = response;

     console.log("Pokemon name: " + self.info.name);
     console.log("Pokemon number: " + self.info.pkdx_id);
     console.log("Pokemon height: " + self.info.height);
     console.log("Pokemon weight: " + self.info.weight);

     $(".js-pokemon-modal").modal("show");
   }
 });
```

IRON HACK

Pokemon details

Taking a look at our HTML, let's identify the elements we want to update and add selectors.

```
<div class="modal-header">
   <button class="close"
       data-dismiss="modal">
     <span>&times;</span>
   </button>

   <h3 class="modal-title">
     <span>Feraligtr</span>
     <small>#160</small>
   </h3>
 </div><!-- .modal-header -->
```

```
<div class="modal-body">
  <dl class="dl-horizontal">
    <dt>Height</dt>
    <dd>23</dd>

    <dt>Weight</dt>
    <dd>888</dd>
  </dl>
</div><!-- .modal-body -->
```

Pokemon details

Taking a look at our HTML, let's identify the elements we want to update and add selectors.

```
<div class="modal-header">
   <button class="close"
       data-dismiss="modal">
     <span>&times;</span>
   </button>

   <h3 class="modal-title">
     <span class="js-pkmn-name">
       Feraligtr
     </span>
     <small class="js-pkmn-number">
       #160
     </small>
   </h3>
 </div><!-- .modal-header -->
```

```
<div class="modal-body">
  <dl class="dl-horizontal">
    <dt>Height</dt>
    <dd class="js-pkmn-height">23</dd>

    <dt>Weight</dt>
    <dd class="js-pkmn-weight">888</dd>
  </dl>
</div><!-- .modal-body -->
```

Pokemon details

Now let's use the `.text()` method to update them!

```
$.ajax({
   url: "/api/pokemon/" + this.id,
   success: function (response) {
     self.info = response;

     $(".js-pkmn-name").text(self.info.name);
     $(".js-pkmn-number").text(self.info.pkdx_id);
     $(".js-pkmn-height").text(self.info.height);
     $(".js-pkmn-weight").text(self.info.weight);

     $(".js-pokemon-modal").modal("show");
   }
 });
```

Pokemon details

Notice the class selectors match the HTML.

```
$.ajax({
   url: "/api/pokemon/" + this.id,
   success: function (response) {
     self.info = response;

     $(".js-pkmn-name").text(self.info.name);
     $(".js-pkmn-number").text(self.info.pkdx_id);
     $(".js-pkmn-height").text(self.info.height);
     $(".js-pkmn-weight").text(self.info.weight);

     $(".js-pokemon-modal").modal("show");
   }
 });
```

Pokemon details: final

```
PokemonApp.Pokemon.prototype.render = function () {
  console.log("Rendering pokemon: #" + this.id);

  var self = this;

  $.ajax({
    url: "/api/pokemon/" + this.id,
    success: function (response) {
      self.info = response;

      $(".js-pkmn-name").text(self.info.name);
      $(".js-pkmn-number").text(self.info.pkdx_id);
      $(".js-pkmn-height").text(self.info.height);
      $(".js-pkmn-weight").text(self.info.weight);

      $(".js-pokemon-modal").modal("show");
    }
  });
};
```

IRON HACK

Exercise

Get your Pokemon box working and add additional information to it.

- Stats
  - HP (hit points)
  - Attack & Defense
  - Special Attack & Special Defense (Sp. for short)
  - Speed
- Types (water, fire, etc.)

Next component preview

Evolutions! Some pokemon can evolve.
You will add a new component for that.

```
PokemonApp.PokemonEvolutions = function (id, info) {
  this.id = id;
  this.info = info;
};

PokemonApp.PokemonEvolutions.prototype.render = function () {
  console.log("Rendering evolutions for: #" + this.id);

  // You will need some AJAX calls!
};
```

Conclusion

There are many different approaches out there to organize browser JavaScript.

That's why there are so many different frameworks.

Every framework has pros and cons.

Conclusion

It’s not important which approach you choose to organize your code.

What’s important is that you spend some time thinking about organization.