JavaScript - Framework - Stimulus

Overview

A lightweight framework designed to augment current HTML and give it behavior. Works well with Rails Turbolinks and doesn't seek to take over the frontend design by generating HTML.

Install - Ruby on Rails

To add to a Ruby on Rails project, add the package using Yarn:

yarn add stimulus

Then import Stimulus into the application javascript file and integrate it into webpack:

// app/javascript/packs/application.js.erb

import { Application } from "stimulus"

import { definitionsFromContext } from "stimulus/webpack-helpers"

const application = Application.start()

const context = require.context("./controllers", true, /\.js(?:.erb)?$/)

application.load(definitionsFromContext(context))

This will load javascript controllers from app/javascript/packs/controllers into the application.

How it Works

Stimulus is designed with three main concepts:

* Controllers - data-controller (defines which JS controller actions on element)
* Actions - data-action (defines actions to trigger controller methods)
* Targets - data-target (finding elements within the controller scope)

The Stimulus application works by monitoring the DOM, waiting for the data-controller attribute to appear in an element. The data-controller attribute connects a element to JavaScript controller, allowing it to gain the desired behavior from the JavaScript.

State is stored in the HTML meaning controllers can be discarded between page changes, but reinitialize as they were when cachced HTML appears again.

For example, a typical HTML element which can be set up with stimulus would be:

<div>

<input type="text">

<button>Greet</button>

</div>

Controllers

Controllers are javascript class extends used to give HTML behavior. Controllers are named <identifier>\_controller.js, where the identifier corresponds to the data-controller='<identifier>' attribute in a HTML element.

Names with multiple words are seperated with underscores in filenames, which are then converted to dashes in controller identifiers. Controllers which are nested in folders are have their path converted to double dashes in the identifier. For example:

**File name Identifier**

clipboard\_controller.js clipboard

date\_picker\_controller.js date-picker

users/list\_item\_controller.js users--list-item

local-time-controller.js local-time

sideMenu\_controller.js sideMenu

To create a controller create the javascript file in the controllers directory:

// app/javascript/packs/controllers/hello\_controller.js

import { Controller } from "stimulus"

export default class extends Controller {

}

Then add the controller identifier attribute to the target HTML element, for example:

<div data-controller='hello'>

<input type="text">

<button>Greet</button>

</div>

Actions

The methods defined inside Stimulus controllers are called actions, each of which can be linked to elements inside the identified HTML using the data-action tag with the following action descriptor syntax:

data-action='<event>-><controller-name>#<action-name>'

For example the following could be added to the sample HTML:

<div data-controller="hello">

<input type="text">

<button data-action="click->hello#greet">Greet</button>

</div>

Multiple actions can be added by chaining them in the data-action attribute, such as:

data-action='touchstart->tile#start touchend->tile#applyAction'

Targets

Targets map elements to controller properties, allowing them to be used dynamically in the controller. A data-target tags descriptor has the following syntax:

data-target='<controller-name>.<target-name>'

Stimulus will then automatically map the target element to the this.<target-name> property in the controller. For example, the data-target attribute could be added to the text field in the sample HTML:

<div data-controller="hello">

<input data-target="hello.name" type="text">

<button data-action="click->hello#greet">Greet</button>

</div>

To access target element in the controller the static targets array including all targets required in a controller is added. Stimulus then uses this find target attributes in the HTML. For each target name in the targets array, three properties are added to the controller:

* this.<target-name>Target - the first element with the data-target attribute
* this.<target-name>Targets - array of all elements with data-target attribute
* this.has<Target-name>Target - boolean true is target is present in HTML

The target property can then be used in the controller to console.log whatever has been typed into the input field when the greet button is clicked:

export default class extends Controller {

static targets = [ "name" ]

greet() {

const element = this.nameTarget

const name = element.value

console.log(`Hello, ${name}!`)

}

}

State

Stimulus keeps application state as attributes in the DOM, making controllers mainly stateless and idempotent.

There are three Stimulus lifecycle callback methods:

* initialize() - once controller is first instantised
* connect() - Anytime controller is connected to DOM
* disconnect() - Anytime controller is disconnected from DOM

State can then be added to various different areas of the DOM. A common place to store state is on the controller element under the attribute:

data-<controller-name>-<data-name>

This can then be accessed in the initialize through traditional getAttribute or using the Stimulus data API:

* this.data.has('<data-name>') - boolean on if data is present
* this.data.get('<data-name>') - Retrive value of data
* this.data.set('<data-name>') - Set value of data

For example, using this.data.get('<data-name>'):

initialize() {

const index = parseInt(this.data.get("index"))

this.showSlide(index)

}

To persist state, custom getters and setters can be used in the controller to modify the controller data attributes upon the trigger of actions. For example, setting the index state of a slideshow:

get index() {

return parseInt(this.data.get("index"))

}

set index(value) {

this.data.set("index", value)

this.showCurrentSlide()

}

Async Requests

Using the connect() method, it is possible to async request html data and insert it to elements:

<div data-controller="content-loader" data-content-loader-url="/messages.html"></div>

export default class extends Controller {

connect() {

this.load()

}

}

Async Refresh

Refresh certain elements of a webpage with a refresh controller which reads the data attribute for the refresh interval, then uses the Async requests load action to fetch new html data. Upon controller disconnect the interval can then be stopped:

<div data-controller="content-loader"

data-content-loader-url="/messages.html"

data-content-loader-refresh-interval="5000"></div>

import { Controller } from "stimulus"

export default class extends Controller {

connect() {

this.load()

if (this.data.has("refreshInterval")) {

this.startRefreshing()

}

}

disconnect() {

this.stopRefreshing()

}

load() {

fetch(this.data.get("url"))

.then(response => response.text())

.then(html => {

this.element.innerHTML = html

})

}

startRefreshing() {

this.refreshTimer = setInterval(() => {

this.load()

}, this.data.get("refreshInterval"))

}

stopRefreshing() {

if (this.refreshTimer) {

clearInterval(this.refreshTimer)

}

}

}

Testing Controllers

* Import stimulus and controller for testing
* beforeAll - add controller to application
* beforeEach - add html to document, set variables such as targets and controllers
* Test as normal with jest
* Ensure html loading is done in beforeEach else tests might be out of sync
* unloading controllers on afterAll to stop errors with disappearing document