



---

# Editoria11y v2: Building a Drupal-integrated Accessibility Checker

John Jameson • DrupalCamp NJ 2023

---

---

– *a True Story* –

Senior dev: "So...if you are going to build this, you are going to need to call *this* class, and *this* service, and..."

Me: "...how did you *learn* that? The *first* time? Like *how* could I have figured that out myself?"

"Uhh...I...saw it in a module somewhere..."



---

# Agenda

- What I had to learn to build this, so that you too can say that you have "seen it in a module somewhere"
  - What was I *thinking??*
  - JavaScript building blocks
  - Drupal building blocks



John Jameson • @itmaybejj  
Digital Accessibility Developer  
Web Development Services, Princeton University



In the  
beginning...



Ramses II obj E635,  
~1200 BC





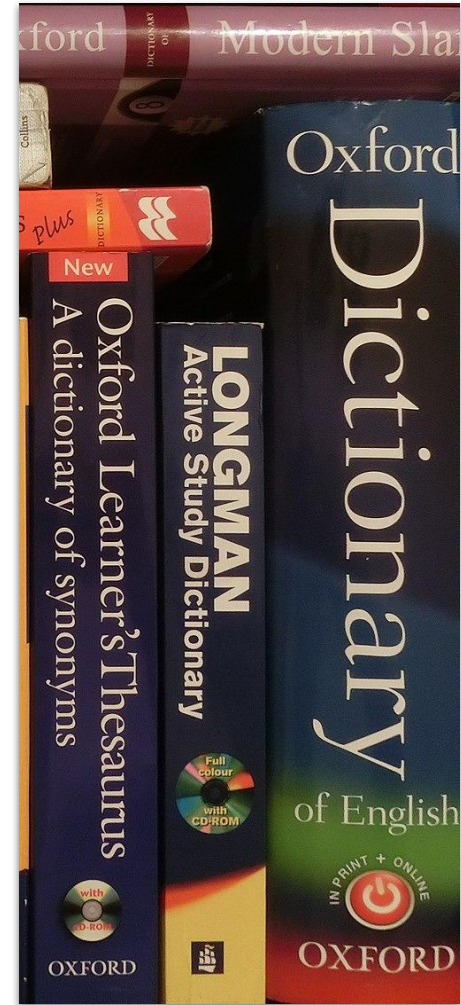
Birds had accessibility features – their beaks indicate the direction to read the line





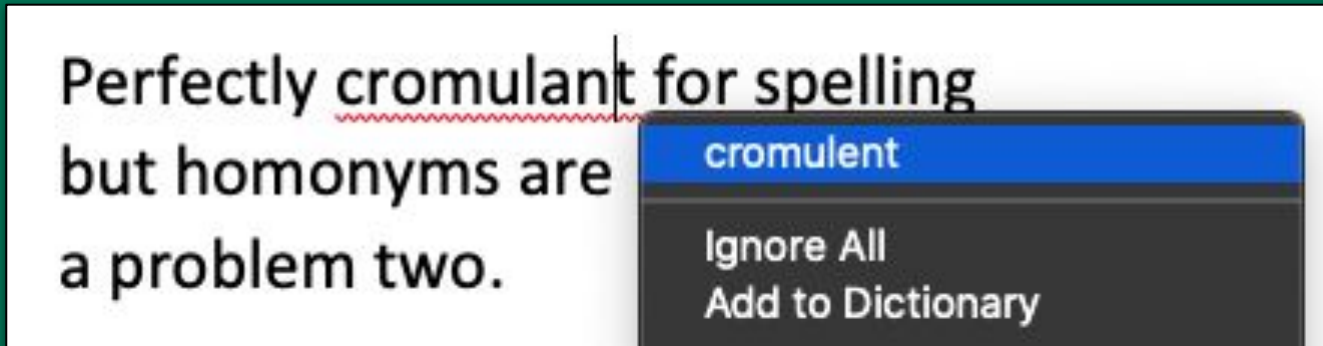
Proofreading  
used to be  
painstaking &  
error prone

Editors were  
expected to master  
a body of knowledge  
from arcane texts



---

# Spellcheck took proofreading from **arcane** to **automatic**



But **accessibility** checkers rely on people having mastery of arcane knowledge

We have 3,000+ authors.  
We cannot train them all.

The screenshot shows the WAVE web accessibility evaluation tool interface. The top left corner displays the WAVE logo and the text "powered by WebAIM" and "web accessibility evaluation tool". The address bar shows "Address: https://www.ryerson.ca/news-events/news/". Below the address bar, there is a "Styles" section with a toggle switch set to "ON". The "Details" section is active, showing a "Summary" tab and a "Details" tab. The "Details" tab lists the following issues:

- 2 Errors
  - 2 X Empty link
- 5 Alerts
  - 1 X Long alternative text
  - 1 X Redundant link
  - 1 X Noscript element

The right side of the screenshot shows a preview of the webpage with various accessibility annotations. A search bar is highlighted with the attribute "aria-label='Search'", and a button is highlighted with "aria='role='presentation'". Other annotations include "Ryerson University" and "aria-label='global'".

The screenshot shows a web browser displaying a video block titled "Video block - embed from mediacentral:". The video player shows a thumbnail of a building with a red play button and the text "273rd Princeton University Virtual Commencement Ceremony". To the right of the video player, the DevTools accessibility panel is open, showing the following information:

- Test URL: https://demo.localhost.princeton.edu/4443/
- TOTAL ISSUES: 5
- AUTOMATIC ISSUES: 5
- Best Practice: 3
- Critical: 1 Serious, 1
- Moderate: 3 Minor, 0
- Best Practices: ON, WCAG 2.1 AA
- Issues: Total Issues: 5
- Issues list:
  - Certain ARIA roles must contain particular children: 1
  - Frames must have an accessible name: 1
  - Ensures landmarks are unique: 1
  - All page content should be contained by landmarks: 2





# I looked for something intuitive and automatic, and of 30+ tools only Sa11y came close

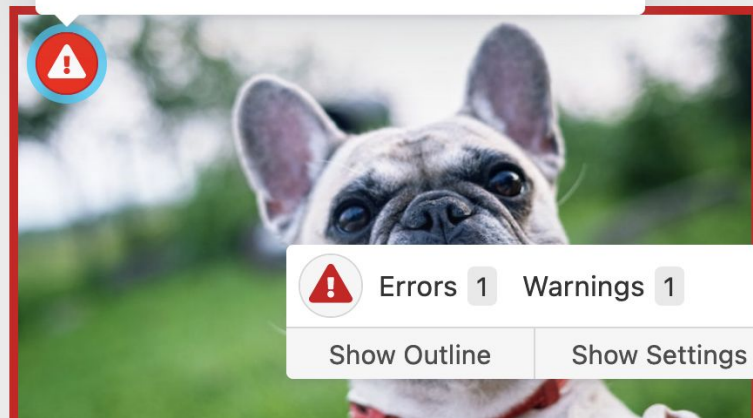
- Simple tips without WCAG AA 4.1.2 jargon
- Installed on *site*, not browser, so it appears automatically for everyone

attention. Sa11y does not only flag images missing alt text, but you can easily  
ance and quality. Sa11y has **Error** how best

then review the alt text

File extension within the alt text found. If the image conveys a story, mood, or important information - be sure to describe the image. Remove: **.jpg**.

Alt text: doggy.jpg



---

"I...think I can make this fit our needs?"

My manager gave me the time and team support to fork it:

- I thought it was too easily ignored when minimized
- I wanted to remove several features to simplify
- I wanted to write much longer tips
- I wanted to rewrite the JS to improve performance



---

## Three years later...

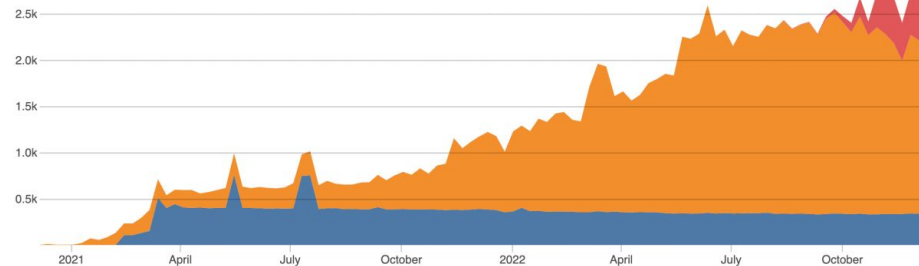
- v1 worked great!

I went from *sending*

repetitive emails to countless site owners after finding the same issues on site after site, to *receiving* emails asking for tips on writing more accessible content.

- v2 adds (synced) dismissals, site reports, theme variations, shadow DOM testing and rewritten tips.

Weekly project usage





---

## Quick interface tour

A little toggle button with an issue count appears on the bottom-right of a *rendered* page...

(live demo at [editoria11y.princeton.edu](http://editoria11y.princeton.edu) if that helps)



---

Automatically opens if there are **new** issues

## Meaningful Links

Links with no text at all



Links titled with a URL

 [s://editorial1.princeton.edu](https://editorial1.princeton.edu)

Issues	Outline	Alt Text	?	-	x
19 issues detected.					First »



# "Manual checks" have a one-click dismissal

## Manual check: was a heading level skipped?



Headings and subheadings are the page's table of contents. The *numbers* indicate indents, in a nesting relationship:

- Heading level 1
  - Heading level 2: a topic
    - Heading level 3: a subtopic
  - Heading level 2: a new topic

This heading skipped from level 3 to level 6. From a screen reader, this sounds like content is missing.

To fix: adjust levels to form an accurate outline, without gaps.

Hide alert for me

Mark OK for all users



**T ? H6 SHOULD HAVE BEEN AN H3**

Issues

Outline

Alt Text

?

-

x

19 issues detected.

« Previous

Next »





# Tests focus on content *alone*

- Links only titled with generic text: “**click here**,” “learn more,” etc.
  - Links with **no text**
  - Links that spell out a **URL**
  - Skipped **heading levels**
  - **Empty headings**
  - **Very long headings**
  - **All-bold paragraphs** with no punctuation that may actually be headings
  - **LARGE QUANTITIES OF CAPS LOCK**
  - Images with **no alt text**
  - Images with a **filename** as alt text
  - Images with **very long alt text**
  - Alt text that contains **redundant text** like “image of” or “photo of”
  - Images in links with alt text that appears to be **describing the image** instead of the link destination
- ... and a dozen or so others



# Less and more...

## AXE browser plugin:

1. Links must have discernible text
2. Heading levels should only increase by one
3. Elements must have sufficient color contrast
4. Id attribute value must be unique
5. Ensures landmarks are unique
6. All page content should only be contained by landmarks


## Editoria11y:

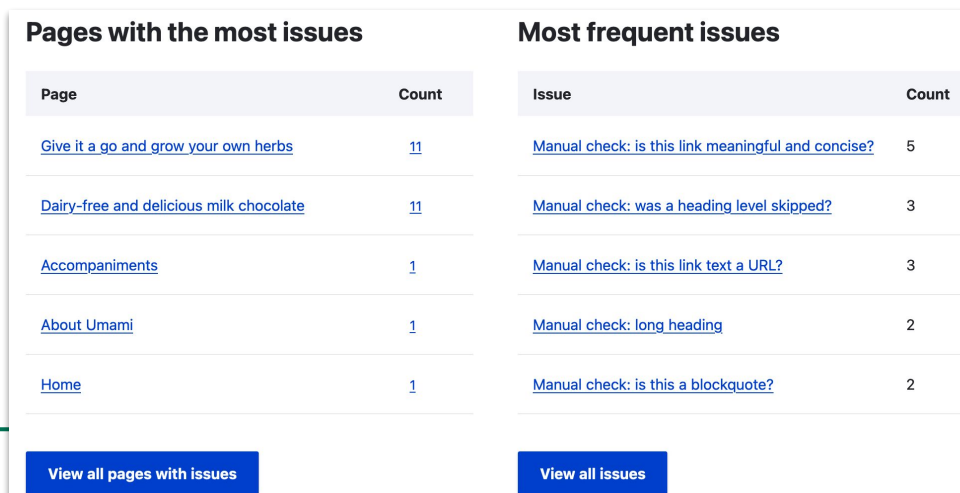
1. Heading tag without any text
2. Table has no header cells
3. Empty table header cell
4. Image's text alternative is unpronounceable
5. Link with no accessible text
6. Manual check: was a heading level skipped?
7. Manual check: image has no alt text
8. Manual check: is this uppercase text needed?
9. Manual check: very long alternative text
10. Manual check: is this a blockquote?
11. Manual check: is this link text a URL?
12. Manual check: is opening a new window expected?
13. Manual check: possibly redundant text in alt
14. Manual check: is this link meaningful and concise?
15. Manual check: is the linked document accessible?
16. Manual check: should this have list formatting?
17. Manual check: is this video accurately captioned?
18. Manual check: long heading
19. Manual check: should this be a heading?



---

# And I wrote a turnkey Drupal integration

- **Defaults** for regions-to-check and elements-to-ignore that work well with common themes and modules
- An **admin GUI** to change defaults (e.g., ignored elements)
- Synched dismissals and site-wide **reports:** 



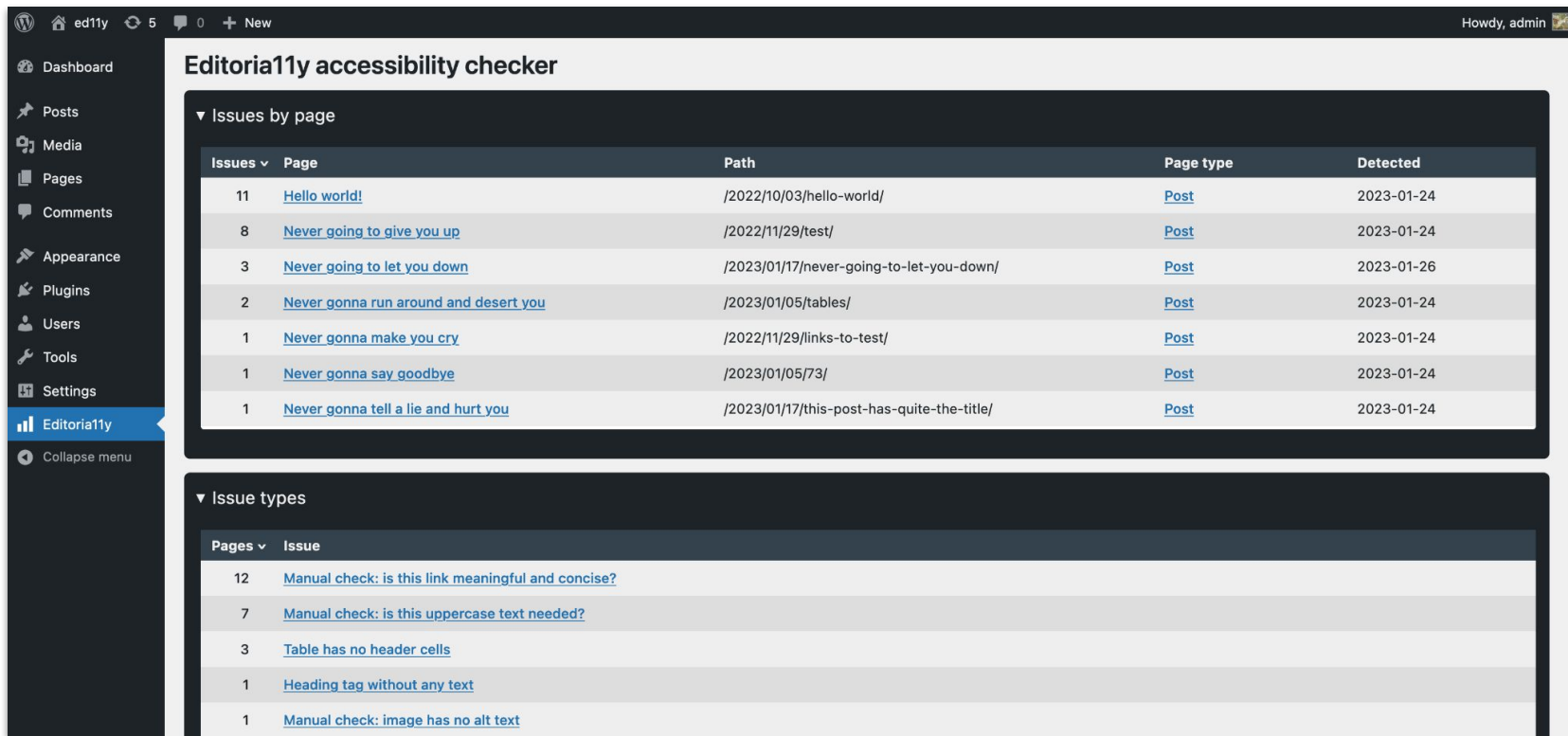
Pages with the most issues		Most frequent issues	
Page	Count	Issue	Count
<a href="#">Give it a go and grow your own herbs</a>	11	<a href="#">Manual check: is this link meaningful and concise?</a>	5
<a href="#">Dairy-free and delicious milk chocolate</a>	11	<a href="#">Manual check: was a heading level skipped?</a>	3
<a href="#">Accompaniments</a>	1	<a href="#">Manual check: is this link text a URL?</a>	3
<a href="#">About Umami</a>	1	<a href="#">Manual check: long heading</a>	2
<a href="#">Home</a>	1	<a href="#">Manual check: is this a blockquote?</a>	2

[View all pages with issues](#) [View all issues](#)





# (and a WordPress plugin)



The screenshot displays the WordPress dashboard with the Editoria11y accessibility checker plugin active. The left sidebar shows the navigation menu with 'Editoria11y' highlighted. The main content area is titled 'Editoria11y accessibility checker' and contains two sections: 'Issues by page' and 'Issue types'.

**Issues by page**

Issues	Page	Path	Page type	Detected
11	<a href="#">Hello world!</a>	/2022/10/03/hello-world/	<a href="#">Post</a>	2023-01-24
8	<a href="#">Never going to give you up</a>	/2022/11/29/test/	<a href="#">Post</a>	2023-01-24
3	<a href="#">Never going to let you down</a>	/2023/01/17/never-going-to-let-you-down/	<a href="#">Post</a>	2023-01-26
2	<a href="#">Never gonna run around and desert you</a>	/2023/01/05/tables/	<a href="#">Post</a>	2023-01-24
1	<a href="#">Never gonna make you cry</a>	/2022/11/29/links-to-test/	<a href="#">Post</a>	2023-01-24
1	<a href="#">Never gonna say goodbye</a>	/2023/01/05/73/	<a href="#">Post</a>	2023-01-24
1	<a href="#">Never gonna tell a lie and hurt you</a>	/2023/01/17/this-post-has-quite-the-title/	<a href="#">Post</a>	2023-01-24

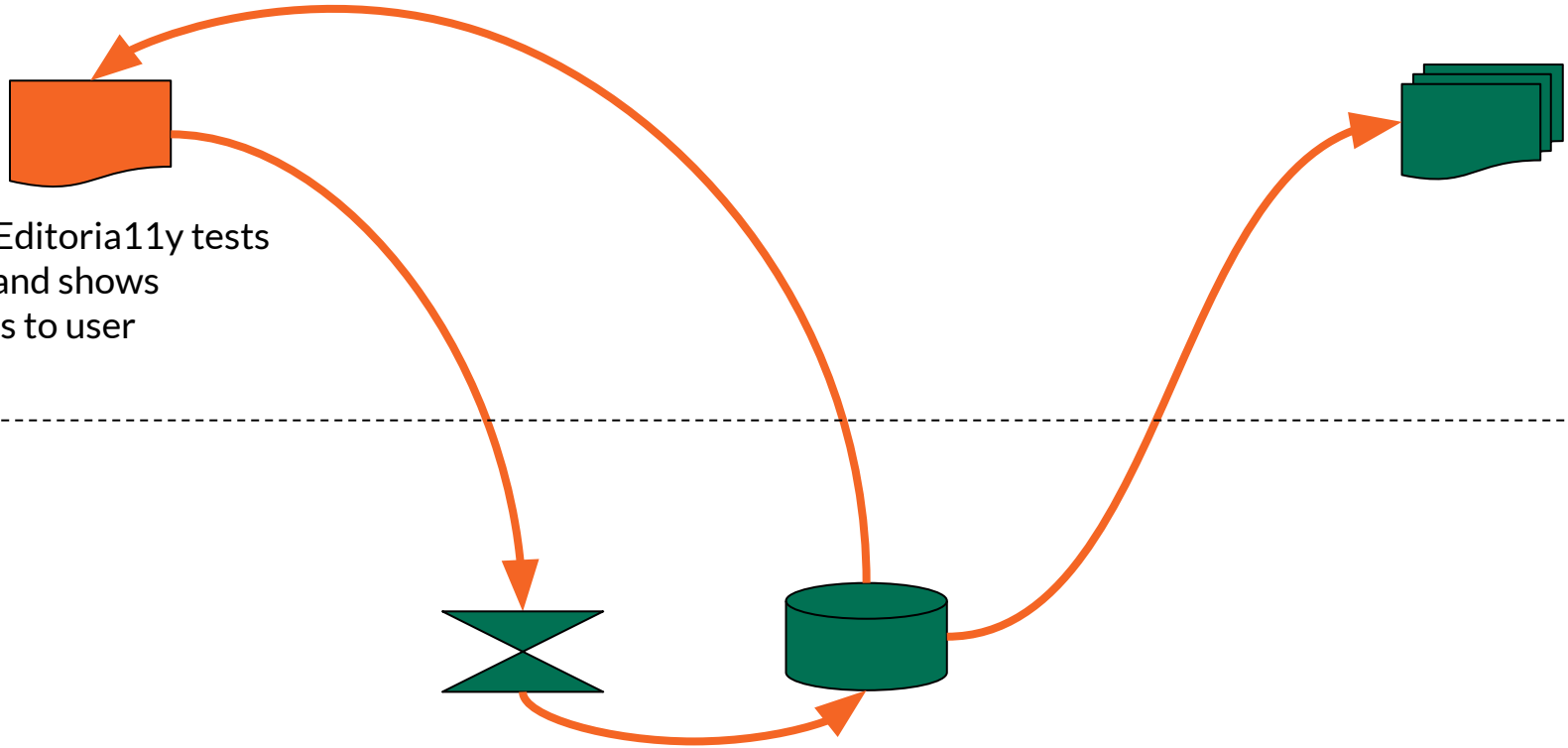
**Issue types**

Pages	Issue
12	<a href="#">Manual check: is this link meaningful and concise?</a>
7	<a href="#">Manual check: is this uppercase text needed?</a>
3	<a href="#">Table has no header cells</a>
1	<a href="#">Heading tag without any text</a>
1	<a href="#">Manual check: image has no alt text</a>



# Building block 1: writing JS tests

1: *JS*. Editoria11y tests page and shows results to user



Tests are just  
if() statements,  
not AI or magic

Conceptually...

1. Get all the `<img>` tags.
2. Check each for an `alt` attribute.
3. If there *isn't* one, display an **error**.
4. If there is one but it's *blank*, display a manual check.
5. If there is one that ends with ".jpg, .gif or .png," display a manual check.





---

## jQuery made DOM traversal easy in v1

```
<div id="foo">  
  <p class="bar">  
    <strong><span>Hi!!!</span></strong>
```

```
console.log($('#foo').find('.bar strong').not('.baz').text());
```

```
→ "Hi!!!"
```

---



---

## Converting to Vanilla JS was less...*forgiving*

```
<div id="foo">  
  <p class="bar">  
    <strong>Hi</strong>
```

```
document.getElementById('bar').textContent();
```

```
! ▼ Uncaught TypeError: document.getElementById(...) is null  
  <anonymous> debugger eval code:1  
  [Learn More]  
  <anonymous> debugger eval code:1
```



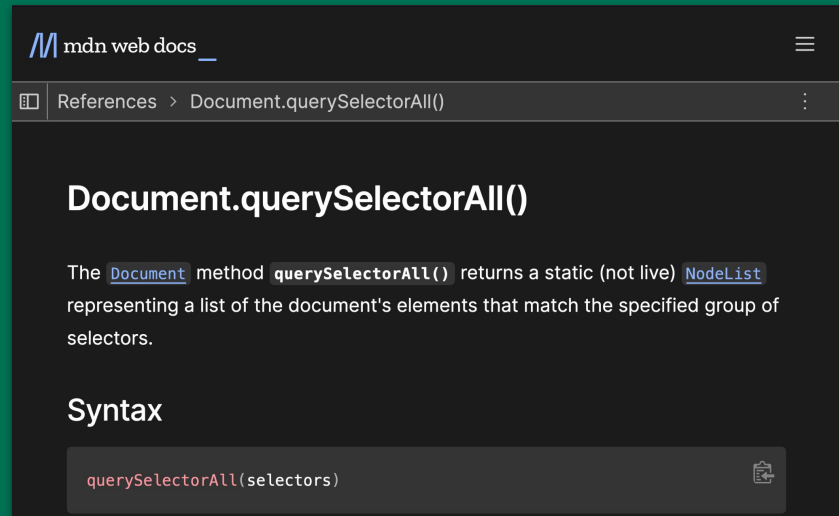
# I pinned these tabs for months.

## You might not need jQuery

jQuery and its cousins are great, and by all means use them if it makes it easier to develop your application.

If you're developing a library on the other hand, please take a moment to consider if you actually need jQuery as a dependency. Maybe you can include a few lines of utility code, and forgo the requirement. If you're only targeting more modern browsers, you might not need anything more than what the browser ships with.

At the very least, make sure you know what [jQuery is doing for you](#), and what it's not. Some developers believe that jQuery is protecting us from a great demon of browser incompatibility when, in truth, post-IE8, browsers are pretty easy to deal with on their own; and after the Internet Explorer era, the browsers do even more.



The screenshot shows the MDN Web Docs page for the `Document.querySelectorAll()` method. The page title is "Document.querySelectorAll()". The breadcrumb navigation shows "References > Document.querySelectorAll()". The main heading is "Document.querySelectorAll()". The text describes the method: "The `Document` method `querySelectorAll()` returns a static (not live) `NodeList` representing a list of the document's elements that match the specified group of selectors." The "Syntax" section shows the signature: `querySelectorAll(selectors)`.



---

## DOM traversal building blocks

1. New CSS makes `querySelectorAll()` easy and performant:

```
:is('h1, h2, h3, h4') :not('nav h2, footer h3')
```

2. Chainable traversals:

```
v.querySelector(), ^.closest(), >.nextElementSibling()
```

3. Optional-chaining returns "undefined" if the element doesn't exist, rather than `Uncaught TypeError`:

```
myElement?.querySelector('.inner-span')
```

---





---

## Here's how I find elements to test

I get all the editable parts of the page, based on module settings:

```
document.querySelectorAll(`:is(${checkRoots})`);  
  
// e.g. `:is("main, .footer-content")`
```

Then I traverse *into* each, respecting module "ignore" settings:

```
Edlly.roots.forEach(root => {  
  // (skipping the business logic to concatenate these results to an array)  
  root.querySelectorAll(`:is(${selectorList})${ignoreList}`);  
  
  // e.g. `:is("p"):not(".ignored p")`
```



## Now let's write a test looking for fake headings

```
// Now check for possible heading.
```

```
let possibleHeading = p.querySelector('strong:not(table strong), b:not(table b)');
```

```
// Exclude paragraphs with links, then check if strong length equals p length.
```

```
if (possibleHeading && !p.querySelector('a')) {
```

```
    possibleHeading = Ed11y.getText(possibleHeading);
```

```
    let length = possibleHeading.length;
```

```
    let maybeSentence = possibleHeading.match(/[\.:;?!"']/) !== null;
```

```
    if (121 > length && length > 5 && length === firstText.length && maybeSentence === false) {
```

```
        let dismissKey = Ed11y.dismissalKey(possibleHeading);
```

```
        Ed11y.results.push([p, 'textPossibleHeading', Ed11y.M.textPossibleHeading.tip(), 'afterbegin',
```

```
    ]
```

```
}
```



Check for <strong>, unless we're in a table...

---

## We need to get the **text** for this test. How?

- **.html / .innerHTML**  
...gets you in OWASP trouble. *Avoid...*
- **.innerText()**  
...gets *rendered* text, including inner nodes and CSS.
- **.textContent()**  
...gets the text node itself.



# Check textContent for punctuation and length

```
// Now check for possible heading.
let possibleHeading = p.querySelector('strong:not(table strong), b:not(table b)');
// Exclude paragraphs with links, then check if strong length equals p length.
if (possibleHeading && !p.querySelector('a')) {
  possibleHeading = Ed11y.getText(possibleHeading);
  let length = possibleHeading.length;
  let maybeSentence = possibleHeading.match(/[:;?!"']/) !== null;
  if (121 > length && length > 5 && length === firstText.length && maybeSentence === false) {
    let dismissKey = Ed11y.dismissalKey(possibleHeading);
    Ed11y.results.push([p, 'textPossibleHeading', Ed11y.M.textPossibleHeading.tip(), 'afterbegin',
  ]
}
```

If there is no link and no punctuation, compare the *length* of the **strong** and the **p**



## Build a "dismissalKey" and add to the results array

```
// Now check for possible heading.
let possibleHeading = p.querySelector('strong:not(table strong), b:not(table b)');
// Exclude paragraphs with links, then check if strong length equals p length.
if (possibleHeading && !p.querySelector('a')) {
  possibleHeading = Ed11y.getText(possibleHeading);
  let length = possibleHeading.length;
  let maybeSentence = possibleHeading.match(/[:;?!"']/) !== null;
  if (121 > length && length > 5 && length === firstText.length && maybeSentence === false) {
    let dismissalKey = Ed11y.dismissalKey(possibleHeading);
    Ed11y.results.push([p, 'textPossibleHeading', Ed11y.M.textPossibleHeading.tip(), 'afterbegin',
  }
}
```

Stores a reference to the node, a triggered test ID and various bits of identifying information



---

# Why am I building an array\* of results?

## 1. For *performance*

- a. In order to **read** some properties, the element must be **Painted**.
- b. When we **write**, the element is queued for re-painting.
- c. In order to **read** the next element, the page must be **repainted**.
- d. Alternating is a **huge** slowdown. Read, then write.

## 2. For *magic*

- a. Syncing copies of the array to servers
- b. WordPress plugin uses headless runs to build in-editor alerts

---

\* Actually Adam Chaboryk realized this is easier to code as an **object** when backporting it to Sa11y. Live and learn.





# Users tolerate slowness on click, but not on load.

## AXE plugin:

1. Links must be...
2. Heading levels must be by one...
3. Elements must have color contrast
4. Id attribute value must be unique
5. Ensures landmarks are unique
6. All page content should only be contained by landmarks

~1s

## Editoria11y:

1. Heading...
2. Table has...
3. Empty t...
4. Image's alt text is pronounceable
5. Link with no access...
6. Manual check: was a heading level skipped?
7. Manual check: image has no alt text
8. Manual check: is this uppercase text needed?
9. Manual check: very long alternative text
10. Manual check: is this a blockquote?
11. Manual check: is this link text a URL?
12. Manual check: is opening a new window expected?
13. Manual check: possibly redundant text in alt
14. Manual check: is this link meaningful and concise?
15. Manual check: is the linked document accessible?
16. Manual check: should this have list formatting?
17. Manual check: is this video accurately captioned?
18. Manual check: long heading
19. Manual check: should this be a heading?

<0.1s



---

## Performance matters because async is a lie

- JS is **single threaded:** "jank" is clicks waiting in line
- "Async" functions wait for their callback to *get in line*; they don't *run* asynchronously once they are in line
- I took to breaking up slow functions with a timeout(). Timeouts don't call back until the line has cleared:



---

## Now...how to draw elements without `.HTML()`?

```
let myP = document.createElement('p');
```

```
myP.classList.add('stylish');
```

```
myP.textContent = "No Bobby Tables here";
```

```
myDivWrapper.append(myP);
```

...

```
<div><p class="stylish">No Bobby Tables here</p></div>
```

---



# I explored **Web Components** to reduce CSS conflicts

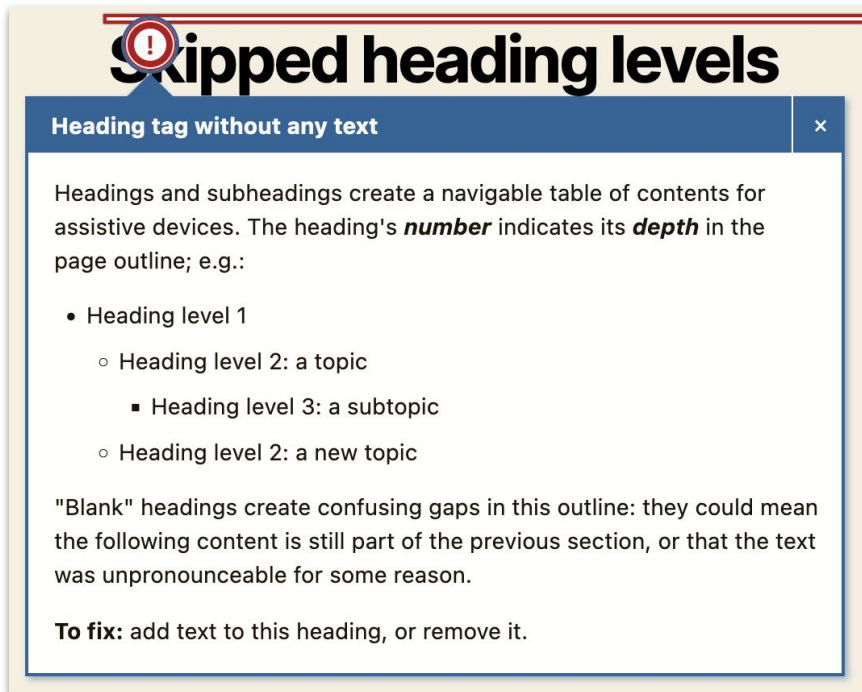
```
<ed11y-element-tip>
```

```
  #shadow-root
```

```
    <div role="dialog"> ...stuff... </div>
```

```
</ed11y-element-tip>
```

The *#shadow-root* is like a porous iframe – it **blocks CSS inheritance** from themes.



## Skipped heading levels

Heading tag without any text

Headings and subheadings create a navigable table of contents for assistive devices. The heading's *number* indicates its *depth* in the page outline; e.g.:

- Heading level 1
  - Heading level 2: a topic
    - Heading level 3: a subtopic
  - Heading level 2: a new topic

"Blank" headings create confusing gaps in this outline: they could mean the following content is still part of the previous section, or that the text was unpronounceable for some reason.

**To fix:** add text to this heading, or remove it.



# There's a boilerplate they all start with...

[developer.mozilla.org/en-US/docs/Web/Web\\_Components](https://developer.mozilla.org/en-US/docs/Web/Web_Components)

```
class Ed11yElementTip extends HTMLElement {  
  constructor() {  
    super();  
  }  
  //==== your JS ====//  
}  
  
customElements.define('ed11y-element-tip',  
  Ed11yElementTip);
```



---

## They include a `connectedCallback()`

```
connectedCallback() {  
  if (!this.initialized) {  
    // Get info from attributes on the tag  
    this.resultID = this.dataset.ed11yResult;  
    this.result = Ed11y.results[this.resultID];  
  
    // Build DOM and attach as "shadow root"  
    const shadow = this.attachShadow({mode: 'open'});  
    this.wrapper = document.createElement('div');  
    shadow.appendChild(this.wrapper);  
  }  
}
```





---

## Attach CSS for content *and* the :host

```
const style = document.createElement('style');  
style.textContent = Ed11y.baseCSS + `  
  :host {  
    position: absolute;  
  }  
  .wrapper {  
    color: ${Ed11y.theme.text};  
  }  
shadow.appendChild(style);
```



---

## Automatic **observedAttributes** callbacks

<ed11y-element-tip data-ed11y-action="observed">

```
static get observedAttributes() {return ['data-ed11y-action'];}
```

```
attributeChangedCallback(attr, oldValue, newValue) {
```

```
  switch (attr) {
```

```
    case 'data-ed11y-action':
```

```
      let changeTo = newValue === 'open' ? true : false;
```

```
      this.toggleTip(changeTo);
```



---

# So browsers build *and* communicate for you



The image shows two browser developer tool panels. On the left is the 'Issues' panel, which has tabs for 'Issues', 'Outline', 'Alt Text', '?', '-', and 'x'. It displays '19 issues detected.' and navigation buttons '« Previous' and 'Next »'. On the right is a 'Skipped heading levels' tip, which has a blue header with a red exclamation mark icon and a close button 'x'. The tip's text reads: 'Heading tag without any text' and 'Headings and subheadings create a navigable table of contents for assistive devices. The heading's *number* indicates its *depth* in the page outline; e.g.:'. A green arrow points from the tip to the 'Issues' panel.

When you click "Next »":

1. My **panel component** *calls* a function from the **main JS**
  2. The **main JS** *changes* the next tip's data attribute to "open"
  3. The **tip component** *observes* the change, triggering its function to *pull* content from the results array and open
- 



# The rest is just HTML:

## Manual check: was a heading level skipped?

Headings and subheadings are the page's table of contents. The *numbers* indicate indents, in a nesting relationship:

- Heading level 1
  - Heading level 2: a topic
    - Heading level 3: a subtopic
  - Heading level 2: a new topic

This heading skipped from level 3 to level 6. From a screen reader, this sounds like content is missing.

To fix: adjust levels to form an accurate outline, without gaps.

Hide alert for me

Mark OK for all users

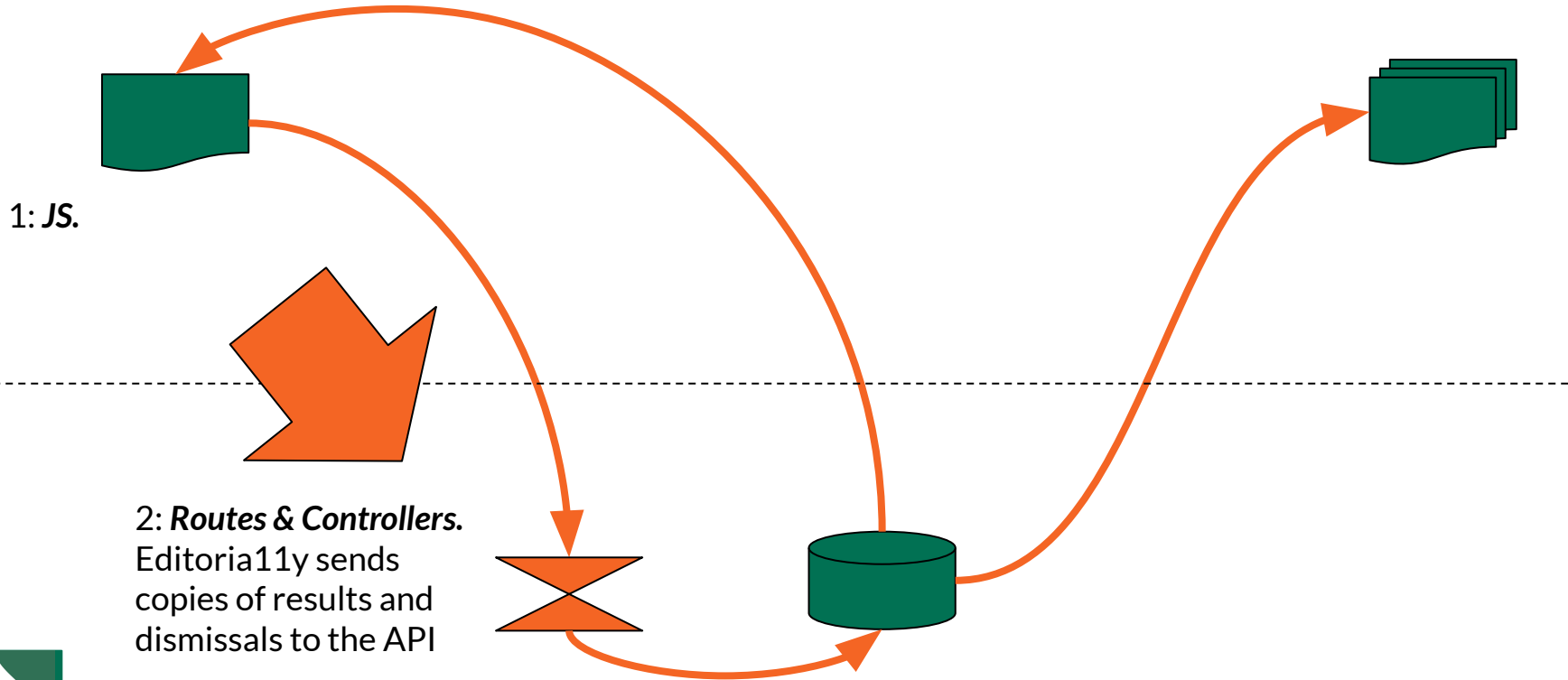


**T ? H6 SHOULD HAVE BEEN AN H3**

```
▼ <ed11y-element-tip data-ed11y-result="10" style="outline: transparent solid 0px; top: 1792.58px; left: 164.5px; transform: translate(-61.6px, 50px);" data-ed11y-action="false" data-ed11y-open="true"> overflow custom...
▼ #shadow-root (open)
  ▶ <style>...</style>
  ▼ <div class="wrapper ed11y-result open" role="dialog">
    <div tabindex="0"></div> event
    <div class="arrow" style="left: 67.6px;" data-direction="under"></div>
  ▼ <div class="tip" aria-labelledby="tip-title-10" style="">
    <div id="tip-10" class="title" tabindex="-1">
      Manual check: was a heading level skipped?
    </div> grid
    <button class="close" aria-label="close">x
    </button> event grid
    ▶ <div class="content">...</div>
  </div>
  <div tabindex="0"></div> event
</div>
</ed11y-element-tip>
```



# Building block 2: Sending data to a Drupal Module



# Refresher

– or –

## Introduction

*Parts of a Drupal module*

### *.yml files*

- **Human-readable files** with config for Drupal: permissions, defaults...

### *.module, .install files*

- PHP files with specifically named functions Drupal ***calls automatically***

### *.php files*

- Custom classes or implementations of Drupal classes, called ***manually***





If these are new  
concepts for you  
(they were for me)

If you don't get lucky and find boilerplate at [drupal.org/docs/drupal-apis](https://drupal.org/docs/drupal-apis) or [drupal.org/docs/develop](https://drupal.org/docs/develop):

1. Ask coworkers/Slack for examples of modules that do something similar
2. Explore Core looking for functions that do something similar

...Once you *know* what questions to ask, there *is* decent help and boilerplate examples out there, but I sure didn't know what to ask at first.



---

# **.permissions** are straightforward

`permissions.yml` lists *names* of permissions you want Drupal to create for you

```
'view editorially checker':
```

```
  title:      'View Editorially checker'
```

```
  description: 'Assign to all content editors.'
```

```
'mark as ok in editorially':
```

```
  title:      'Mark as OK in Editorially'
```

```
  description: 'Adds button to suppress a manual check for  
all users, if sync is enabled.'
```

! editoria11y.info.yml

! editoria11y.install

! editoria11y.libraries.dev.yml

! editoria11y.libraries.yml

! editoria11y.links.menu.yml

! editoria11y.module

! editoria11y.permissions.yml



! editoria11y.routing.yml

! editoria11y.services.yml

---

## `.routing` sets up my API reporting URL

```
editorially.api_report:  
  path: '/editorially/api/results/report'  
  defaults:  
    _controller:  
      '\Drupal\editorially\Controller\ApiController::report'  
  methods: [POST]  
  requirements:  
    _permission: 'view editorially checker'  
  (etc)
```



# JS boilerplate sends my data to my route

```
let postData = async function (action, data) {  
  if (!csrfToken) {  
    getCsrftoken (action, data);  
  } else {  
    let url = `${apiRoot}${action}`;  
    fetch (url, {  
      method: 'POST',  
      headers: {  
        'Content-Type': 'application/json',  
        'X-CSRF-Token': csrfToken,  
      },  
      body: JSON.stringify (data),  
    })
```



# The route forwards to a **controller**:

```
public function report(Request $request) {  
    try {  
        $results = Json::decode($request->getContent());  
        $this->api->testResults($results);  
        return new JsonResponse("ok");  
    }  
    catch (\Exception $e) {  
        return $this->sendErrorResponse($e);  
    }  
}
```



- src
  - Controller
    - ApiController.php**
    - DashboardController.php
    - ExportCSVController.php
  - Exception
  - Form
  - Api.php
  - Dashboard.php
  - DismissalsOnPage.php
- tests

# The controller sends \$results to API.php

```
public function testResults($results) {  
    $this->validateNumber($results["page_count"]);  
    $this->validatePath($results["page_path"]);  
    foreach ($results["results"] as $key => $value) {  
        $this->validateNumber($value);  
        if ($results["page_count"] > 0) {  
            $this->connection->merge("editorially_results")  
                ->insertFields([  
                'page_title' => $results["page_title"],  
                'page_path' => $results["page_path"],  
                (etc)
```

- src
  - Controller
    - ApiController.php
    - DashboardController.php
    - ExportCSVController.php
    - Exception
    - Form
    - Api.php**
    - Dashboard.php
    - DismissalsOnPage.php

The database  
takes care of the  
rest based on a  
"schema"

- ! editoria11y.info.yml
- ! editoria11y.install
- ! editoria11y.libraries.dev.yml
- ! editoria11y.libraries.yml
- ! editoria11y.links.menu.yml
- ! editoria11y.module
- ! editoria11y.permissions.yml
- ! editoria11y.routing.yml
- ! editoria11y.services.yml

```
function editorially_results_table () {  
  $data = [  
    'fields' => [  
      'id' => [  
        'description' => 'Test result',  
        'type' => 'serial',  
        'size' => 'big',  
        'not null' => TRUE,  
      ],  
      (...etc...)  
    ],  
    return $data;  
  }  
}
```

```
function editorially_schema () {  
  $schema['editorially_results'] =  
    editorially_results_table ();  
  return $schema;  
}
```





# And then...it's alive!

## JSON

```
entity_type: "Basic page"
language: "en"
oks: {}
page_count: 22
page_path: "/editoria11y/demo"
page_title: "Page with many errors"
page_url: "/editoria11y/demo"
▼ results: {...}
  Content heading inside a table: 1
  Empty table header cell: 1
  Heading tag without any text: 1
  Image has no alternative text attribute: 1
```

## Headers

Filter Headers

POST

Scheme: https

Host: jameson.mycpanel.princeton.edu

Filename: /editoria11y/editoria11y/api/results/report

Address: 159.206.78.14:443

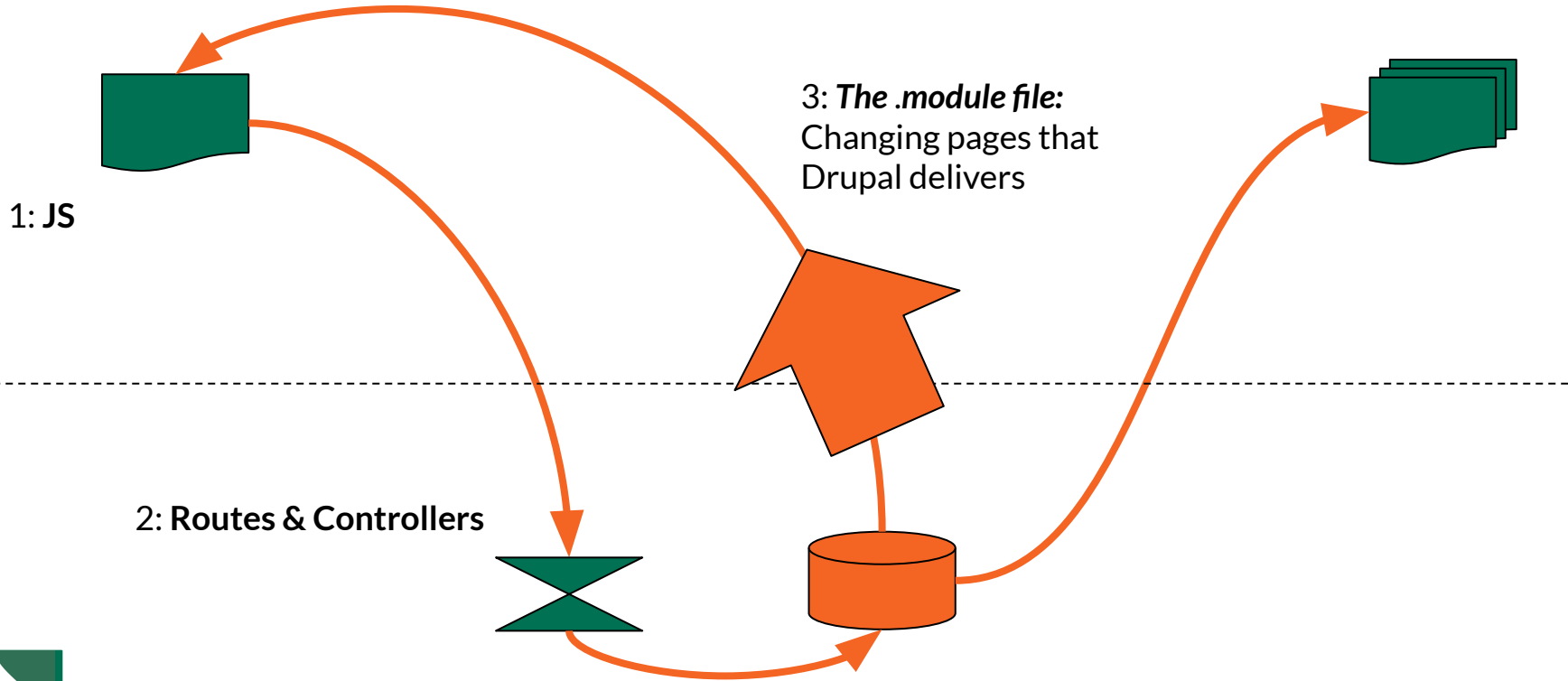
Status 200 OK ?

Version HTTP/2

Transferred 350 B (4 B size)



# Building block 3: getting data *back*



---

## `drupalSettings` is how I communicate

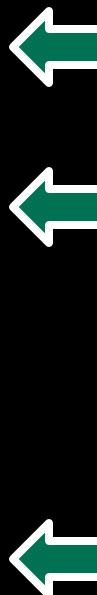
Modules can piggyback Drupal's global JSON object:

```
>> drupalSettings.editoria1ly
< ▼ Object { page_title: "Home", allow_hide: true, allow_ok: true,
  /api/results/report", session_url: "/session/token", lang: "en",
  allow_hide: true
  allow_ok: true
  api_url: "/editoria1ly/api/results/report"
  assertiveness: "smart"
  changed: "1677081157"
  content_root: ""
  dashboard_url: "/admin/reports/editoria1ly" }
```



## `.module` adds `page_attachments`:

```
function editorially_page_attachments(array &$attachments) {  
  
    $attachments['#attached']['library'][] = 'editorially/editorially';  
  
    $apiUrl = Url::fromRoute('editorially.api_report')->toString();  
    $attachments['#attached']['drupalSettings']['editorially']['api_url'] =  
        $apiUrl;  
  
    $attachments['#attached']['drupalSettings']['editorially']['root'] =  
        $config->get('content_root');
```



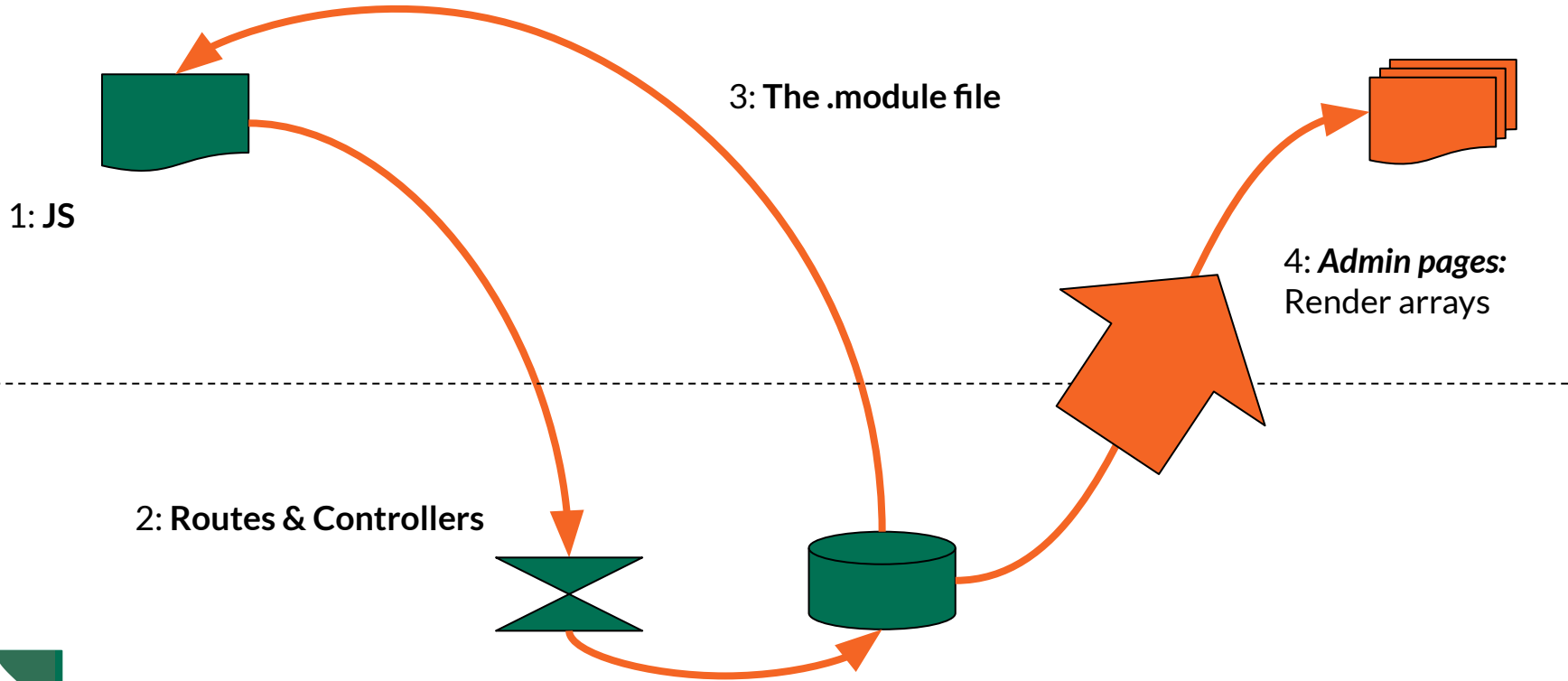
# Use a \$query (boilerplate...) to get our dismissals

```
public function getDismissals ($page_path) {  
    $query = $this->database  
        ->select('editorially_dismissals')  
        ->fields('editorially_dismissals',  
            [ 'uid',  
              'result_key',  
              'element_id',  
              'dismissal_status',  
              'page_path',  
            ] )  
        ->condition('page_path', $page_path);  
    return $query->execute();  
}
```



- src
  - Controller
    - ApiController.php
    - DashboardController.php
    - ExportCSVController.php
  - Exception
  - Form
  - Api.php
  - Dashboard.php
  - DismissalsOnPage.php

(time permitting) **Building block 4: making custom pages**



# Reports: more routes, more controllers

- src
  - Controller
    - ApiController.php
    - DashboardController.php
    - ExportCSVController.php
  - > Exception
  - > Form
  - Api.php
  - Dashboard.php
  - DismissalsOnPage.php

The screenshot shows the 'Content Accessibility Issues' report in a Drupal administration interface. The top navigation bar includes 'Back to site', 'Manage', 'jameson', 'Prod', 'Help', and 'Subscription active (expires 2023/9/29)'. The main content area is titled 'Content Accessibility Issues' and includes a breadcrumb 'Home » Administration » Reports'. It features two tables: 'Pages with the most issues' and 'Most frequent issues'. Below these are buttons for 'All pages with issues', 'Issues by type', and 'Issues by date'. At the bottom, there is a 'Recent dismissals' table.

PAGE	COUNT
Editorially	241
Resource Links	11
Publications: Citations List	4
Event Title 4	3
John Jacobs	3

ISSUE	COUNT
Manual check: is this link text a URL?	103
Manual check: should this have list formatting?	51
Manual check: is the linked document accessible?	28
Manual check: is this link meaningful and concise?	14
Manual check: possibly redundant text in alt	13

PAGE	ISSUE	MARKED AS	USER	DATE
Home	Manual check: is this video accurately captioned?	ok	jameson	01/19/2023 - 5:00 pm

```
public function results(): array {  
    return $this->getTestResults();  
}
```



---

## We process a \$query into rows in an array...

```
$results = $this->dashboard->getResults();  
foreach ($results["results"] as $record) {  
    $rows[] = [  
        $linkToPage,  
        $linkToIssuesByPage,  
        $record->entity_type,  
        $record->page_path,  
    ];  
}
```



- src
  - Controller
    - ApiController.php
    - DashboardController.php
    - ExportCSVController.php
  - Exception
  - Form
  - Api.php
  - Dashboard.php
  - DismissalsOnPage.php



---

Then Drupal **renders** the array as a <table>

```
$render[] = [  
  '#type' => 'table',  
  '#header' => $header,  
  '#rows' => $rows,  
  '#cache' => [  
    'contexts' => [ 'user', 'url', ],  
    'tags' => [ 'editorially.dashboard', ],  
  ],  
];
```



- src
  - Controller
    - ApiController.php
    - DashboardController.php
    - ExportCSVController.php
  - Exception
  - Form
  - Api.php
  - Dashboard.php
  - DismissalsOnPage.php

---

## And voila – a table!

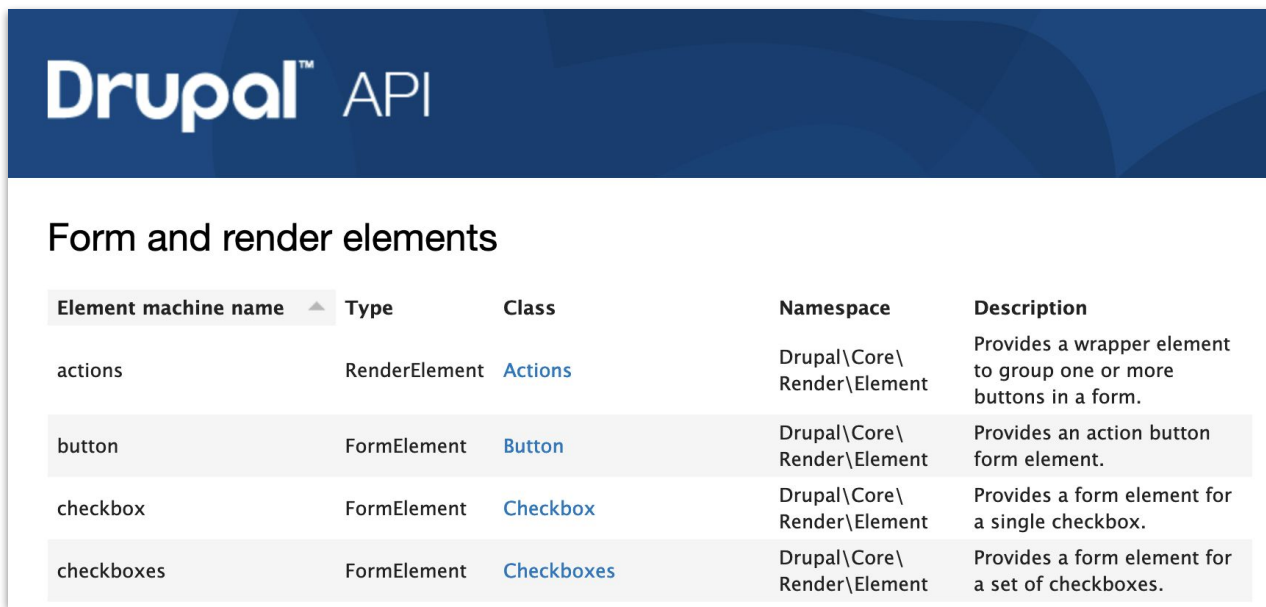
PAGE	ISSUES FOUND <span>▼</span>	TYPE	PATH
<a href="#">Editoria1ly</a>	241	Page	<a href="/editoria1ly">/editoria1ly</a>
<a href="#">Resource Links</a>	11	Page	<a href="/content-types/resource-links">/content-types/resource-links</a>
<a href="#">Publications: Citations List</a>	4	Page	<a href="/content-types/publications/citations-list">/content-types/publications/citations-list</a>
<a href="#">Resource Links Library with filters</a>	3	Page	<a href="/content-types/resource-links/resource-links-library">/content-types/resource-links/resource-links-library</a>



# Element rendering syntax is documented...

I had this open in a tab the whole time:

[api.drupal.org/api/drupal/elements/](https://api.drupal.org/api/drupal/elements/)

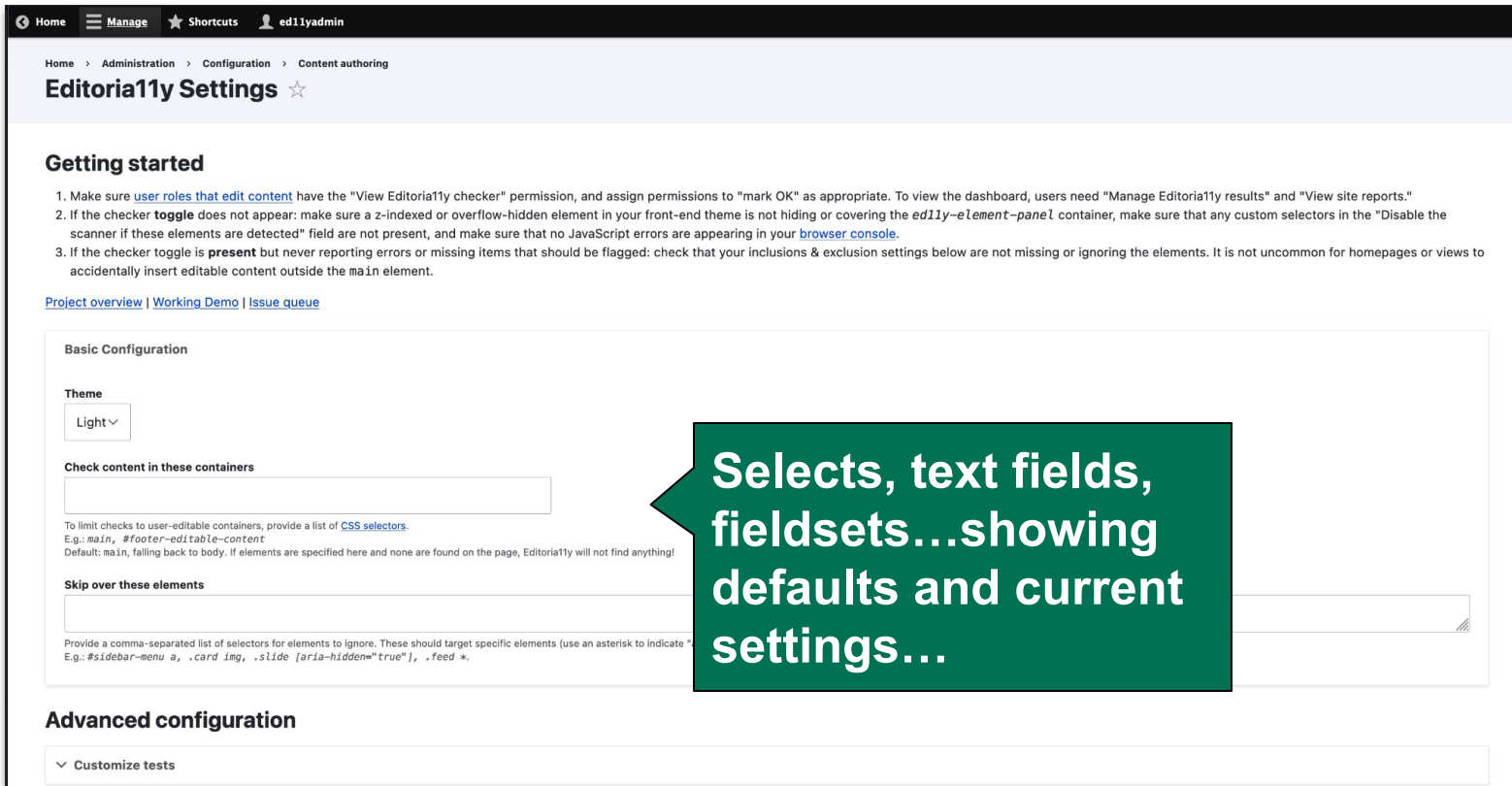


The screenshot shows the Drupal API documentation page for 'Form and render elements'. The page has a dark blue header with the 'Drupal API' logo. Below the header, the title 'Form and render elements' is displayed. A table lists various element types with columns for 'Element machine name', 'Type', 'Class', 'Namespace', and 'Description'. The table contains four rows of data.

Element machine name	Type	Class	Namespace	Description
actions	RenderElement	<a href="#">Actions</a>	Drupal\Core\Render\Element	Provides a wrapper element to group one or more buttons in a form.
button	FormElement	<a href="#">Button</a>	Drupal\Core\Render\Element	Provides an action button form element.
checkbox	FormElement	<a href="#">Checkbox</a>	Drupal\Core\Render\Element	Provides a form element for a single checkbox.
checkboxes	FormElement	<a href="#">Checkboxes</a>	Drupal\Core\Render\Element	Provides a form element for a set of checkboxes.



# What about settings pages?



The screenshot shows the Editoria11y Settings page. The breadcrumb trail is Home > Administration > Configuration > Content authoring. The page title is "Editoria11y Settings".

### Getting started

1. Make sure [user roles that edit content](#) have the "View Editoria11y checker" permission, and assign permissions to "mark OK" as appropriate. To view the dashboard, users need "Manage Editoria11y results" and "View site reports."
2. If the checker **toggle** does not appear: make sure a z-indexed or overflow-hidden element in your front-end theme is not hiding or covering the `ed11y-element-panel` container, make sure that any custom selectors in the "Disable the scanner if these elements are detected" field are not present, and make sure that no JavaScript errors are appearing in your [browser console](#).
3. If the checker toggle is **present** but never reporting errors or missing items that should be flagged: check that your inclusions & exclusion settings below are not missing or ignoring the elements. It is not uncommon for homepages or views to accidentally insert editable content outside the main element.

[Project overview](#) | [Working Demo](#) | [Issue queue](#)

#### Basic Configuration

**Theme**

Light

**Check content in these containers**

To limit checks to user-editable containers, provide a list of [CSS selectors](#).  
E.g.: `main, #footer-editable-content`  
Default: `main`, falling back to `body`. If elements are specified here and none are found on the page, Editoria11y will not find anything!

**Skip over these elements**

Provide a comma-separated list of selectors for elements to ignore. These should target specific elements (use an asterisk to indicate \*).  
E.g.: `#sidebar-menu a, .card img, .slide [aria-hidden="true"]`, `.feed *`

#### Advanced configuration

Customize tests

A callout box on the right side of the settings page contains the text: "Selects, text fields, fieldsets...showing defaults and current settings..."



# Built automatically with "ConfigFormBase"

```
class EditoriallySettings extends ConfigFormBase {  
    public function buildForm(array $form, FormStateInterface $form_state)  
    {  
        $config = $this->config('editorially.settings');  
  
        $form['setup']['content_root'] = [  
            '#title' => $this->t("Check content in these containers"),  
            '#type' => 'textfield',  
            '#placeholder' => '',  
            '#description' => $this->t('To limit checks to user-editable  
containers, provide a list of CSS selectors.'),  
            '#default_value' => $config->get('content_root'),  
        ];  
    }  
}
```



- src
  - Controller
  - Exception
  - Form
    - DashboardFilters.php
    - DismissalFilters.php
    - EditoriallySettings.php**
  - Api.php
  - Dashboard.php
  - DismissalsOnPage.php

# `.schema` created fields, `.settings` the defaults

```
! editoria11y.settings.yml
  schema
    ! editoria11y.schema.yml
  css
  js
  JS editoria11y-admin.js
  JS editoria11y-drupal.js
  JS editoria11y-localization.js
```

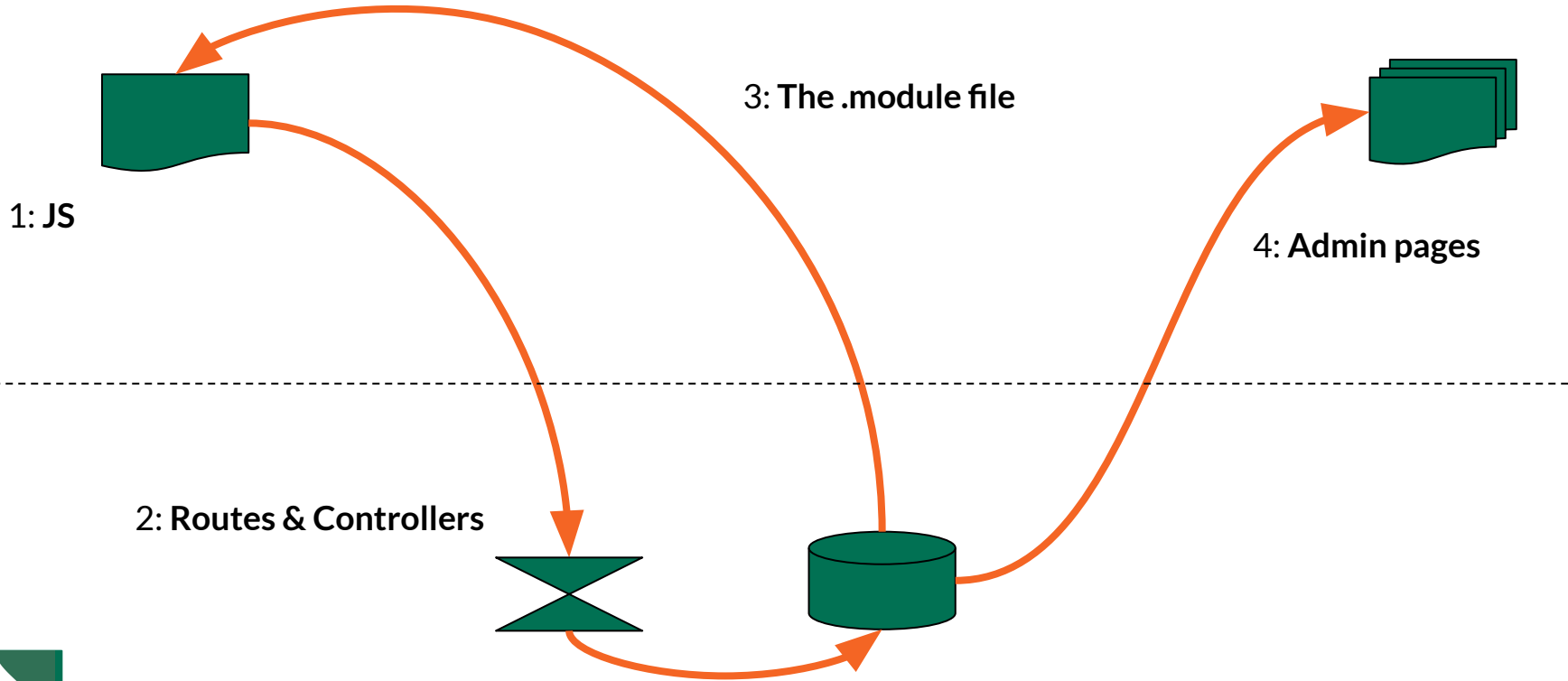
```
1 editoria11y.settings:
2   type: config_object
3   label: 'Editoria11y Settings'
4   mapping:
5     content_root:
6       type: string
7       label: 'Check elements in these containers'
8     assertiveness:
9       type: string
10      label: 'Alert panel assertiveness'
```

```
! editoria11y.settings.yml
  schema
    ! editoria11y.schema.yml
  css
  js
```

```
1 content_root: ''
2 no_load: ''
3 ignore_elements: ''
4 embedded_content_warning: ''
5 assertiveness: 'smart'
6 download_links: ''
```



# And that's how I built Editorial11y v2



---

## Ending with links is mandatory, right?

- [editoria11y.princeton.edu](http://editoria11y.princeton.edu) (demo & docs)
- [git.drupalcode.org/project/editoria11y](https://git.drupalcode.org/project/editoria11y) (source)
- [youmightnotneedjquery.com](http://youmightnotneedjquery.com)
- [drupal.org/docs/develop/drupal-apis](http://drupal.org/docs/develop/drupal-apis)  
[drupal.org/docs/develop/creating-modules](http://drupal.org/docs/develop/creating-modules)

