

Implementing an accessibility-focused design system in Drupal

Amy M. Drayer

jen neveau

Gabe Ormsby



“This is how bad design makes it out into the world. Not due to malicious intent, but having no intent at all.”

-Mike Monteiro

Project goals and vision

Design with intent

Putting principles into action

(Universally) Accessible

Honest

Inclusive

Mindful

Private

Simple

Sustainable



(Universally) Accessible

Deliver content and services where barriers to access are removed for all people to use regardless of technology, format, or methods of delivery.

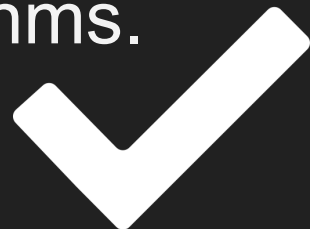
Avoid building to one way of doing or being. Build to be understandable, in a way that allows us to be human and make errors.



Honest

Be transparent.

Provide only accurate content written with non-biased language and clearly identify opinionated content. Fight disinformation, or the act of intentionally deceiving in content and algorithms.



Inclusive

Lead with person-first design, designing with people, not for them. Be aware of our own biases and assumptions, and recognize we are not the user.

Embrace people as complex beings, where average doesn't exist.



Mindful

Make decisions that prioritize user wellbeing, don't build to steal attention, and avoid deceptive and manipulative patterns.



Private

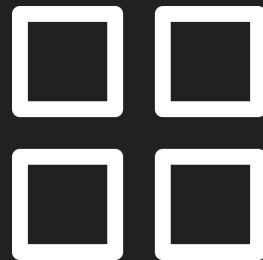
Promote and ensure privacy through security and personal data ownership. Provide these tenets in systems and services to the best of our ability and be transparent where we cannot.



Simple

Simple is challenging; it's curating and cultivating the message concisely and clearly.

It's discovering the most elegant semantic technical solution, and the intentional use of the resources available.



Sustainable

Factor in energy source and consumption for optimizations, from server to client side. Just as people do not deserve a reduced experience, our planet does not deserve to suffer the consequences of bad design and web delivery.


Sustainability is as much about our planet's wellbeing as it is our own, as they are intertwined.



University Libraries' design system

“This is how bad design makes it out into the world. Not due to malicious intent, but having no intent at all.”

–Mike Monteiro, How Designers Destroyed the World

 **Principles**
Core principles that guide all the decisions to holistically develop web assets.


[Go to guiding principles](#)

 **Content**
Learn how to use language to design more thoughtful experiences.

[Go to content guidelines](#)

 **Design**
Find out how we approach the visual elements of our interface with purpose.

[Go to design guidelines](#)

 **Code**
Use components as building blocks to develop new or modify existing content.

[Go to code](#)

(Universally) Accessible

Honest

Inclusive

Mindful

Private

Simple

Sustainable



Accessibility is aesthetic.

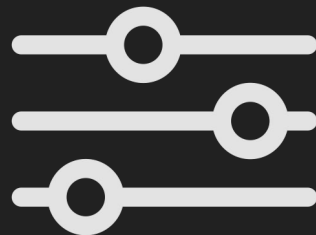
simplify code

simplify selectors ~~#this one #that thing .this~~ button span
:not, ~, +, :first-child, :only-child, :nth-child(),
:first-of-type, :last-of-type, :nth-of-type(), [att=value]

honor user settings to improve accessibility

mind high CPU styles for sustainability

- border-radius
- box-shadow **and** text-shadow
- opacity
- transform
- filter
- position: fixed



Making it work in Drupal

- Provide a web components-based editing system for content contributors
- Theme using web components provided by the design system

**The editing experience
for content contributors**

Removing options...

- Prevents layout and content choices that are counterproductive (break the intent of the design system)
- Saves the content contributor from having to think about irrelevant choices

Setting constraints

- No Layout Builder
- No In Place Editing
- Minimal WYSIWYG toolbar configurations, tailored to specific content (Inline elements, lists, larger-scoped components)

Keep the Design System in mind

- Editing interface matches Design System language for each web component
- Field-level help text links back to Design System for details and rationale
- Staff training similarly references both Design System components and Drupal implementation

Design System-to-Drupal

The Design System gives us web components and the canonical markup for them.

[Links](#)

[Buttons](#)

[Images](#)

[Optimization](#)

[Photos](#)

[Illustrations](#)

[Iconography](#)

[Lists](#)

[Ordered](#)

[Unordered](#)

[Description list](#)

[List group](#)

[Cards](#)

[Forms](#)

Card anatomy

Cards have several pieces that can come together to suit the content. They are presented in the visual order the card presents.

Subsection heading	Optional Use sparingly. When a card represents an entire section of content, the subsection heading may be included.
Image	Optional Use an image if it helps understand the content of the card. Prefer the most optimized level with grayscale.
Content heading	Required The heading summarizes the content in the card.
Card body	Required The card body holds the primary content of the card. It can consist of paragraphs, lists, or list groups. List groups are different from lists by providing more visual separation between each bullet. Use it if the bullets are more separate than a visual breakdown of a composite sentence.
Related content footer	Optional, cannot be paired with the action link



[Content preview with image and related content](#)

The content is not contained to the card, but it is a preview to more content.

Along with an image for the content preview, it has a related content footer.









[More related content](#)

Applying the abstract web components to Drupal

Paragraphs as building blocks

- <https://drupal.org/project/paragraphs>
- Design system components were implemented as Paragraphs
- Allows for nesting of elements (cards in card decks, text paragraphs inside sections)
- Each type contains fields for content and also for editor-accessible settings.

Example paragraph types

Icon	Label	Machine name	Description	Operations
	Action link	action_link	A link for linking to a web form. Background on our Design System Links page .	Manage fields 
	Action link group	action_link_group	A grouping paragraph to arrange a group of action links side-by-side, or to bring a single action link into alignment with paragraphs nearby.	Manage fields 
	Assignment step	assignment_step	Describes an assignment step for Assignment Calculator pages, including step title, description, and percentage of total assignment time for the step.	Manage fields 
	Basic list	ds_basic_list	An ordered or unordered list of items. Options are explained on our Design System Lists page .	Manage fields 
	Block quote	block_quote	A highlighted quotation with attribution. See our Design System Quotations page for usage guidelines.	Manage fields 
	Card	card	A card, presenting a visually and semantically distinct group of content. Guidelines are on Design System Cards page .	Manage fields 
	Card deck	card_deck	A configurable collection of cards. See our Design System Cards page for guidelines.	Manage fields 
	Content section	ds_content_section	A section for grouping a heading and content within a page. Allows custom ids and classes for the section element.	Manage fields 

Page body as Paragraphs

The screenshot displays a content management system interface. At the top right, there is a toggle for "Show row weights". Below this is a section titled "Page body" with an "Edit all" button and a vertical ellipsis menu. The main area contains a list of five content sections, each with a move icon, a weight indicator (2), a title, a description, and an "Edit" button with a vertical ellipsis menu. The sections are:

- Content section: General services, These core services support your work at any level, and any stage of the research process., core
- Content section: Student support services, Services to help students with their studies, assignments, research, projects, and papers., student
- Content section: Instruction support services, Instruction support services help find quality course materials that fit your class. We can also help you integ...
- Content section: Research support services, Research support services provides U of M scholars and academics quick access to information and tools to pl...
- Content section: K-12 support services, Information and resources for middle and high school students participating in U of M programs., k-12

At the bottom left, an "Add Content section" dropdown menu is open, showing a list of content types. The "Add Action link" option is highlighted. A green box highlights the upward-pointing arrow icon next to the "Add Content section" header. The background shows a preview of the page with a "Contact information" section.

Content section fields

A section for grouping a heading and content within a page. Allows custom ids and classes for the section element.

Section heading *

Text format

[About text formats](#)

Rich text for inline text (UMNLib)

- Allowed HTML tags: <a id target rel class="ck-anchor" name hreflang download href>
 <p> <cite> <address> <time datetime> <abbr title> <code> <sub> <sup>
- Global and entity tokens are replaced with their values. [Browse available tokens](#).
- <p> and
 tags will be removed (regardless of what may be stated above.)

Section subtitle



- Allowed HTML tags: <a id target rel class="ck-anchor" name hreflang download href>
 <p> <cite> <address> <time datetime> <abbr title> <code> <sub> <sup>
- Global and entity tokens are replaced with their values. [Browse available tokens](#).
- <p> and
 tags will be removed (regardless of what may be stated above.)

[About text formats](#)

Adds additional text to the section heading that will not display in navigation or other space-limited contexts.

Section items

Add Text paragraph to *Section items*

Section id

id attribute to add to this section.

Section classes

The editing experience in summary

- The "blob of body text" and wealth of rich text options is gone.
- Content editors think structurally about how their content fits into available components.
- Formatting options are tightly constrained to those that make sense within a given component and which meet accessibility requirements.
- Fields are mostly for displayed content, but some turn into configuration as the theme processes them.

**Theming (the Paragraphs)
to fit the design system**

Theme flow as a 3-step process

- **Catch:** Hook into the theming process where Drupal would normally go -- The Twig files
 - For Paragraphs,
`paragraph--paragraph-type.html.twig`
- **Tweak:** Manipulate inbound data where needed to work with design system expectations -- Twig or theme hooks
- **Redirect:** Send tweaked data to theme's design system components

Theme flow

Step 1: Catch

Catch: Standard Drupal template behavior

[...]

```
<!-- THEME DEBUG -->
```

```
<!-- THEME HOOK: 'paragraph' -->
```

```
<!-- FILE NAME SUGGESTIONS:
```

```
  * paragraph--ds-content-section--default.html.twig
```

```
  x paragraph--ds-content-section.html.twig
```

```
  * paragraph--default.html.twig
```

```
  * paragraph.html.twig
```

```
-->
```

```
<!-- BEGIN OUTPUT from
```

```
'sites/www.lib.umn.edu/themes/umnlb/templates/paragraphs/paragraph--ds-content-section.html.twig' -->
```

```
<section class="color-block hero" id="homepage">
```

[...]

"Catch" just means create a template where Drupal is going to look. For a specific example:

Our design system component: Content section

Implemented as: A paragraph of type *ds-content-section*

Catch the theming process at: `our-theme/templates/paragraphs/
paragraph--ds-content-section.html.twig`

Let's dig into that Twig file for the next two steps...

Excerpt 1: paragraph--ds-content-section.html.twig

```
{#
/**
 * This template passes final rendering off to a design system pattern
template
 * in theme's templates/components/ directory. Here, we munge the data just
 * enough to fit the expectations of the abstract DS pattern. That means:
[...]
```

* This approach will allow other Drupal Paragraphs, Block or Views, to also
* rely on the component templates, by ensuring a consistent 'inbound'
* vocabulary.

```
*/
#}
```

Excerpt 2: paragraph--ds-content-section.html.twig

```
{% set section_classes =  
content.field_ds_section_classes|render|striptags|trim|split(' ') %}  
  
{% set section_id = content.field_ds_section_id|render|striptags|trim|split(''  
)|first ?: 's-' ~ paragraph.id() %}
```

This is Step 2...Tweak!

Why tweak? Paragraph data is not necessarily Component data

- Stuff that makes sense for an editor to enter may not perfectly map to stuff a design system component needs
- We alter existing data and add new data when sending a Paragraph to a Component
- Easy stuff can happen in Twig
- Complicated stuff happens in umnlib.theme, mostly preprocess and alter hooks -- 2,000 lines worth (maybe 50% comments)
- Recall the Paragraph entry form for a content section...

Part of the form for a content section paragraph

Section id

id attribute to add to this section.

Section classes

Classes to add to this section element. Separate multiple classes with a space.

Excerpt 2 (again): paragraph--ds-content-section.html.twig

```
{% set section_classes =  
content.field_ds_section_classes|render|striptags|trim|split(' ') %}  
  
{% set section_id = content.field_ds_section_id|render|striptags|trim|split(''  
)|first ?: 's-' ~ paragraph.id() %}
```

Excerpt 3: paragraph--ds-content-section.html.twig

```
{% embed '@components/section/section.html.twig' with {
  content: content.field_section_items,
  heading: content.field_ds_section_heading,
  heading_element: content.heading_element,
  subtitle: content.field_section_subtitle,
  section_classes: section_classes,
  section_id: section_id,
  classes: classes,
  back_to_top: content.back_to_top,
} %}
{% embed %}
```

...And here is Step 3: Redirect. What's going on here?

The "redirect" step relies on the **Components** module

- <https://drupal.org/project/components>
- "The **Components** module makes it easier for a theme to organize its components. It allows themes (and modules) to register Twig namespaces and provides some additional Twig functions and filters for use in Drupal templates."
- Rephrased, allows us to define a special templates directory for components in our theme.info.yml file and use `@components` in Twig to reference that directory.

Excerpt of umnlib.info.yml

...

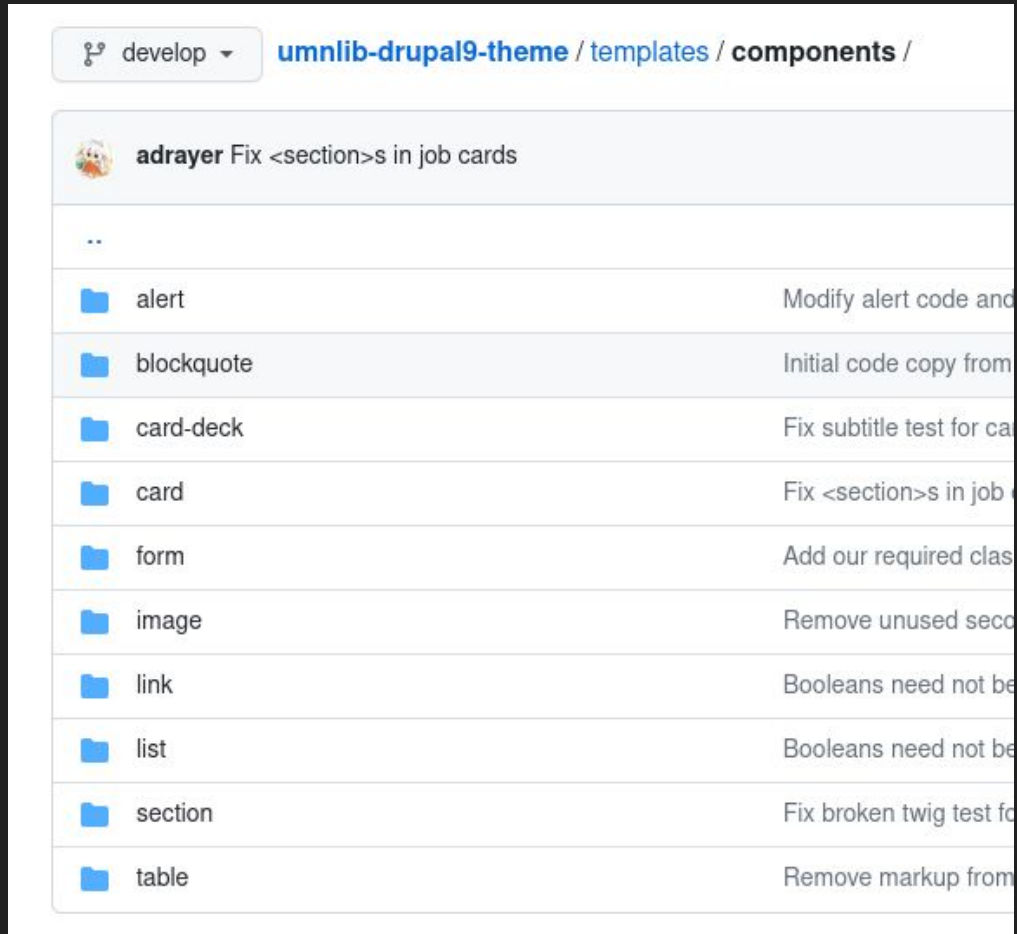
```
components:
```

```
  namespaces:
```

```
    components: templates/components
```

...

What's in templates/ components?



The screenshot shows a file explorer view of the `templates/components` directory in the `umnl-lib-drupal9-theme` repository on the `develop` branch. The directory contains several subfolders, each with a corresponding commit message. The commit messages are truncated on the right side of the table.

Folder Name	Commit Message
<code>..</code>	
<code>alert</code>	Modify alert code and
<code>blockquote</code>	Initial code copy from
<code>card-deck</code>	Fix subtitle test for ca
<code>card</code>	Fix <section>s in job e
<code>form</code>	Add our required clas
<code>image</code>	Remove unused seco
<code>link</code>	Booleans need not be
<code>list</code>	Booleans need not be
<code>section</code>	Fix broken twig test fo
<code>table</code>	Remove markup from

Excerpt 3 (again): paragraph--ds-content-section.html.twig

```
{% embed '@components/section/section.html.twig' with {
  content: content.field_section_items,
  heading: content.field_ds_section_heading,
  heading_element: content.heading_element,
  subtitle: content.field_section_subtitle,
  section_classes: section_classes,
  section_id: section_id,
  classes: classes,
  back_to_top: content.back_to_top,
} %}
{% endembed %}
```

That was the theme template for the content section paragraph type.

Let's go look at the theme template for the content section web component.

Excerpt 1: templates/components/section/section.html.twig

```
[...]  
/**  
 * Available variables:  
 * - content: The content items within the article. Optional.  
 * - heading: The heading for the article. Required.  
 * - heading_element: The appropriate level heading. Required.  
 * - subtitle: Extension to heading, not included in side navigation. Optional.  
 * - back_to_top: Boolean flag indication whether to include a 'back to top' \  
   link. Optional.  
 * - section_classes: Array. Class options from paragraph type. Optional.  
 * - section_id: String. ID attribute value. Optional.  
 * - classes: Classes array passed by parent theme or module.  
[...]
```

Excerpt 2: templates/components/section/section.html.twig

```
[...]  
* This file may be used via twig 'embed' by any number of theme templates for  
* Views, Blocks, or Paragraphs. The calling template is responsible for  
* altering source data to fit the pattern's expected variables.  
*  
* See this theme's  
templates/paragraphs/paragraph--ds-content-section.html.twig  
* for a sample implementation.  
*/  
[...]
```

Excerpt 3: templates/components/section/section.html.twig

```
<section{{ attributes.addClass(section_classes).setAttribute('id', section_id) }}>
{% if subtitle.0 %}
    <{{heading_element}}>
        {{ heading }} <span class="subtitle">{{ subtitle }}</span>
    </{{heading_element}}>
{% else %}
    <{{heading_element}}>{{ heading }}</{{heading_element}}>
{% endif %}
{% if content|render %}
    {{ content }}
{% endif %}
{% if back_to_top %}
    <p><a href="#top">Back to top</a></p>
{% endif %}
</section>
```


Recap: Implementing our Design System in a Drupal theme

1. Use the default `paragraph--paragraph-type.html.twig` files to catch the theme flow
2. Tweak data available to the paragraph type right in the twig template or through theme alter or preprocess functions to fit design system component needs
3. Redirect rendering to a web component template with the help of the Component Libraries module and Twig `embed` functionality
4. The same approach works with Views, Blocks, and other templates

Tweaking revisited: Fixing a thing that's bugged me since 2008

Remember this from our component template `section.html.twig`?

```
<{{heading_element}}>{{ heading }}</{{heading_element}}>
```

`umplib_get_heading_level()`:

A helper function in our theme that finds the appropriate heading level for any heading-like field.

- We know what fields are used for headings and we know what paragraph types they are in (card decks and content sections)
- Instead of hard coding "`<h2>`" or "`<h3>`" in a twig template and hoping it works, we use "`heading_element`"
- Probably rather expensive, but we have Varnish caching in front.

Accessible content

Content strategy

“The University of Minnesota Libraries’ website prioritizes the support of users new to our site and services, and by doing so, supports all users in their goal to find and access our resources and apply them to their work, and our goal to engage and inspire our diverse community of users.”

Content guidelines

Content guidelines

- Use descriptive headings and links
- Use sentence casing
- Write for a 8th grade or lower literacy level
- Use short sentences and paragraphs
- Use bulleted lists

Content audit and remediation

Content governance and workflow

Draft > Pending WCMC review > Published

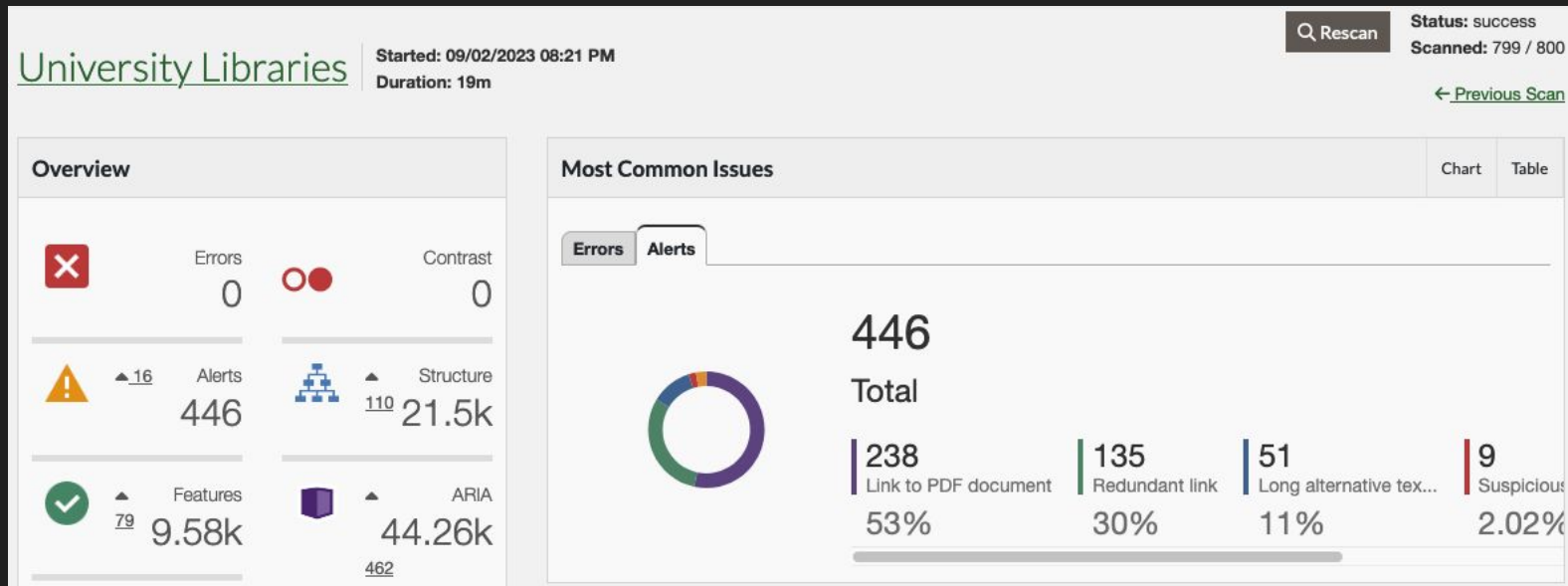
Pending deletion or Archived

Lessons learned

- Content first development worked! (mostly)
- The shift away from WYSIWYG authoring was *hard*.
- Ongoing communication is key.
- Workflow is important, but keep it simple.
- Real content doesn't always fit.

Lessons learned

- It paid off!



Thank you.

Amy Drayer adrayer@umn.edu

jen neveau jneveau@umn.edu

Gabe Ormsby gormsby@umn.edu