OU Campus™ Design Standards

omniupdate.com

Contents

Design Requirements

Validation

All template source files must be pre-validated to W3C standards and accessibility compliance, as required by the institution.

W3C Validation
http://validator.w3c.org

Accessibility Checks
http://achecker.ca
https://wave.webaim.org/
Higher education websites generally have to comply with the following:

- WCAG 2.0 (Level AA)
- Section 508

Source Files

OU requires delivery of static HTML source files for all page types, so that each page will render within a browser, ZIP file recommended.

This includes: HTML, CSS, JS, Server-Side Scripts, Images, Support Files (includes, txt, navigation blocks, Google Analytics, search code, etc.)

Style Guide

Within the HTML deliverables it is advisable to provide a style guide or “All Elements” document to demonstrate the CSS and styling for basic HTML elements used in the editable regions, such as H1 – H6, tables, paragraphs, special image classes, and ordered and unordered lists.

This style guide should contain all the complex design elements and show that they will work even if re-ordered.

Ideal resource structure

The OU Campus implementation process will move all files into the directory structure shown below. If possible, deliver template source files using this structure to reduce possible errors when OU moves these resources.

/_resources
    /css
    /fonts
    /images
    /includes
    /js
    /php
/App_Code
/page-template.html
/style-guide.html

Best Practices

General Practices

OU Campus editable regions are edited very similar to editing a Word document through the WYSIWYG (What You See Is What You Get) editor. Editable content regions should only contain elements that are as similar as possible to this environment. Care should be taken to avoid making the user visually think of how their content will fit into the design, but be more content oriented with general styles applied to elements.

Use a proper heading structure, as set forth by validation and accessibility standards. Below are a couple examples of heading structure, with parentheses around potential items certain headings may be used for.

Bad example:

```
1. h4 (Logo)
      a. h2 (Page Heading)
            i. h1
           ii. h4
          iii. h5
                  1. h2
      b. h4
```

Good example:

```
1. h1 (Logo)
      a. h2 (Page Heading)
            i. h3
           ii. h3
                  1. h4
                  2. h4
                        a. h5
                              i. h6
                        b. h5
                  3. h4
          iii. h3
      b. h2
      c. h2 (Footer maybe)
```

Headings can be styled however you would like, such as making an <h3> appear larger than an <h2>, but the actual source of the heading should follow proper nesting as shown in the good example above. You could also provide heading CSS classes to change the look and feel from the default styling heading if needed, such as what Bootstrap offers: https://getbootstrap.com/docs/4.3/content/typography/#headings

HTML tags should be placed in their proper order, as defined by validation and accessibility standards. Elements that appear in a different ordering may be "fixed" by the WYSIWYG, so the styling should account for that.

Bad example:

```
<a href="page.htm"><p>Header</p></a>
```

Good example:

```
<p><a href="page.htm">Header</a></p>
```

Keep form elements within form tags, such as repurposed select elements. This is particularly important for search boxes or other form elements that will be hooked up to 3rd party systems, such as Google Custom Search.

Avoid using empty elements when possible.
Often times, an empty span will be used for an icon or an empty div for other styling purposes. When these are edited in OU Campus, the elements will either be removed by the WYSIWYG or a non-breaking space will be added, unless OU adds a workaround for the user at the time of implementation through the use of some special transformation or Component.

Designs should not break if an element is left out.
As an example, if a user doesn't provide the first heading of a content area, the rest of the content and page styling should still display correctly.

- Users will often choose to turn on/off different features of a page, so sibling structures should not be relied upon to create spacing.

HTML5 Semantic Tags

Using semantic (or outdated) tags in content areas will create a more difficult user editing experience in the OU Campus WYSIWYG. The editor supports the following tags:

- , <p>, , , <h1> through <h6>, <a>, <img>,
, <hr>, <iframe>, <blockquote> (without a citation), <sup>, <sub>, <table>(all elements), <ul>, <li>,

Icons

Do not use the <i> tag to display items from icon libraries such as Font Awesome, as the element will be flagged during OU Campus accessibility scans. is a better alternative that achieves the same effect without triggering the accessibility errors/warnings.

CSS & JavaScript Placement

CSS should be placed in the head tag of the pages.

JavaScript can be placed in the head tag of the page, but is normally the last code before the closing body tag for optimization purposes.

Avoid using straight JavaScript code in the HTML.

- Move JavaScript to external files when possible.
- If necessary, add it to the head or before the end of the body.

Avoid placing CSS/JavaScript directly above or below the element it is modifying. This will make it more difficult to create a simple editing experience for the user.

Limit the number of CSS/JavaScript libraries and consolidate them into 1 CSS file and 1 JS file when possible. If necessary, break out the items that are only needed on a certain page, but this should be kept to a minimum as users can typically place content elements within any page's editable regions and so those added elements should display correctly in all locations.

CSS & JavaScript Targeting

CSS targeting should allow page elements to be placed anywhere within the core content and still retain its styling. The following are recommended practices for CSS and JavaScript targeting.

- Avoid inline CSS Styles.
- Avoid targeting by ID (both CSS and JavaScript).
- Avoid styling via direct descendant selectors. This can create styling issues when editing in OU Campus, though the published page will appear as provided.
    - *Bad example:*
      `.bodyclass > .content > h1`
    - *Good example:*
      `.bodyclass h1`
- Avoid using required distant ancestor classes (i.e. a class/id on the body).
    - Elements should have a default appearance without distant ancestor classes.
    - Distant ancestor classes should only modify how elements look on certain pages; elements should not depend on them.

JavaScript Document Modification

Most designs can be accomplished without JavaScript, so there is typically no need for JavaScript to modify the whole document. This can potentially be a large issue causing the website to fail to render if there is any issue that causes the JavaScript document modification to break. Too many document modifications may also cause errors with other functionalities that OU may add for the user for convenience, or from separate projects. In some cases, a JavaScript modification may also interfere with the user's editing experience in OU Campus.

Not all JavaScript document modifications are bad. Some good uses for JavaScript document modification include:

- Image sliders
- Thumbnail galleries
- Large mega/mobile navigation
- Side navigation collapsing
- Accordions/Tabs

None of the core settings for the above features are typically adjusted by the user and can be safely added without impacting the content editing experience.

Avoid relying on specific content in order to determine a JavaScript modification for document elements. For example, navigation highlighting should not rely on a heading in the content area to be the same as the text content of a link in the navigation, as most text should be editable by the user in OU Campus.

If jQuery or similar libraries are used in the design, there should be a single reference to the included library so OU developers will know that jQuery is available to use for custom features. Make sure the same version of jQuery is used on each design file.

Limit SVG Usage

SVG images are difficult to manage in OU Campus and create a less than ideal user experience. Limited use of SVG images should be okay, such as in the header or footer (or other places that won't be edited as often), but they shouldn't be used in content regions. It is recommended to use .jpg or .png for images that will be placed in editable areas.

Styling for OmniUpdate Modules

Some OU modules come with default output that only needs to be styled instead of recreated. The default output for these modules can be provided upon request for applicable modules.

Custom styling can be accommodated for most of the modules, but it may increase the OU implementation effort and should be discussed with OmniUpdate before deviating from the standard output. Custom module outputs should generally contain the same basic elements and functionality as the default output to ensure there is no loss of expected functionality or additional elements that may not be accounted for.

Forms Styling

OU Campus implementations typically include Live Delivery Platform (LDP) forms, so sample HTML markup should be provided for each standard form element. If no styling is provided, OU Campus will output a simple Bootstrap design that is not guaranteed to stylistically fit well with the provided design.

- Basic form elements: Text input, text area, multi-select, checkboxes, radio buttons, dropdown selects, error messages for success/failure, helper text.
- Advanced form elements can be added by more advanced users. These include: legends, classes to an element block, fieldsets, size/rows/cols attributes.

Consistency & Combining Templates

Because OU Campus templates are customized to each design, it is possible to "combine" templates so that similar design layouts can be managed with a single OU Campus template. Some examples of options that may be set by the user are: allowing users to potentially add or remove a right column on an interior page template, or switch between a slider or header image, or even disable a region with a simple dropdown option associated with a page.

The more similar the HTML markup is for different design templates, the easier it is to combine these templates in OU Campus. To facilitate these types of options, we ask for as much consistency across design variations as possible. Adherence to the following guidelines will allow for the greatest flexibility and ease of use for the user, even when templates aren't combined.

Common/Global Code

Use a consistent header/footer structure and main content wrappers/regions across all templates. If an element is the same across all layouts, it should be documented as "global" with identical source code and dependencies across all pages.

CSS/JS should be shared across templates whenever possible, particularly for similar page layouts, rather than using different CSS/JS files for every page.

Avoid using similar-looking layouts with vastly different HTML. Every page should have the exact same HTML markup surrounding the core content when possible.

Simplify HTML Structure

Keep the use of elements to a minimum, avoiding excessive nesting. This allows for more versatility across less editable regions, which allows for a simple and higher quality user experience while managing their content.

Bad example:

```
<div class="page-content">
    <div class="header"><h1>Header</h1></div>
    <div class="content"><p>Content</p></div>
    <div class="content2"><p>More Content</p></div>
</div>
```

Good example:

```
<div class="page-content">
    <h1>Header</h1>
    <p>Content</p>
    <p>More Content</p>
</div>
```

Avoid nested lists inside tabs/accordions. It will be more difficult for a user to manage these elements and their use may indicate the design strategy could be improved.

Avoid styling on unordered or ordered list elements.

Bad examples:

```
ol.bulleted
ul.bulleted
```

Good examples:

```
.sidebar ul
.body-copy ol
```

Avoid CSS classes on each element. This translates into more work for the end user to curate content with the desired styles and layout.

- Try to standardize the header, paragraphs, styles, etc., using one style for each tag, and apply classes to specific items only when necessary.

Complex Elements

Use consistent code blocks (accordions, sliders, tables, lists, etc.) across all templates. This helps the users know which snippets to add to each region, when multiple blocks look nearly identical and the differences are not obvious to the end user.

Avoid section-specific elements. Each complex element should work when placed in different content areas on a page or on different page types. If an element must be limited to a specific page, please be sure to call that out to the OU onboarding team so additional code can be added to help encourage the users to avoid placing complex elements in the wrong region.

Ensure each element will work when different amount of content is used, such as adding very large text strings, or removing content to use a short text length.

Global, Section, and Page-Specific Resources

When designing for OU Campus, it is important to keep in mind the intended use for each part of a template, as the OmniUpdate implementation team will break apart the design into different pieces to allow for an editing experience that makes the most sense based on the school's specific needs.

Global Elements

Any element that is exactly the same on every page on the site would be considered global. In OU Campus, this is usually accomplished using an include file that would either be edited in source code, or through a special PCF page that publishes an include to allow less technical users the ability to edit a more defined element. Common parts of the site that would be considered global are:

- Header
- Footer
- Common code in the <head> that is shared across all pages, such as CSS references.
- Common code just below the closing </body> tag, such as JS references.
- Main navigation at the top of every page, often with dropdowns or other "mega-nav" type functionality.

Section Based Elements

In OU Campus, a section is a folder and all the resources necessary for the elements that are common across all pages in that folder/section. Most OU Campus implementations will make use of two files for this purpose, the section properties file (_props.pcf) and a local navigation file (_nav.inc).

Section Properties

The section properties file is generally used to allow the user to set the text for the breadcrumb for that particular level. Following this logic, each section should have a landing (index) page, as clicking the breadcrumb for any level should land the user at the section's landing page.

If the design indicates that additional information is shared across a section, the section properties file may be customized to allow for additional section-based data to be entered.

OU Campus Directory Variables may also be used to define section-level elements. For example, if the athletics section of the site will have a custom header that is not used elsewhere within the site, an additional section header file may be created and pointed to through the use of a Directory Variable.

A section may also have its own custom pre-footer section that needs to be managed on a per-section basis.

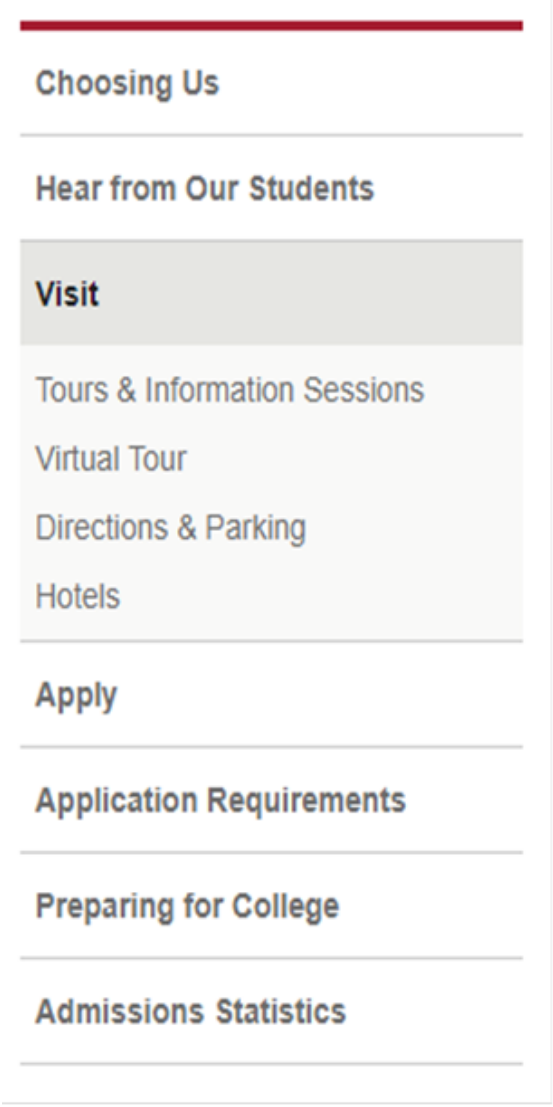
In short, these section-specific elements should be considered with the design files, so that these elements are consistent across all pages in a section. It is also very useful to provide this information to the OU project manager for a given project, so they can ensure that the OU Campus implementation fits in line with the expectations that have been set with the customer during the design phase.

Local Navigation

OU Campus uses local navigation files for each level of navigation, which can then be nested through the use of various scripts. As such, a simple ul/li structure for navigation is preferred to allow for an easier editing experience for the end user.
Notes should be provided defining information about how the nested navigation is intended to function. Important information includes:

1. Which way is the nested navigation going?
    a. Current > child > grandchild
    b. Grandparent > parent > current
2. How many levels will the navigation include?
3. If there is an accordion-type functionality built into the navigation design, can you click to the index page for the section and also expand to see its children links?

OU Automatic Nesting Details



This section contains some strategy decisions related to the "Local Navigation" section above, as it outlines the standard way that these types of navigations are generated in OU Campus. For this section, it is assumed that every page in a section will appear in a single level of the navigation along with a link to the index page of each child section. It is also based off the following sample screen shot. Understanding how this data is managed and implemented into OU Campus may provide insights into the most accommodating designs for OU Campus.

Automatic nesting is set up with one _nav.inc file per section. When you create a page in OU Campus, the "autonavigation" feature allows you to automatically append the new page's link to the navigation, after which you have to edit the links for that section manually. To trigger nesting, a link to the index page of a child directory has to be placed in the _nav.inc file. This can happen manually, or automatically through "autonavigation" in a new section template. Our server-side script then uses the pattern of looking for index pages in the navigation to trigger the nesting automatically. In all cases, we can also limit how many levels are nested as well, but it usually ends up being 2 levels (parent & child) or sometimes 3 levels (parent, child, grandchild). There are some variations on how this nesting can be done with our default code, but it can essentially be boiled down to the following (assuming they all have the linking set up to the appropriate index pages):

- **Current-Up**: the current section's navigation and any parent navigation files are nested. This would look like the screen shot provided if you were looking at the navigation as it would appear on one of the pages in the sub-menu, such as "Virtual Tour".
- **Current-Down**: the current section's navigation and any children navigation files are nested. This would look like the screen shot provided if you were looking at the navigation as it would appear on one of the pages in the parent section, such as "Visit" or "Apply".
    - A common variation of this option is that schools would like to only show the nesting if you are on the index page for a section. In the case of the screen shot provided, the nested items would only show if you are on the "Visit"

section's index page. In this case, once you clicked into that section, then the sub-items would become the navigation.

The provided screen shot is an example of two levels of navigation. The parent navigation could be for a school's `/admissions/` section, while the child navigation lists pages within a subdirectory, such as `/admissions/visit/`. In this example, there are two `_nav.inc` files controlling the content of this navigation:

- /admissions/_nav.inc
- /admissions/visit/_nav.inc

In this example, the "Visit" link will be placed in `/admissions/_nav.inc` and will link to `/admissions/visit/index.html`, which triggers the nesting of the visit navigation. **If there will be any feature to collapse this nesting, it should not include the word "Visit" as there would then be no way to navigate to the "Visit" index page without duplicating the link in a location that is clickable.** A common solution to this potential design issue is to create an arrow to the right of the "Visit" link that can be clicked independently to trigger the collapsing. Alternatively, a navigation does not have to allow for users to expand/collapse the navigation and can be automatically expanded or collapsed depending on the page the user is on.

Page Specific Elements

This category includes any code that is specific to an individual page. This could include basic content, or more complex code blocks that would be implemented with a more advanced solution than using the simple editor, such as Table Transformation Snippets or Components. Any code that is intended to only be used on a specific template (and shouldn't be reused on another page type) or is expected to be edited in a more advanced way should be documented and discussed with the OU onboarding team to ensure there is an alignment on expected effort.