Module: UI/UX

HTML/CSS – Intermediate and Advanced Topics.

*We’re taking a quick break after some intense units in React and Redux.*

*Agenda:*

*Semantic HTML*

*Response web Design & Mobile-first design*

*CSS Grids*

1. Hypertext Markup Language
   1. Hypertext links you somewhere
   2. Markup is the rendering of what to show
2. HTML is block or inline.
   1. Block elements force a line break after the element
   2. And take up the full width of the parent by default.
      1. <div>
   3. Inline elements allow other elements to sit on the left or right.
   4. We cannot set width and height.
      1. <span>
3. Semantic HTML
   1. This is about *readable* HTML – it should give us the meaning of the page components.
      1. What is the content?
      2. What role does the content serve?
      3. How is the content related to the page/other content?
      4. Tags are used by browsers to build/render the page and search engines to find/index the page.
      5. So the more specific tags can provide extra information to the browser and search engine.
      6. Does the web page make sense *without the browser?*
   2. Rather than “div” everywhere, we would specify:
      1. Header
      2. Section
      3. Article
      4. Nav
      5. UL
      6. Footer
      7. Blockquote
      8. Strong
      9. Em
      10. Mark
      11. Cite
      12. Etc etc etc.
   3. Basically – rather than just throwing out “div”s everywhere, let’s actually tell our browser and our users what we’re showing them.
      1. This is important for SEO
      2. And especially important for accessibility
         1. Visually impaired users using screen readers will rely on our semantic html for parsing data.
      3. And makes it easier for us to provide hooks for CSS styling.
   4. ARIA elements (accessibility rich internet application)?
   5. Avoid using semantic tags for styling purposes if the meaning does not match.
      1. Don’t use a header tag just to make a paragraph stand out.
4. CSS
   1. Cascading Style Sheets
   2. Defines the appearance of website via rule-sets applied to HTML tags, IDs, and classes…
   3. If multiple CSS rules target the same element,
   4. Those styles **cascade** to create the final styling.
      1. The **lowest** styles take precedence.
      2. This resolves styling conflicts
   5. If we want to take a style and have it take precedence, we use the !important keyword after the styling.
5. Responsive Design
   1. Web design with a variety of screen sizes & devices in mind
   2. Modern development is for multiple browsers and devices – page requirements vary.
   3. Some different rules for different screen sizes & orientations, but focus on maintaining DRY in CSS.
   4. In other words, **MOBILE FIRST (progressive enhancement)!! Not desktop first (progressive degrading)**
      1. Only once we get the mobile-friendly site functioning,
      2. **Then** we expand to tablet/desktop scale.
   5. 92% of internet users experience the internet primarily through their phone.
6. Tools for responsive web design
   1. Avoid sizing in terms of pixels
      1. We should prefer proportion based sizing, like percentages or em/rem.
   2. CSS properties like min-width and max-height
   3. CSS tools like flexbox and CSS grid, or CSS libraries like Bootstrap.
   4. @media queries
      1. For checking screen size and displaying different styles.
      2. Below is for a cell phone screen.

*@media* screen and (max-width: 568px) {

*.selector* {

//rules

}

}

1. Advanced targeting
   1. The below is targeting tablets in landscape.

*@media* screen and (mid-width: 768px) and (max-width: 1024px) and (orientation: landscape) {

*.selector* {

//rules

}

}

1. Flexbox
   1. A *linear* CSS system for 1-d array arrangement of content.
   2. The parent container must be set to display: flex
   3. And child elements will follow rules set on the parent, including direction, spacing, alignment, and resizing.
   4. We can quickly achieve positioning
   5. And can be highly adaptive to viewport size.
   6. Justify-content
      1. positions content on the main axis direction
   7. align-items
      1. positions content on the cross-axis direction.
      2. Baseline – “bottom of text”
   8. Flex grow
      1. Determines the **rate of growth** of each element **respective to one another.**
      2. If one item has a flex-grow 1 and another has flex-grow 2
         1. For every 300 pixels that the screen size grows,
         2. Item one should grow 100px
         3. And item 2 should grow 200px.
   9. Media queries
      1. Can happen **inside** the selector as well as **outside.**

*#container* {

display: flex;

flex-direction: row;

@media screen and (max-width = 700px) {

#container {

//rules

}

}

}

1. CSS Grid
   1. 2-d grid system
   2. Parent must be set to **grid**
   3. The number of cols and rows can be set using grid-template properties
   4. Width and height of columns can be absolute or relative
      1. As pixels, or as a fraction/pct of available space
   5. Child elements are added to grid as they appear in html
      1. L->R then T->B

*.container* {

display: grid;

grid-template-columns: 1fr 2em 1fr;

grid-template-rows: 200px 100px;

align-items: start;

justify-items: center;

grid-gap: 10px;

}

*.container* {

display: grid;

// the backslack indicates column;

// and we repeat it 4 times, with 1frame? fraction? each.

grid-template: auto/repeat(4, 1fr);

grid-template-areas:

"nav nav nav ad"

*/\* "news content " ; \*/*

}

//these are going to be used in the above container to set up our grid template.

*#navigation* {

grid-area: nav;

}

*#big-ad* {

grid-area: ad;

}

*#news* {

grid-area: news;

}

1. CSS preprocessor – SASS
   1. Sass - Syntactically awesome style sheets
   2. Variables in Sass begins with a $ sign.
   3. In addition to font-size, color is a godo use case for variables.
   4. Variable names are much more readable than hex or rgb color.
   5. It also offers nesting.
      1. Nesting can provide a visual hierarchy while working with child selectors.
      2. Lement.Nesting can provide a visual hierarchy, making your stylesheets more redable.
   6. Sass provides color functions
      1. Function($variable, $changeamt)
2. Web design principles
   1. Visual hierarchy
   2. One of the core techniques which are applied to the design process.
   3. It strives to present content in a manner such that users can comprehend the level of importance for each element
   4. We can implement this by providing different color gradients
      1. For example, for highlighting important or less important text, or links.
   5. Buttons?
      1. Tips on primary button
      2. Primary color for brand.
   6. Security Considerations for UX/UI Design
      1. The more complex a website is for the user, the more likely they are to accidentally skip or incorrectly follow security protocols
      2. So your site should be intuitive and simple.
      3. Make it clear what data is required and where it will be used.
      4. A secure UI helps control access to your site. Limit data access so that only registered users can view and share what they want.
      5. Adjust settings so administrators and users can select who can share content.
      6. Use end-to-end encryption to build trust.
      7. Don’t use emails as usernames!
      8. Always require strong passwords
      9. Ensure error handling doesn’t compromise the site’s safety
         1. (When a wrong password is entered, don’t spell out the email)
      10. Don’t use two-factor excessively, but **do** use it for anything involving financial transactions.
      11. When possible find alternatives to passwords for secure authentication, like biometric authentication.