Lecture 4: Web Content

CS-546 – WEB PROGRAMMING

Recap and Intro

How does the browser get the data?

The browser makes a request.

- Has headers
- Has a body

The server receives the request

- Parses the request
- Figures out what to return

The server sends a response

- Has headers and such
- Has a body

The browser interprets the response

This week

What are some of the types of HTML elements we'll be using this term?

- Text
- Tabular data
- Lists
- Organizational / layout elements

How can we structure our data meaningfully?

- What does it mean for our data to be structure meaningfully?

Assignment 2

Lecture 4 Repositories

Lecture Code

- git clone https://github.com/Stevens-CS546/Lecture-4.git

Express and Static Files

What are static assets?

Last week, we saw a server that was setup to send different content based on what route was requested.

In web vernacular, an asset is something that your page uses:

- CSS stylesheets
- JavaScript files
- Images
- Fonts

Static assets are, simply, assets that don't change. This means that they are not dynamic content. You will often need many static assets to build out your web page.

Setting Express to serve assets

Express has pricelessly one inbuilt middleware for you to use: the static middleware. You can see that in **server.js** for lecture 4's code

This will allow you to access your static assets by going to the following path:

- /assets/path/to/file.ext

Express is seeing that you match the /assets path, then navigating inside your asset folder until it finds /path/to/file.ext; if it finds it, it will serve the file.

Sending a file through a route

Sometimes, you want to simply return a file that's not a necessarily asset. Some reasons are:

- Serving different HTML pages or CSS files to different users
- Using a pretty url to represent a file that has to be downloaded
- Proxying content and manipulating it

For this, you can use two methods on the response object:

- response.sendFile(file.ext): this method will send a file to the user's browser; if the browser can render it, it will render it; else it will ask the user to download it.
  - Useful for sending HTML, CSS, image, etc files
- res.download(file.ext); this method will send a file to the user to be downloaded.

Today's Code

Today's repository is very simple, and serves mostly as a way to get static assets and serve static files

- `git clone https://github.com/Stevens-CS546/Lecture-4.git`

We will walk through the HTML together throughout the lecture.

Introducing HTML

What's in an HTML Document?

An HTML has a series of elements
- Open tag plus attributes and properties
- Nested elements

Some very important
- HTML Doctype
- HTML Element
  - Head Element
  - Body Element

Elements can be identified by an ID
- ID can only be used once per document

Elements can identify a group by their class

Elements are described in the document and rendered in the DOM

Improving our HTML Documents

This week we're going to create a meaningful document about different types of coffee, and explain how to make it marked up in a sensible way.

We will learn about:

- Formatting text
- Organizing our data
- Lists
- Tabular data

Reusability, repetition

For web programming, it's very necessary to think of everything based in terms of reusable components. There's a lot of repetition in web programming, where you're displaying many different instances of similar data

- Every tweet has all the same info
- Every blog post has a title, time, body, etc.
- Product descriptions all have prices, titles, etc.

Because of this, we're going to look at our programming in terms of:

- Is this reusable? If so, how?
- How can I make this component accessible?
  - More on this when we get to forms next week.

The Browser's Only Half The Battle

The web isn't just accessible via a browser. As modern web developers, we have to care about:

- Screen readers
- Search Engine Crawlers / Other AI

There is a growing movement to make the web more accessible

- Leveraging HTML's strengths
  - Navs in the nav
  - Labels in forms
  - Using headings properly
  - Attributes to help screen readers
- Making designs accessible
- Tables for tabular data only
- Make sure you can navigate via keyboard

Separating Style and Content

Before we can think in terms of organizing our data meaningfully, we need to understand what HTML does *not* accomplish; the way your document looks.

- **Elements are used to describe your data; CSS is used to style your data.**
- While browsers give many native styles to elements by default, elements are not inherently used for styling. This is why tags for bolding and italicizing text, or changing fonts, were deprecated in HTML5.
- There needs to be a clear separation between style and content; any overlap is a happy coincidence.

While writing HTML, thinking in terms of content first, then styling often leads to more logical, and easier to style documents.

Types of Text

Across the web, text is used to portray many different types of things.

- Headings / Titles in your content (h1, h2, h3, h4, h5, h6)
- Regular paragraph (p)
- Generic groups of text / adding custom definitions or functionality to text (span)
- Emphasized text (em)
- Important text (strong)
- Addresses (addr)
- Citations (cite)
- Abbreviations (abbr)
- Quotes (blockquote)

Using the right kind of element to describe text is very important for SEO, non-browser accessibility, and readable code.

- Even without different styles, those tags help readers understand their document.

The layout of your content

There are many elements that describe the layout of your content

- How to navigate content / your document (nav)
- Grouping your content into sections that have something to do with each other (main, section)
- Denoting a header for content or your document (header)
- Denoting a footer for content or your document (footer)
- Grouping content into a self-contained article (article)
- Stating that certain content is secondary (aside)
- Grouping divisions of content (div)

List Data

Lots of data you'll see and create is some form of a list

- Unordered lists state that the order of the items in the list don't matter (ul)
- Ordered lists state that the order of the items in the list have some sort of meaning (ol)
- Each entry in a list is a list item

You'll very often find see nested lists

```
<ul>
    <li>Item 1</li>
    <li>Item 2
        <ul>
            <li>Subitem</li>
        </ul>
    </li>
</ul>
```

Tabular data

Data is also often presented in a table format. Each table has:

- A table element (table)
  - A table header (thead) (optional)
    - A table row (tr)
      - Multiple table header cells (th)
  - A table footer (tfoot) (optional)
    - A table row (tr)
      - Multiple tada table cells (td)
  - A table body
    - Multiple table rows (tr)
      - Multiple table data cells (td)

Meaningfully grouping a news article

A news article is easily represented properly in HTML.

- Article
  - Header
  - By line (Subheader)
  - Body paragraphs
  - Footer with comment form

Meaningfully grouping a recipe

Just because the tag is ‘article’ doesn’t mean it just has to be news! The article is “an article of content”.

- Article
  - Header: title of recipe, possibly details like cooking skill required
  - A list of ingredients
  - Body paragraphs explaining how to cook recipe
  - An aside with nutritional information

Referencing Assets

Relative: When you specify a path as a relative location, the browser attempts to find these assets relative to your current location.

- When you are at http://localhost/blogs/ and use a relative path of my_image.jpg and styles/background.png, your browser will attempt to find the resources at http://localhost/blogs/my_image.jpg and http://localhost/blogs/styles/background.png respectively.

Root Relative: Similar to relative, you can have *root relative* paths; these will be relative locations based on the root of your host (it will not take the current path into account)

- When you are at http://localhost/blogs/ and use a root relative path of /images/my_image.jpeg your browser will attempt to find the resources at http://localhost/images/my_image.jpg

Absolute: You can reference elements by an entire URL (protocol, host, path, etc) and your browser will look for these directly

- When you are at http://localhost/blogs/ and use an absolute path of http://localhost/images/my_image.jpeg your browser will use that URL to locate the image

Referencing External Assets

Images

- You reference images as normal HTML elements. The image tag is the *img*

Stylesheets

- In the head of your document, you can specify CSS stylesheets to apply to your document using the *link* tag; these will be loaded in order of tag appearance

Scripts

- You reference scripts using the *script* tag.
- Your script files should almost always be placed right before your closing body tag; this will allow your browser to render the page and *then* add function to it, rather than locking the page up to perform JavaScript tasks while the page is still loading. Scripts will be referenced in the order you include them.

Updated boilerplate

There have been several updates to the boilerplate repository. You can get them in two ways:

Navigating to an existing boilerplate repository that you have cloned and running the command:

- `git pull origin master`

This will pull in changes from the repository, assuming you did not make any conflicting changes yourself.

You can also re-clone the repository to a new directory.

- `git clone https://github.com/Stevens-CS546/Boilerplate.git`

I have provided comments to the server.js file to explain the small changes we have made; adding support for static assets, and adding a default route that sends an index.html file as its response content.

Validating HTML

For this course, the validity of your HTML is highly important.

Having valid HTML means your browser does not have to guess how to fix it, which can lead to drastically wrong web pages and pages that cannot be made sense of.

The w3 website has an easy to use validation service that tells you issues and proposed solutions:

- https://validator.w3.org/#validate_by_input

You should view the source of your page, copy, and paste it all into the HTML validator's 'direct input' section before submitting HTML in this class.

You should strive to write as perfect HTML as possible.

Twitter Bootstrap and jQuery

If you check out the updated boilerplate, you will see that we have some references to a few files we did not create:

Bootstrap Files: bootstrap.min.css and bootstrap.min.js

- These files are Bootstrap's CSS and JS files. Bootstrap is a frontend framework that was created to assist in making internal tools for Twitter. It is now used by many developers to make fast and mobile friendly frontend websites.
- For the sake of this course, you will be allowed to use Twitter Bootstrap: **you must also be prepared to demonstrate that you can write HTML and CSS without it**. There will be many times you have to demonstrate your own knowledge of HTML and CSS.

jQuery

- https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js
- jQuery is a library to make DOM manipulation easier. Years ago, it changed the face of web development forever.

The DOM and JavaScript

What is the DOM?

The **DOM** (Document Object Model) is how the programmer / browser interacts with the HTML document.

- The DOM has an API to access and manipulate the document. Each element is represented in the DOM.
- You can access the DOM with JavaScript.
- You can think of the DOM as the document-in-memory, and you can manipulate many aspects of it. This leads to programmers being able to create extremely powerful applications.

```
<!DOCTYPE html>
<html>
  <head>
  <body>
      <p id="first" class="content">Hello, world!</p>
      <p id="second">Hello, class! I've been dynamically updated!</p>
    <p class="content">
    <script src="lecture_1.js" type="text/javascript">
  </body>
</html>
```

Where does it fit in?

The DOM is a programming interface for HTML.

- The rendering engine takes in the HTML document
- The rendering engine parse the HTML into the DOM tree
- The rendering engine takes the DOM tree and creates the render tree
- The rendering engine paints the render tree

You can then manipulate the web page through the DOM, which is accessible via JavaScript.

- You will target DOM elements
- You will manipulate them
- The rendering engine will recreate that portion of the rendering tree.
- The rendering engine will repaint.

Why is this important?

Being able to manipulate your web page in real time through the DOM API allows you to do many, many things :

- Enhance the functionality of your web page
- Update data on your web page to reflect the user's actions
- Turn web pages into robust web applications

Modern web applications constantly mutate the DOM

A practical example of using the DOM

You are creating a page with a comment box and list of comments. You've created a very controversial post that will definitely spawn a large amount of comments.

You would manipulate the DOM in three ways:

- Every 5 seconds you would poll the server in order to check for new comments; if there are new comments, you would use the DOM to create and insert new elements with the comment info that would then appear on screen
- When the user submits a comment, you will use the DOM to:
  - Retrieve their new comment information, submit it to the server, and add their new comment to the page
  - Reset the comment box form to its default state

DOM Events

There are many events that you can listen for, representing interaction between the user and the page or the page and other resources. Some common events to listen to are:

- Users hovering over an element
- Images loading or failing to load
- Scrolling to occur
- Forms to be interacted with
- Keys to be pressed
- The DOM to be modified.

Previewing Browser Based JavaScript

If you take a look at `/static/js/main.js`, you will see a tiny bit of browser based JavaScript!

The comments will walk you through what is occurring. You'll notice that the syntax is familiar, but that we seem to be using an awful lot of global variables! That's because in the browser, we have access to many variables by default that are used by the browser, as well as any variables that are global in any scripts.

Preparing for Next Week

Lab 3

Lab 3 has been posted! Lab 3 requires you to write a semantic document; remember to validate it before submitting!

You will be submitting your entire package, excluding your **node_modules** folder. Remember to save any packages you use to the **package.json file**.

Assignment 2

Assignment 2 has been posted! It is due in two weeks, before lecture.

Assignment 2 will combine some skills with Node, some skills with Express, and a touch of HTML!

Reading

Next week, we will begin to use Forms and Node together even more to start changing the page as we go along!

Read up on Effective JS

- http://ejs.co/

Read up on forms

- https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Forms

Learn about about Twitter Bootstrap

- http://getbootstrap.com/

Read up on what jQuery is, why it happened, and what it accomplishes

- https://en.wikipedia.org/wiki/JQuery

Questions?