Task #02

How to web interacts

IEEE

Created by: Muhammed Kasem

Table of Contents

The Philosophy of Semantic HTML

Understanding Semantic Meaning in Web Structure

Semantic HTML made developers shift from using and to now mainly using the evolved <header>, <nav>, <main>, <article>, <section>, and <footer>. Why? you may ask; well, the simple answer is they give meaning about the content they contain, transforming how both machines and humans interpret web pages.

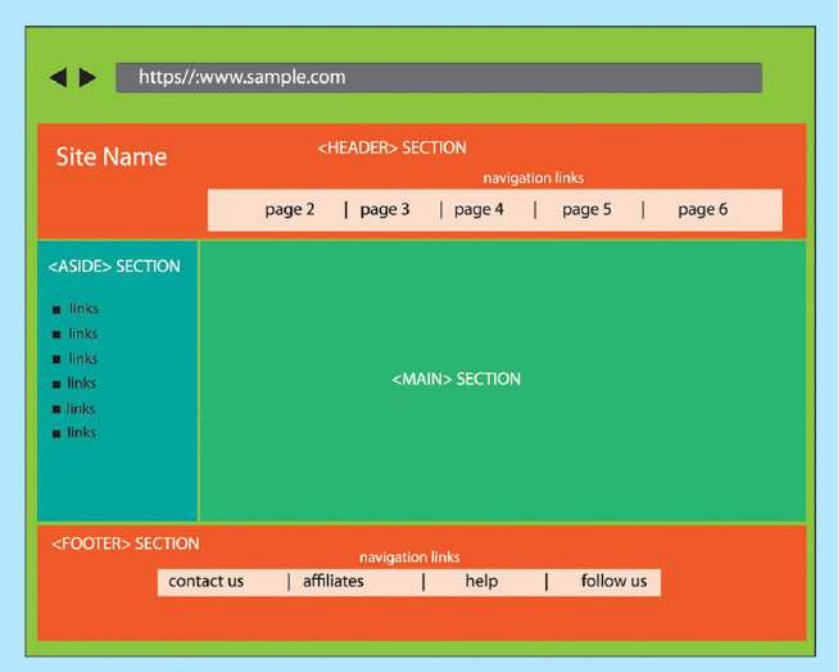

fig1: Semantic element Visualization 1

The SEO Impact of Semantic Markup

Search engines have evolved from simple keyword matchers to sophisticated content interpreters. When you use semantic HTML, you're essentially creating a detailed map for the search engine:

- **Content Hierarchy Recognition**: differentiating between you main content <main>, you supplementary content <aside>, and the navigation content <nav>.
- **Enhanced Content Understanding**: Elements like <article> signal to the browser that content is self-contained and potentially valuable for indexing. Which we talked about in SEO research.

- **Rich Snippet Opportunities**: it creates snippets that the SEO can link to or find corelated to the search prompt.

Accessibility Through Semantic Structure

Semantic elements like and transform accessibility by providing screen readers with crucial vocal cues; while adds info about nuanced emphasis, delivers pronounced vocal stress for urgent or critical information.

Combined they ensure content is not just heard but understood with its intended meaning and importance.

The Power of Metadata & Social Sharing

What?

It's much simpler than it sounds like. When you send a YouTube video, LinkedIn account, or a tweet through WhatsApp, you see a little preview card; that little picture, title, URL, etc.

Open Graph (OG) Protocol: The Universal Standard

Originally made by Facebook, however now used by most platforms. As I said, you basically tell the platform what to display. It's written in the head element of course, and let me show a code example for the syntax:

```
<head>
    <!-- Primary Meta Tags -->
    <title>3ayez anam</title>
    <meta name="description" content="A description for search results.">

    <!-- Open Graph Meta Tags -->
    <meta property="og:title" content="Slept a sum of 3 hours in the past 2 days" />
    <meta property="og:description" content="an engaging description like: insomnia
medication" />
</head>
```

Essential tags to note:

- og:Title
- og:Description
- og:Image
- og:Image:width
- og:Image:height
- og:URL
- og:Type

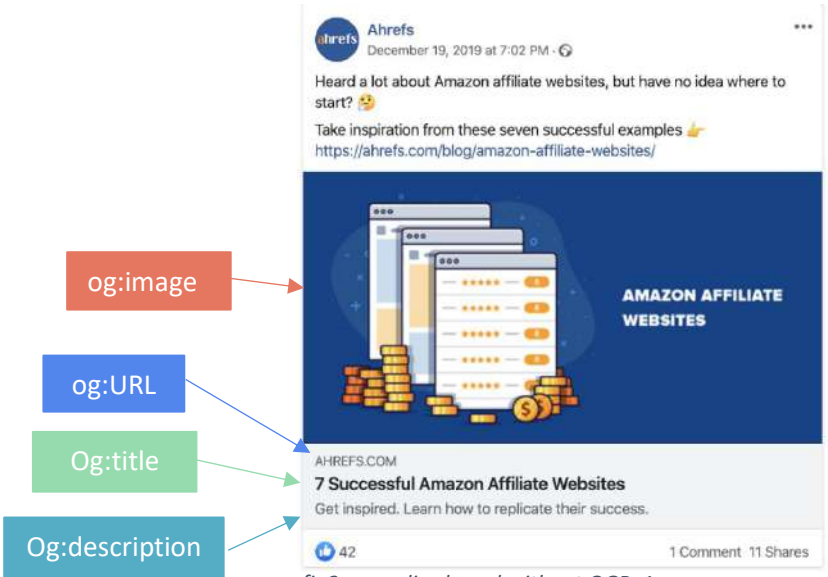

fig2: a media shared without OGP 1

Twitter cards: twitter tailored

Similar to OGP, however more tailored to twitter. More tailored how exactly? Well, let me tell you.

Twitter offers specific card formats that match how content is consumed on the platform:

- **summary_large_image**: Optimized for Twitter's timeline with large, attention-grabbing images
- **summary**: Traditional small-image format for text-heavy content
- **app**: Designed specifically for mobile app installs (Twitter's strong mobile user base)
- **player**: For embedding video/audio content that plays directly in the timeline

fig3: twitter cards 1

The rest of the **code** you basically switch the “og:” for “twitter:”. But here is an example:

```
<meta name="twitter:card" content="summary_large_image" />
<meta name="twitter:title" content="World's 50 best foods" />
<meta name="twitter:description" content="The world is full of good food. But
what are the 50 best dishes -- those so delicious you should factor them into
your travel plans?" />
<meta name="twitter:image" content="https://www.istockphoto.com/photos/big-
meal" />
<meta name="twitter:site" content="@CNNTravel" />
```

Note: when twitter fails to load the link using the twitter card code, it switches to the og code. So, make sure to add both so that if you’re 100% sure it’ll only be shared on twitter, which is impossible, also add the og attributes.

The Logic of File Paths

In the task we were asked to link a page in the same file, which means we needed a little knowledge about paths and the file system. So let's discuss it in more details

What is a file system?

A file system is the method and data structure an operating system uses to manage, organize, and store files on a storage device like a hard drive or SSD. There is the basic windows system (NTFS) that you're mostly familiar with, the linux system (ext4), and the mac system (APFS) for you rich folks. Here is a quick comparison:

Feature	Windows (NTFS)	Linux (ext4)	macOS (APFS)
Primary Filesystem	NTFS	ext4	APFS
Path Separator	Backslash (\)	Forward Slash (/)	Forward Slash (/)
Drive Letters	Yes (e.g., C:\)	No	No
Case Sensitivity	Case-insensitive	Case-sensitive	Case-insensitive
Permissions Model	ACLs	POSIX	POSIX + ACLs
Key Technology	Journaling	Journaling, Extents	Copy-on-Write, Snapshots

Table1: Comparing the 3 main file system 1

Keep in mind most of this info is extra. Now let's conceptualize the difference between Relative and absolute path.

Relative vs absolute

- **Absolute Path:** The full, complete address from the very top of the file system (root/partition) to your specific file.
- **Relative Path:** A shorter, contextual address that starts from where you currently are (i.e., the location of the HTML file you're working on).

```
<a href="E:\PROJECTS\Tasks\Task_01\absolute.html">Absolute path</a>
<a href = "Task_01\relative.html">Relative path</a>
```

fig4: Absolute path example 1

Tip: On VScode, you can right click the intended file and copy either its relative, or absolute path.

“./” vs “../”

- “./” means “hey, look here in this folder”, and the dot can be omitted on most applications, command line prompts and IDEs.
- “../” means “well, it’s outside of here (parent folder) by one level”. And yes, it can be repeated if you want to go up multiple levels. Look at this example:

```
<link rel="stylesheet" href="../../css/style.css">
<img src="../../../images/photo.jpg">
```

- And yes you can omit the “/” if it’s the last execution. And you can put it in quotations “`../photo.png`”. all under one condition, you’re a *psychopath* 😊.

Root directory

Unfortunately, a lost concept in windows as windows is a Drive-based system. The root directory Is , as the name implies, the root of all the branches (subfolders) under it.

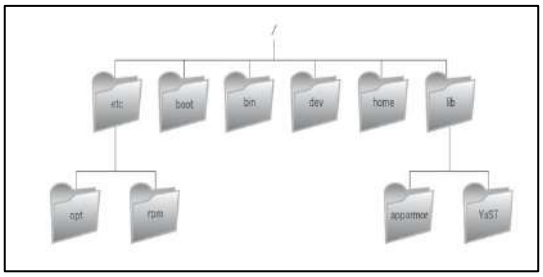

fig5: linux filesystem 1

The root overs a better and easier structure when coding or launching system calls in the shell.

Web Image Format

It can be summed up in a yt short, and you can make your pHD on it. So we'll meet in the middle. But before we can compare the different image types. There are key concepts to discuss first:

Raster vs Vector

Raster: Your normal image that you see in most media, Composed of a fixed grid of tiny, colored squares (pixels). When scaled it gets "pixelated"

Vector: Composed of mathematical paths defined by points, lines, and curves. Think of apps icons. When scaled up, it retains its look and quality.

fig6: raster vs vector visualization 1

Lossy vs non-lossy compression

Lossy Compression → Prioritizes drastic file size reduction by permanently discarding data considered less important to human perception. Which makes it more or less ideal for web use.

Lossless Compression → Reduces file size by eliminating statistical redundancy without sacrificing any original image data, allowing for a perfect reconstruction of the original file.

It is essential for logos, maps, or any situation where you need pixel perfect quality. The drawback is that it achieves more modest size reductions compared to lossy methods, resulting in significantly larger files that are less practical for web use

Alpha color channels

An alpha channel is a dedicated component of an image file that stores transparency information for each pixel. Think of it as a separate grayscale map that works alongside the standard color channels (Red, Green, and Blue).

So, when coding you can identify the color by typing in 4 slots instead of the normal three.

```
 color: rgb(0, 255, 0);         /* Solid green */

/* RGBA with transparency */
    color: rgba(0, 255, 0, 0.5);  /* Semi-transparent green */
```

Comparison between the main Image file types

Feature	JPEG / JPG	PNG	SVG	WebP
Raster or Vector	Raster	Raster	Vector	Raster
Compression (Lossy/Lossless)	Lossy - Quality decreases with each save.	Lossless - No quality loss on compression.	Lossless (as code) - No quality loss when scaling.	Both - Supports both lossy and lossless modes.
Alpha (RGBA) Support	No - No transparency. Solid background only.	Yes - Full alpha channel transparency (soft edges).	Yes - Full transparency control per shape.	Yes - Full alpha channel transparency.
Animation Support	No	Limited (APNG, not widely supported)	Yes - Via SMIL or CSS/JS animation.	Yes - Supports animated WebP.
Best Utility & Use Cases	Photographs and complex images with many colors where small file size is critical.	Logos, icons, graphics with sharp edges, text, or when transparency is needed.	Icons, logos, illustrations, graphs, and interactive graphics that need to scale perfectly on any screen.	General-purpose replacement for both JPEG (photos) and PNG (transparency) to improve performance.

Now, let’s discuss some questions that you should be asking:

What’s the difference between JPEG and JPG?

They’re basically the same, just a file naming limitation, by windows of course, sparked the JPG. As windows, and I am not joking, couldn’t use more than 3 letters as a file extension. Full name is “Joint Photographic Experts Group”.

How does SVG work?

As the name suggests, and we discussed, it’s a victor technology; “Scalable Vector Graphics”. But how do they work exactly. You code them, and it’s kinda simple, not easy through.

This is the most important and unique aspect of SVG. An SVG file is not binary data; it's actually a text file written in XML (a markup language similar to HTML).

Let's look at a simple example. This code creates a red circle with a black outline:

```
<svg width="100" height="100" xmlns="http://www.w3.org/2000/svg">
  <circle cx="50" cy="50" r="40" stroke="black" stroke-width="3" fill="red" />
</svg>
```

Let's break down the code:

```
<svg …>
```

This is the root element, defining the canvas (100px by 100px).

```
xmlns="h…
```

Basically lets the browser know that this is a SVG through a namespace

```
<circle cx="50" cy="50" r="40" stroke="black" stroke-width="3" fill="red" />
```

This defines a circle with*x* and *y* coordinates of the center of the circle at 50 each. r="40" defines The radius of the circle. Stroke and stroke-width is for the outline, and fill is, you guessed it, the color of the filling.

Because it's just code, you can open an SVG file in a text editor and read or edit it.

See all of that, it's **legacy code** now. SVG are embedded in HTML5 now and you can easily make them in your files.

And if you're frustrated that I've written all of that for nothing, thank allah because there was about a page and some more talking about the namespaces and their history.

A real world example of an SVG code using html is in fig 7

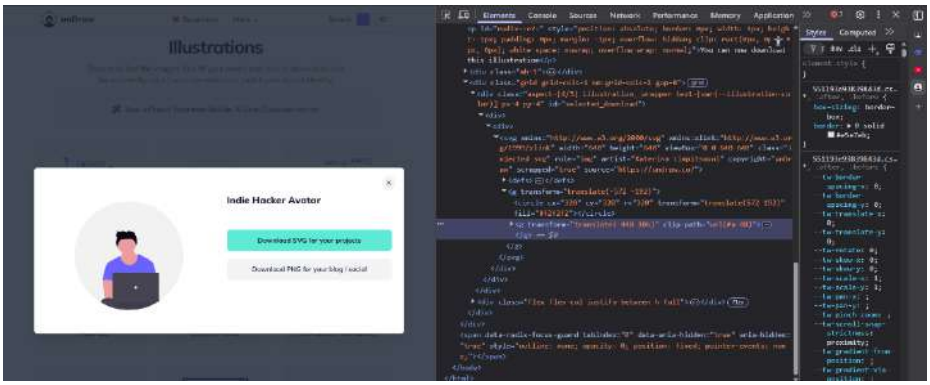

fig7: actual SVG on a design website 1

The code is a little bit more complex than that, as each element has a code like that. For example, the code highlighted in the picture is for the grey circle only.

All of that, and we're only scratching the surface of the image file types. There are important topics like color depth, DPI, ect. That we're yet to talk about. But as a front-end developer, that's a decent headstart.

HTML VALIDATION

Browsers are designed to be forgiving; they will often render a webpage even if the HTML code contains errors. They use a process called "tag soup" parsing to guess the developer's intent and fix mistakes on the fly. However, relying on this is a dangerous practice.

W3C Markup Validation Service

fig8: The W3C validation service 1

The W3C Markup Validation Service is a free online tool created by the World Wide Web Consortium (W3C)—the main international standards organization for the web. Mainly validating HTML and XML files.

How can you use it to find invisible errors?

You can use the validator by providing a web page URI, uploading an HTML file (as in figure 8), or pasting your source code directly. The service then compares your code against the official web standards. It flags errors and warnings that are not necessarily visible in the browser, such as:

- Missing or incorrect closing tags.
- Missing required attributes (e.g., the alt attribute in an <img> tag).
- Incorrect element nesting.
- Deprecated tags or attributes.

These "invisible errors" might not break the page's visual layout immediately but can lead to unpredictable behaviour, accessibility issues, and search engine optimization (SEO) problems.

Why use a validator when I have my beloved VScode with all of its capabilities?

VS Code is a text editor, not a real HTML validator. While it has excellent syntax highlighting and can warn of obvious typos, it cannot catch all logical and standard-compliance errors. The W3C Validator is a dedicated tool that will find issues VS Code ignores, such as:

- **Incorrect element nesting** (e.g., putting a inside a <p> tag).
- **Missing or invalid attributes** for specific elements.
- **Subtle accessibility issues** related to document structure.
- **Deprecated code** that may break in future browsers.

Using the W3C validator is like having a senior developer review your code for standards compliance, catching problems before they cause issues.

Now let's experiment:

VScode gave me the green flag for this code:

```
1   <!DOCTYPE html>
2   <html>
3   <head>
4       <title>Test Page</title>
5   </head>
6   <body>
7       <p>This is a paragraph.
8           <div>This is a div inside a paragraph.</div>
9       </p>
10  </body>
11  </html>
```

fig9: code example for bad logical code 1

Not a single squiggly underline. Let's see wheat W3C validator has to say about it shall we?

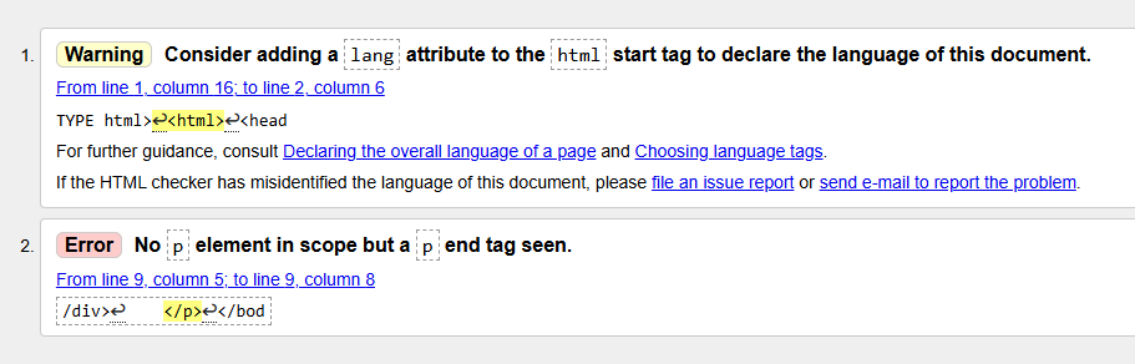

fig10: W3C response for bad logical code 1

As you can see, the validator didn't take so kindly to our code. And That's exactly why most companies require a fully validated HTML code.

Note: There is a different type of HTML validation, called HTML **form** validation. And it's concerned with the user input as the name suggests. It's like try catch in programming languages. As seen in this MDN article.

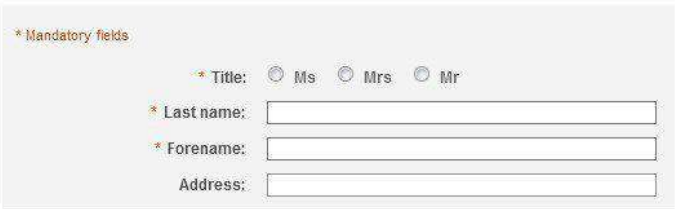

fig11: Form validation example

References

I. Semantic elements: https://www.searchenginejournal.com/why-you-should-consider-semantic-html-for-seo/506384/, https://www.youtube.com/watch?v=RF_5qiMwNX4
II. OGP: https://www.youtube.com/watch?v=RW5HCOMbvuQ, https://ogp.me/ , https://www.freecodecamp.org/news/what-is-open-graph-and-how-can-i-use-it-for-my-website/,
III. Twitter cards: https://www.youtube.com/watch?v=QJxzgjo47q8, https://www.youtube.com/watch?v=QJxzgjo47q8,
IV. File system: https://8grams.medium.com/linux-101-filesystem-2790c8ce0721, https://support.apple.com/en-eg/guide/disk-utility/dsku19ed921c/mac
V. Root: https://unix.stackexchange.com/questions/615887/linux-root-directory-and-partition,
VI. Web Images:https://developer.mozilla.org/enUS/docs/Web/Media/Guides/Formats/Image_types , https://www.youtube.com/watch?v=WblPwVq9KnU, https://vector-conversions.com/vectorizing/raster_vs_vector.html,
VII. SVG: https://developer.mozilla.org/en-US/docs/Web/SVG/Guides/Namespaces_crash_course,
VIII. HTML validation: https://www.sistrix.com/ask-sistrix/onpage-optimisation/what-is-html-validation, https://www.youtube.com/watch?v=qrU3ghYJjEw, https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Forms/Form_validation,