Lecture 8: Introduction to CSS, HTML Forms, and Node Templating

CS-546 – WEB PROGRAMMING

Express and Static Files

What are static assets?

Lately, we have seen servers that were 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.

- A middleware is a function that runs and modifies a request before it hits a route.

You can see the usage of the static middleware in **app.js** for the lecture 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.

CSS

What is CSS?

CSS is the language that we **style** HTML documents with.

- Cascading
- Style
- Sheets

CSS allows you to define rule-sets, which are selectors (identifiers that target HTML Elements) and rules (rules that define visual properties)

You can use CSS to describe how a document is presented in different contexts

- In a browser
- On a projector
- When printed

It allows you to take your documents and make them look like fully designed pages.

Adding CSS

It is very easy to add a CSS stylesheet to your HTML document.

In the **head** element, you add a *link* element like such:

- `<link rel="stylesheet" type="text/css" href=”/path/to/style.css">`

How does CSS work?

CSS is a simple language based on only three pieces of data:

- A selector, which is a pattern that will match elements
- A declaration, which has:
  - A **property**, which defines what property you will update
  - A **value**, which defines what you want to do to that property.

A set of selectors and a set of declarations makes a *rule-set*

Each rule-set can have multiple selectors, and multiple declarations.

CSS Syntax / Example rule-sets

```
h2 {
  text-align: center;
}

p.bio, .about-me .career-info {
  font-size: 16pt;
  border-bottom: 1px solid #333;
}
```

The first rule-set will target **all** h2 elements in the document, and center all the text contained inside the element.

The second rule-set will target all **p** tags that have a class of **bio**, as well as all elements with a class of **career-info** that are contained inside an element that has a class of **about-me**; there can be any number of elements and nested layers of elements between **about-me** and **career-info**. Any matching elements will have a bottom-border that is 1 pixel in size, grey colored, and a solid line; they will also have their font set to be 16pt.

What can we change with CSS?

The way text appears

- Color
- Size
- Line Height

The way our elements appear

- Shape and size
- Background color
- How other elements are positioned inside of an element

The position of elements on the page

- How the elements are spaced out
- Where they appear on the page

So much more.

Design in this course

Fortunately, the only type of design we care in this course is designing in an accessible manner

Since this is not a web-design course, your design skills will not be taken into account.

You will be graded on your understanding of CSS and how to apply designs.

Basic CSS Selectors and Units

Selectors

CSS	Name	Selects
`*`	Universal	Any and every element
`div`	Element	All div elements
`.foo { }`	Class	All elements with a class of *foo*
`#bar { }`	ID	The single element with the id of *bar*
`#bar .foo { }`	Descendant	All *.foo* that are contained inside *#bar*
`.parent > .child`	Child	All .child directly inside of .parent
`.post + .divider`	Adjacent Sibling	All .divider that are directly after .post in their parent.
`.post ~ .subtext`	General Sibling	All .subtext that come anywhere after .post in their parent.
`.post:pseudoclass`	Pseudo-Class	All .post that have a particular pseudoclass.
`[role='navigation']`	Attribute	All elements that have an attribute named 'role' with the value of 'navigation'. You can do this for any attribute and value.

Pseudo Classes (All Types)

Pseudo Class	Selects
`:first-child`	The first child of a parent
`:last-child`	The last child of a parent
`:first-of-type`	The first element of a type, inside of a parent; does not accept a class or id.
`:last-of-type`	The last element of a type, inside of a parent; does not accept a class or id.
`:nth-child(an+b)`	Selects all .child directly inside of .parent The formula (*an+b*) describes which elements are targeted; elements start at index 0. (*2n+1*) would select elements at index 1, 3, 5, 7 while (*3n*) matches at index 0, 3, etc.
`:nth-last-child`	Same as :nth-child, except starts from the last element
`:nth-of-type`	As as :nth-child, except works with element types
`:empty`	An element that has no content, including whitespace; can have comments.
`:target`	The element matching the target of your current hash.

Size Units

Unit	Name	Description
px	Pixel	Size of a pixel on the screen
mm	Millimeter	Size of a millimeter
cm	Centimeter	Size of a centimeter
in	Inch	Size of 1 Inch; generally, 96 pixels per inch
pt	Point	Size of 1/72 inch
pc	Pica	Size of 12 Points
em	em	Calculates the size based on the elements font size, or parent's font size. Literally, relative to the "M" in a font at a size.
%	Percent	Calculates the percentage of a size relative to a property on the parent element.
rem	Rem’s	Same as em, but based on the root element. Allows for very responsive layouts just by updating the root element.

Color Units

Color	Unit Name	Description
`rgb(255, 255, 255)`	RGB	Allows you to describe colors as how red, green, and blue they are from 0 to 255 each. Allows for 16,581,375 colors.
`rgba(0, 192, 45, .5)`	RGB Alpha	Same as hex, where the last decimal is transparency from 0 (clear) to 1.0 (solid)
`#A90BEF`	Hex	A hex triplet; has 2 hex digits representing each color (Red, Green, Blue). Describes your color from Allows for 16,777,216
`hsl(0, 20%, 50%)`	Hue, Saturation, Lightness	Takes the hue from 0 to 360; 0/360 are red, 120 green, 240 blue. Saturation determines what percent of that color you use. Lightness is a mask over saturation, adding a white layer. 50% is the normal value.
`hsla(0, 20%, 50%, .5)`	Hue, Saturation, Lightness, Alpha	Same as above, where the last value is transparency.

Basic and Common Rules

Text Rules

Rule Name	Example	Outcome for selected element
font-family	`font-family: "Open Sans", "Helvetica", sans-serif;`	If the user has Open Sans font, will use open sans; else Helvetica; else sans-serif (generic font).
font-size	`font-size: 18pt;`	Sets font size at 18pt (about ¼ of an inch)
line-height	`line-height: 3;`	Will make each line have a height of 3x font size.
text-decoration	`text-decoration: underline;`	Text will be underlined.
font-weight	`font-weight: 700;`	Font will be bold; norm is 300.
text-align	`text-align: center;`	Text will be centered inside parent element.
font-variant	`font-variant: small-caps;`	Makes an element use small capitalized letters for lowercase.
font-style	`font-style: italic;`	Makes text italicized.
font	`font: 2em "Open Sans", sans-serif;`	Sets font at 2em large, Open Sans (fallback to sans-serif)

Color and Background

Rule Name	Example	Outcome for selected element
color	`color: #690;`	Changes text color to a green.
background-color	`background-color: #000;`	Sets background color to black.
background-image	`background-image: url(gradient.png);`	Uses gradient.png as background for element.
background-attachment	`background-attachment: fixed;`	Makes the background image fixed in browser.
background-clip	`background-clip: border-box;`	Sets the background to fill entire box model.
background-position	`background-position: left center;`	Positions your background anchored to the left horizontally and center vertically.
background-repeat	`background-repeat: repeat-x;`	Sets background to only repeat horizontally.
background-size	`background-size: contain;`	Scales the background image to fit length or width and letterboxes.
background	`url(gradient.png) no-repeat;`	Shorthand for multiple properties.

Setting a border

Example	Outcome for selected element
`border-left: 1px solid #999;`	Sets a 1px wide solid border on the left; grey.
`border-right: 5px dashed #558abb;`	Sets a 5px wide dashed border on the right; light blue.
`border-top: 1px solid #999;`	Sets a 1px wide solid border on the top of the element; grey.
`border-bottom: 1px solid #999;`	Sets a 1px wide solid border on the bottom of the element; grey.
`border: 2px dotted #558abb;`	Sets a 2px wide dotted border on all sides; light blue.

Putting it all together.

Example	Outcome for selected element
`box-size: border-box;`	Makes your element include padding and the border in width.
`padding: 5px;`	Adds 5 pixels of padding on the element.
`margin: 0px auto;`	No margins on top or bottom; left and right will automatically be calculated, possible centering element in parent.
`width: 75%;`	Sets width of element to be that of 75% of the parent's width.
`height: auto;`	Height will automatically be determined by content;
`max-width: 500px;`	Element will not exceed 500px wide.
`position: relative;`	Element will be moved relative to static position.
`top: 10px;`	Element will be 10px below where it was originally meant to be

Form Elements

What is a form?

"HTML Forms are one of the main points of interaction between a user and a web site or application. They allow users to send data to the web site. Most of the time that data is sent to the web server, but the web page can also intercept it to use it on its own."

- Mozilla Developer Network
- https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Forms/My_first_HTML_form

A form is a collection of fields used to signify user input.

Forms allow for interactive experiences

Forms allow users to interact with the server

- Can submit data to server through forms

You can have multiple forms per page

What is a form made of?

Forms are made of a form element

- Has an `action` attribute that signifies where the form will be submitted to
- Has an `method` attribute that signifies how the form will be submitted (GET, POST, etc)
- Can have a name that helps identify the form through the DOM API

Forms have one or more inputs that allow users to populate data

- This data will be submitted to the server
- This data can be accessed through the DOM API

```
<form method=”POST" action=”/questions/2/">
    <input type="text" name=”title" />
    <input type="submit" value="Search" />
</form>
```

What is an input?

An input is an individual piece of data.

- Inputs have names
- Inputs can have id's
- Inputs have values
- Some types of input can have placeholders

The server receives data in a key-value format, where the key is the name and the value is the value.

Names are unique on a form-level basis

Ids are unique on a document level basis

Many types of input use the input tag

```
<label> Result Name:

<input type="text" placeholder="result
name" name="result_name" /></label><br />

<label> First Value:

<input type="number" name="first_value"
/></label><br />

<label> Second Value:

<input type="number" name="second_value"
/></label><br />

<input type="submit" value="Multiply Them"
/>
```

What types of input can I have?

There are many types of input, most of them revolving around some sort of text.

You will most commonly use:

- text
- email
- number
- radio
- file
- checkbox
  - select

By going to https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input you can see a list of all input types and their properties.

Text based inputs

There are many types of input, most of them revolving around some sort of text.

- Text can be numerical
- Text can be string based

When submitting through an HTML form, this text is submitted as a string, no matter what the data type is.

```
<label> Result Name:

<input type="text" placeholder="result
name" name="result_name" /></label><br />

<label> First Value:

<input type="number" name="first_value"
/></label>
```

Single-Select and Radio Inputs

Some types of inputs internally store strings, but give a limited number of options.

In the `select` example, the form will send information with `query` as the **key** and whichever option you select as it's **value**. It uses the option's **value** field to determine what query will get, when that option is selected

The `input type="radio"` example, as far as the form is concerned, puts the same data as the `select` example. The value of whichever radio input is checked (which changes when the user selects a new value) will become the value of `query`.

```
<select name="query">
  <option value="Austin">Austin</option>
  <option value="Helvetica">Font</option>
</select>
```

```
<input type="radio" name="query" value="Austin">Austin <br />
<input type="radio" name="query" value="Helvetica" checked>A Font
```

Multi-Select and Checkbox Inputs

If you add the `multiple` attribute to a select element, you will be able to select multiple values.

In order to use a series of checkboxes, you would use the `checkbox` input type.

Express will automatically parse these as arrays for you, but all their values will be strings

```
<select name="query" multiple>

<option
value="Austin">Austin</option>

<option
value="Helvetica">Font</option>

</select>



<input type="checkbox"
name="query" value="Austin"
/>Austin <br />

<input type="radio" name="query"
value="Helvetica" checked />A Font
```

Hidden Inputs

We can make inputs with a type of *hidden* that will not show up on the user's screen.

Often, we use this to set data about a request that we don't want a user to update.

We can use this to fake being able to PUT and DELETE via a browser without AJAX.

Express will detect these inputs and serialize them into the request body.

```
<input type="hidden"
name="method" value="PUT" />
```

Benefits of using correct input type

Modern browsers are able to use the input type in order to help users with their input

- email types of input will automatically show up as invalid with non input strings
- number types of input will only allow users to input numbers
    - Smart mobile devices will only show numbers

Screen readers and other devices will be able to more intelligently comprehend your website.

The proper type will not be picked up by the server when submitting this data, unless submitting JSON over AJAX

Labeling your inputs

Labels describe your input, both visually and accessibility-wise.

You can define which input you are labeling in two ways

- Wrapping the input in a label
- Using the `for` attribute, where it has a value equal to the `id` of the input you are labeling.

Massively improves accessibility by allowing screen readers and other such things to describe the input in a non-visual way.

You must use labels for your inputs for accessibility reasons.

```
<label><input type="checkbox"
name="query" value="Helvetica"
checked>A Font</label>
```

```
<input type="checkbox"
name="query" value="single-
origin coffee" checked
id="checbox-coffee">
```

```
<label for="checkbox-
coffee">Coffee</label>
```

Submitting a form: client side

You submit your form to a file or location on your server.

- You submit using a `button` or `input` with `type="submit"`
- Can submit to same location that your form is located on

Data you fill out on that form will be submitted

- If you have multiple forms, only data from the form that triggered the submission will be sent

You can also use JavaScript to intercept the submission of a form and perform actions on the page, instead of sending the data to the server.

```
<form method="GET"
action=”/search">

    <input type="text"
name="query" />

    <input type="submit"
value="Search" />

</form>
```

Issues submitting forms

When submitting via a normal-browser form, most browsers will only send via GET or POST

- We can get around this with hidden inputs and middleware to change how to interpret the request
- Using AJAX requests we can often set the proper http verb

Normal forms do not carry the type of input with them

- Form variables are sent as text via:
  - application/x-www-form-urlencoded
  - multipart/form-data
  - text/plain
- Your server will therefore get every data type as a string, even if it’s a number or boolean
- We can get around this by submitting JSON encoded data, which will allow us to get JS primitives of the proper type

Accepting Form Input

Accepting Form Data

Generally, forms are submitted via the browser natively making HTTP requests or setting up an AJAX request.

- By default, form data is sent either via a Querystring (when we set the method to GET) or the request body (POST)
- PUT and DELETE data have to be faked when the browser is natively making that request.

We can see all of our form data being used all over *routes/calculator.js*; the *request.body* will have a formatted object describing any submitted data.

Faking PUT and Delete

I have provided a simple middleware that will look for hidden inputs and allow normal HTML forms to be routed into PUT and DELETE routes. While we will not be covering middlewares until later on, however you may find it interesting to look.

Error checking

While we can check for errors on the client side, we must also check for errors on the server side

- You can submit HTTP requests without a browser – this makes client site validation easy to circumvent.

There are a number of ways to handle errors. The simplest way is to send a failing status code and a message to the user. We can also redirect to an error page.

We can also take the error and pass it to the same page as the form, as well as passing back the data they already filled out.

Common errors:

- Assuming input will be a type (ie: number) when it is a string
    - Try to parse it into the proper type first!
- Assuming input exists at all!
- Assuming input is “within bounds” (ie: expected a number between 1-10)

Explaining Errors to the User

For now, we will be passing the error to the same page that the user filled out the form in.

Later on, we will configure middleware that will take input and automatically convert form data to normal objects; we will also make middlewares to determine whether or not these objects are sent in the proper bounds.

Even further on, we will dynamically inform the user about errors with AJAX calls.

Templating in Node

What is templating?

Web applications would be very limited and boring if you could only view static files that never changed unless the files were updated.

Templating is the process of passing data into a file that describes what HTML should be generated based on the data you provide.

- To display a blog post, rather than saving one file per blog post, you would query your database for the blog post and pass it to a template file, which will generate the HTML from the template and

Web development would be **very** painful if you had to manually concatenate HTML strings in the middle of all your controller logic.

- See: PHP without an MVC framework

With that in mind, Express gives an easy way to template: template engines.

What is a template engine?

Template engines are modules that can be hooked into Express. These modules will then allow you to take static files that have special markup and replace said markup with data that you provide.

In order to have express respond with a rendered template, you will use the *response.render* method.

What is our template engine?

We will be using the *express-handlebars* module as our templating engine

- https://github.com/ericf/express-handlebars
- http://handlebarsjs.com/

The express-handlebars engine will allow us to perform many common template tasks:

- Print variables
- Iterate over arrays
- Create a single layout that all views inherit from
- Add partial views