HTML Canvas

Mari Good

CS 233JS

Topics

- MemeCreator application
  - Webpack review
  - CSS 3 Flexbox
  - HTML5 Canvas
  - Asynchronous operations
  - Using the FileReader to read files from the client machine

The MemeCreator App

- This lab involves just one app, the MemeCreator. I'll write a version of the app with you and then ask you to add some features to the app independently.
- Let me start by showing you what the app does on my machine.

Introducing Webpack

- This lab, just like the previous lab, uses webpack, a javascript module bundler.
  - In 133JS, if we had more than one external js file, we got them to work together by adding a script element for each file in the html file. That requires the front end (designer) to have to know the details of how your js code is structured. That's a bad idea.
  - ES6 introduced language support for breaking js code up into modules. Professional JS developers put each "component" in a separate file and make it available to other js files by exporting some or all of the contents of the file. Another js file can import the contents of the module by adding an import statement, must like you would do in C#.
  - Webpack actually combines all of the js modules into 1 file – and if you've used other modules like jQuery or axios – it bundles all of the js code based on the dependencies of the individual modules.

The Starting Files

- Get the starting files from the github repo. Notice that I've given you
  - The css for the application. You won't change this file.
  - The html for the application. You'll change this file only minimally. I'll tell you about that.
  - 2 JavaScript files. You'll add to both of these files with me as we build the application.
  - 2 configuration files
    - package.json – documents all kinds of things about your project including the node modules you're using and the dependencies of each of these as well as the "macros" or scripts you're using during the development process
    - webpack.config.js – documents how you're using webpack to bundle your application. This file is read by webpack so you don't have to type all kinds of things on the command line when you're building your application

Setting up Your Dev Environment

- We're going to start by setting up our development environment. You did this in the last lab so all of this should be a review.
- I've given you the package.json file as well as the webpack.config.js file.
  - I'll talk, in general, about what each file does for you but I won't go over any details.
  - Any time you need to build an application that uses npm and webpack, you can start with these 2 files. I'll talk about that process when we're finished with this lab. You'll need to do it when you create your dev environment for your project in week 7.

Package.json

- Let’s start by installing the node modules we’ll be using
  - npm install
  - This will read package.json and install all of the node modules listed there. If you look at the devDependencies section, you’ll see babel (we used this in our last lab), a bunch of “loaders” and webpack.

Using Webpack

- Let’s use webpack to “bundle” our application
  - npm run webpack
  - webpack is an entry in the scripts section of package.json
  - The webpack command without any command line arguments will look for the webpack.config.js file for configuration information. We’ll look at some of this in a minute.
  - It creates a dist folder for us. The “transpiled” version of our code will be bundled with resources like css and bootstrap and will be in this folder. The dist folder now contains
    - index.html – is a copy of index.html from the src folder with a script tag that refers to the bundled code at the bottom of the body.
    - images – is a copy of the images folder from src
    - memes.bundle.js – the transpiled and bundled code
    - memes.bundle.js.map – maps the bundled code to our original code for debugging purposes

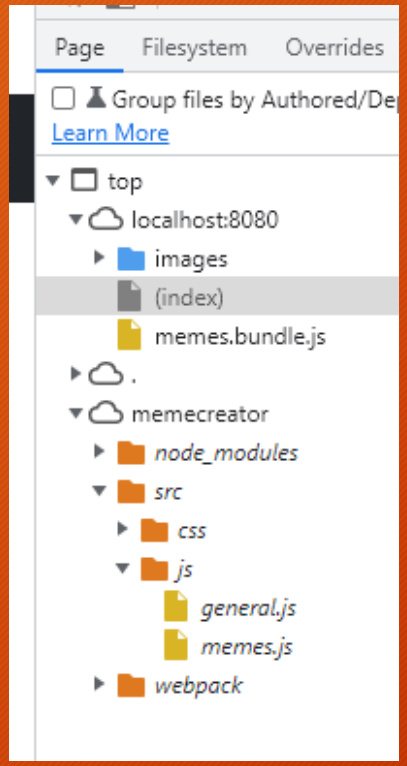
Webpack Development Server

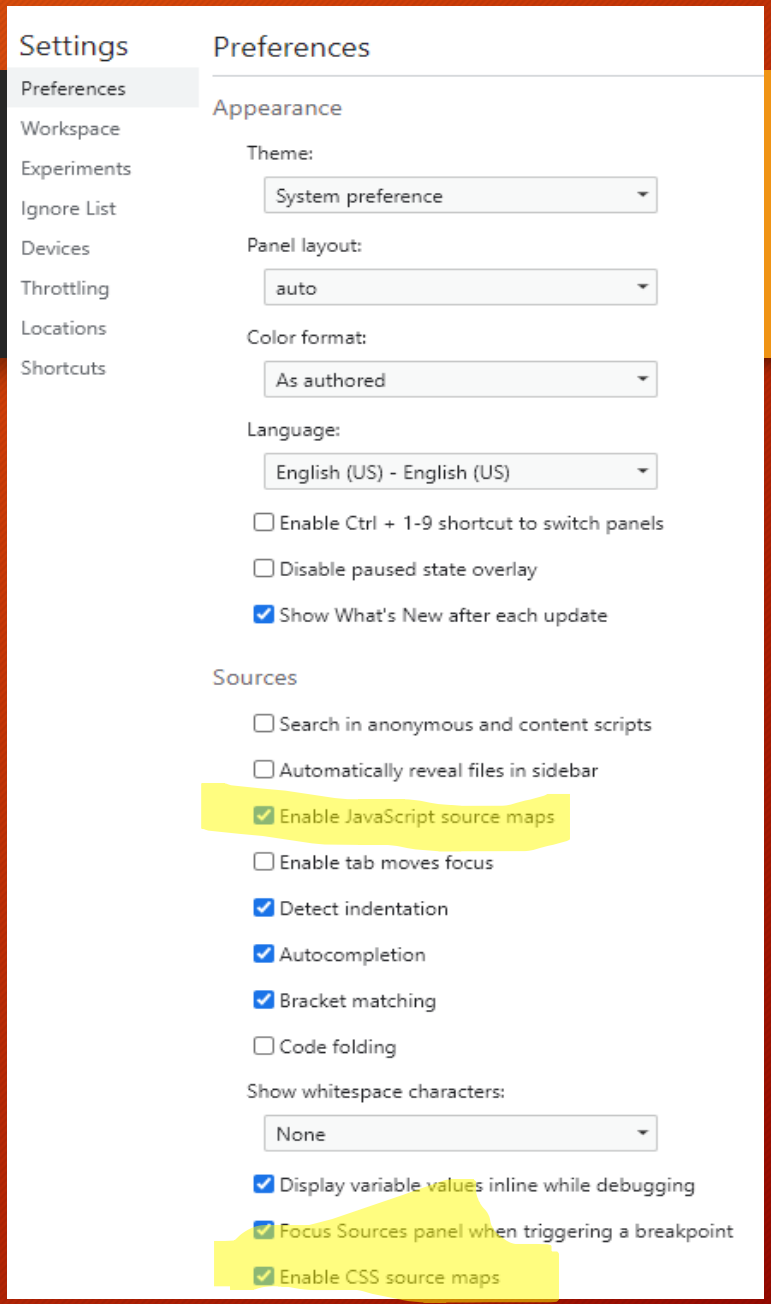
- Let’s run our application
  - npm run watch
  - watch is an entry in the scripts section of package.json
  - The webpack development server is a simple web server that is tightly integrated with webpack. Every time you save a file that’s part of your application, webpack will “transpile” and bundle the application and reload the app in its web server.
  - Open the developer tools and look at the console. There should be several messages (from the general.js and memes.js files) in the console.
  - Add a console.log statement to the top of one of the js files (or change one of the existing statements) and save the file. You should see the page reload and the comment should display in the console.

Debugging

- While we're at it let's make sure source maps work and you can debug.
- This process is a little different in every browser. I'll demonstrate in Chrome. The next slide has a couple of screenshots of what my developer tools look like in Chrome.
  - Open the developer tools settings and enable javascript and css source maps. You're running the transpiled and bundled version of your application and you'll want to debug by working in your original files. Source maps allow the browser to map the contents of the running file to your original files.
    - When you close the developer tools settings and look at the sources tab, you should see cloud icon that refers to the running application (localhost:8080) as well as your source files (memecreator). Make sure you can find the js source files you wrote.

Chrome Debugging Images





Introduce an Error

- Make an error in one of those js files.
  - I'll capitalize the word Console in memes.js.
  - Save your files. OH, BY THE WAY, I turned OFF automatic saving in VS Code, so that the webpack dev server is not constantly trying to transpile and bundle your work.
  - You should see an error message in the console and the error message should reference a line of code in the file you just edited NOT a line of code in the bundled file.
- Being able to use the debugger in the browser is VERY important in the process of writing and debugging a client-side application. Make sure all of this works for you or see me during office hours before you continue if you're having trouble.

The Webpack Configuration File

- I tried to create a webpack.config.js file that would work for our labs in general but wouldn't be overly complicated. The configuration is still pretty complicated, but I want to point out a couple of things
  - Line 7 – this is a development configuration. Later in the term I'll add production configuration.
  - Lines 8 - 10 – lists the js files that are being transpiled
  - Lines 11 - 16 – output is written to the dist folder
  - Lines 18 – 20 – development server is serving from the dist folder
  - Line 21 – create source maps for debugging
  - Lines 22 – 50 – loaders to use with different kinds of files based on the file extension
  - Lines 52 – 57 – use html webpack plugin to create the html file dynamically
  - Lines 58 – 65 – use copy plugin to move the images from src to dist dynamically

Let’s Get Started

- Now that we’ve got our development environment set up, we can finally begin to write some code.
- The application introduces 1 significant CSS concept and several programming concepts.
  - CSS flexbox rather than bootstrap grids for layout
  - Drawing to the page using HTML canvas
  - Asynchronous operations
  - Using the FileReader to read a file from the client machine
- I’ll talk a little bit about each of those things as we build the application from my comments in the source code.

CSS Flexbox

- Look at index.html.
  - The nav bar is clearly bootstrap. How do you know that?
  - But how are the rest of the controls positioned on the page? Notice that the div tags have custom styles – body, canvas-area and input-area.
- Look at styles.css. The body is the flexbox container. Can you guess what each property does?

```
.body {
    display: flex;
    flex-direction: row;
    flex-wrap: wrap;
    justify-content: space-around;
}
```

CSS Flexbox Properties

- Here are the 2 flex elements. Notice that both are elements of the top level container and containers themselves.

```
.canvas-area {
    flex: 2;
    display: flex;
    align-items: center;
    justify-content: center;
    margin: 10px;
}
```

```
.input-area {
    flex: 1;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    margin: 10px;
}
```

CSS Flexbox References

- There are lots of good references for the css flexbox. Here are just a few. This information is in your reading quiz for this week.
  - https://www.w3schools.com/css/css3_flexbox.asp
  - https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Flexbox

Getting Started – general.js

- Let's start by looking at the general.js file
  - What does it do?
  - Why might that be a good thing?
  - Is there a link element in index.html that loads bootstrap?
  - Is there a link element in index.html that loads styles.css?
  - The ability to import css into js was added with ES6 but isn't supported in all browsers. This code would generate an error in the browser if we weren't using babel to transpile our code.

Getting Started - memes.js

- Let’s start to work on memes.js
  - How is the contents of general.js made available to memes.js?
  - Is there a script element in index.html that loads general.js?
  - Creating a class
  - Creating a constructor
    - Initializing instance variables for all of the UI elements that we’ll refer to throughout our code. Notice that I’ve added a $ in the names of the instance variables that refer to dom elements. This isn’t conventional but there are lots of instance variables here and it seemed simpler than putting UI at the end of each variable name.

Introducing the Canvas

- The canvas allows your apps to use code to draw on the surface of a web page. In every instance, you'll start by getting a reference to either a 2d or 3d context and will call methods on the context object to draw on the canvas.
- W3Schools has a good tutorial on the canvas. Reading canvas related code is pretty intuitive. Let's look at some examples together:
  - https://www.w3schools.com/html/html5_canvas.asp
- This page has a good "cheat sheet" for properties and methods.
  - https://bucephalus.org/text/CanvasHandbook/CanvasHandbook.html

CreateCanvas Method

- Let’s start writing the code for our application by
  - Create the method createCanvas
    - It initializes the height and the width of the canvas. You can either set the size of the canvas in the html file OR in code. What property of the canvas do you think would set the height or set the width?
    - This method should get called when the page loads so we’ll call it in the constructor.

CreateMeme Method

- Let’s start writing the code for our application by
  - Create the method createMeme
    - What method do we use to draw an image on the context?
    - What method do we need to draw text on the context?
    - What method do we need to outline text on the context?
    - What property do we use to change the font?
    - What property do we use to change the font color?
    - What property to we use to change the outline color for the font?
    - Let’s see if we can write the code …
  - Call createMeme in your constructor. At this point it will load the default image. We’ll let the user pick an image in a few minutes.
  - Test and debug your code

AddEventListeners Method

- Now we're ready to make the application interactive. Let's start with something easy – allowing the user to change the text.
  - Our method createMeme already writes the text from the UI to the page. All we need to do is setup event handlers so that it gets called when
    - The user changes the text on the bottom - keyup
    - The user uses something other than the keyboard (like a context sensitive menu) to change the text on the bottom – change
    - Make sure that all of the methods we're calling interpret the keyword this to mean the class rather than an input tag. We call bind in addEventListeners or in the constructor.
  - Create the method addEventListeners.
  - Call addEventListeners in the constructor.
  - Test and debug your code.

DownloadMeme Method

- Let’s allow the user to download the meme they’ve created
  - Did you know that you can use an anchor tag to allow the user to download a file by specifying the file in the href attribute and adding a download attribute?
    - Add an href attribute to the download link on index.html. It should reference the src for the default image on the page.
    - Click the link. It should download the image (but not the meme).
  - Create the method downloadMeme. It will
    - Convert the contents of the canvas to data that can be saved to a file. What method will we need to do that?
    - Set the href attribute of the download link to the data from the canvas.
  - Add the binding as well as the event handler to addEventListeners.
  - Test and debug your code.

Asynchronous Processing

- Now we need to allow the user to select a file
  - https://developer.mozilla.org/en-US/docs/Web/API/FileReader
  - We're going to use a FileReader object to read the file and an Image object to load the actual image. Both of those processes can take a little while. We need to wait for the process to complete before we try to create the meme BUT we don't want to lock up the browser while we're waiting. This kind of processing - sending off a request, allowing the app to continue and doing the rest of the processing when the request finishes – is called ASYNCHRONOUS.

LoadImage Method - Design

- In order to do this with the FileReader we'll
  - Instantiate the file reader object
  - Identify the callback function that should be executed when the request completes
    - Instantiate a new Image
    - Identify the callback function that should be executed when the request completes - createMeme
    - Send off the request – set the src of the image
  - Send off the request – read the file that the user has selected
- The code is on the next slide

LoadImage Method - Code

```
loadImage() {
    if (this.$imageInput.files && this.$imageInput.files[0]) {
        let reader = new FileReader();
        reader.onload = () => {
            this.image = new Image();
            this.image.onload = () => {
                this.createMeme();
            };
            this.image.src = reader.result;
        };
        reader.readAsDataURL(this.$imageInput.files[0]);
    }
}
```

LoadImage Method – Add Event Listeners

- Create the function loadImage
- Change the addEventListeners method to
  - Bind the class to the method loadImage
  - Create an onchange event handler for the file input element that calls loadImage
- Test and debug your code

Remove the Default Image

- At this point we could remove the code that draws the default image on the page when the application loads.
- Or you can leave the code as it is and ship your application with a default image of your choice.
- OR you could write the information necessary to redraw the previous meme to local storage and load that when the page loads. This would be an “improvement” to the first version of the application and would count toward part 6 of the lab.

ResizeCanvas Method

- The last method let's us deal with images that are really big
  - Create the function resizeCanvas. The code is on the next slide.
  - Change the addEventListeners method to
    - Bind the class to the method resizeCanvas
    - Call the method resize Canvas in the method createMeme just before you draw the image

```
this.resizeCanvas(this.$canvas.height, this.$canvas.width);
```

  - Test and debug your code

ResizeCanvas Method - Code

```
resizeCanvas(canvasHeight, canvasWidth) {
    let height = canvasHeight;
    let width = canvasWidth;
    let scale = 1.0;
    while   (height > Math.min(1000, this.deviceWidth-30) &&
             width > Math.min(1000, this.deviceWidth-30)) {
             height /= 2;
             width /= 2;
             scale /= 2;
    }
    this.$canvas.height = height;
    this.$canvas.width = width;
    this.$context.scale(scale, scale);
}
```

New Syntax in this Lab

- Canvas properties and methods
  - height
  - width
  - getContext()
  - toDataURL()
- Context properties and methods
  - font
  - textAlign and textBaseline
  - linewidth, strokeStyle, fillStyle
  - drawImage()
  - strokeText()
  - fillText()
- FileReader object
  - readAsDataURL()
  - onload event fires when reading is done
- Image object
  - Setting the src attribute loads the image
  - onload event fires when load is done
- Both onload events are illustrations of “asynchronous” processing

Lab 4

- MemeCreator is the ONLY problem in lab 4 because it's complex BUT you're not done yet.
  - Part 6 asks you to provide some additional functionality to the app. You might
    - Allow the user to add text to the top of the meme
    - Allow the user to pick a font
    - Allow the user to pick a font color
    - Save the meme to local storage so that the last meme loads when the page loads
    - I'm sure there are other more creative things that you can think of …
  - For each of the 2 additional features
    - Add html to the page for the user to interact with
    - Change the JS class to add event handlers to your new UI elements
    - Change createMeme to add your features to the canvas
- Some of you may chose to work more with the canvas in your project.

What's Next

- Intro to AJAX
  - AJAX with fetch
  - Promises
  - Lab 5
    - Bookmarker version 2 (get image and title from a web api)
    - Weather app
  - Reading Quiz 5
- Don't forget
  - Reading Quiz 4 - The second (5 point) reading quiz question covers the syntactical elements we covered in building the MemeCreator.
  - Participation Score 4
  - Lab 4