Jesse M. Ellis - Development notebook for my personal website.

Note: *Revision 1 will be developed from 11/24/2024 to 12/11/2024.*

Table of Contents

First things first

11/24/2024

The point of this website is to build a personal website that highlight's personal projects using concepts we've learned in Caterina Paun's Intro to WebDev at PSU. I am going to attempt to create something that is more oriented toward my personal game development endeavours by theme-ing the website with elements from LunaLight. I will highlight some of the milestones in that project as well as other projects I have done.

Some technical constraints

To check the boxes around concepts learned in WebDev this site should be built using the following:

- HTML
- CSS
- JavaScript
- GitHub pages for deployment

High Level Implementation Plan

Concept: *This will be a single-page scrolling website with a fixed navigation bar that alows the user to jump to sections.*

1. Sketch a layout design for the following required sections:
    - Navbar: Fixed at the top.
    - About: Brief introduction and a professional photo.
    - Previous Work: Resume highlights or course/skills section.
    - Projects: Showcase 2-3 projects with links to GitHub/deployed sites.
    - Contact: Form for user inquiries.
2. Build basic HTML structure to represent each section.
3. Style the website with CSS.
4. Implement website interactivity with Javascript.
    - Scrolling
    - Form validation
    - Email contact
    - Embedded YouTube Video
5. Test, Refine and Repeat (steps 3-5)
6. Deployment with GitHub pages *(we'll worry about this when we get there)*

A starting point

Just to get things going I'm going to bring some initial files that create a form with user input functionality.

`form.html`, `form.js` and `styles.css`

These files create a simple form that gets user input and prints it to the console. We will build the entire website from here ;)

Design

11/25/2024

Here's a quick design sketch that captures te aesthetic of the site and some styling. I want the site to mainy highlight my current retro-style game ev project so I think using some pixel art fro mthat as the background is a cool idea.

Note: I did this in procreate, which is not great for this sort of thing.

Quick design sketch of the personal website.

Figma Design

11/27/2024

I decided to go ahead and design the website using Figma so that everything I need is ready to go when I start actual implementation.

Figma design for the personal website.

Navbar and background

11/30/2024

To get started on the above design I think we should get a simple navbar working that can scroll to the various sections. We should use classes to make things easier later.

```
<body>
    <header class="site-header">
      <nav class="navbar">
        <a href="#about" class="nav-link">About</a>
        <a href="#work" class="nav-link">Work</a>
        <a href="#projects" class="nav-link">Projects</a>
        <a href="#contact" class="nav-link">Contact</a>
      </nav>
    </header>

    <section id="about" class="about-section">
      <h1 class="section-title">About Me</h1>
    </section>

    <section id="work" class="work-section">
      <h2 class="section-title">Previous Work</h2>
    </section>

    <section id="projects" class="projects-section">
      <h2 class="section-title">Projects</h2>
    </section>

    <section id="contact" class="contact-section">
      <h2 class="section-title">Contact</h2>
    </section>
  </body>
```

This is a pretty bland webpage with just the html, so let's add the backgound image and some styling.

```
body {
  margin: 0;
  font-family: Arial, sans-serif;
  background-image: url("../images/Clouds.png");
  background-size: cover;
  background-position: left;
  background-attachment: fixed;
}
```

```
.navbar {
  position: fixed;
  top: 0;
  width: 100%;
  background: rgba(0, 0, 0, 0.7);
  color: #fff;
  padding: 1rem;
  text-align: center;
}

.nav-link {
  color: #fff;
  margin-right: 1rem;
  text-decoration: none;
}

section {
  padding: 2rem;
  color: #fff;
  margin-top: 2rem;
}
```

Much nicer!!! Now let's add in some simple JavaScript to scroll to the different sections by clicking the navbar links.

```
document.querySelectorAll('.nav-link').forEach(anchor => {
    anchor.addEventListener('click', function (e) {
      e.preventDefault();
      document.querySelector(this.getAttribute('href')).scrollIntoView({
        behavior: 'smooth'
      });
    });
  });
```

Lookin pretty good!

Initial navbar work for the personal website.

Profile Pic

12/01/2024

As in the Figma design I want the profile pic and name to be in a rounded box (pill shape) over the background such that it sends focus to the pixel art in the background and then name and pic of the person who created it. Which is me in this case ;). The html portion is not terribly complicated:

```
    <section id="title" class="title-section">
      <div class="title-box">
        <div class="title-text">
          <p>Developer Profile</p>
          <h1>Jesse M. Ellis</h1>
        </div>
        <img
          src="images/Profile.png"
          alt="Profile Picture"
          class="profile-pic"
        />
      </div>
    </section>
```

The CSS is a little more tricky as I need the position to be off center to highlight the background but also responsive when changing screen size or scrolling the view. Here is what I came up with:

```
.title-section {
  height: 70vh;
  display: flex;
  justify-content: right;
  margin-top: 3rem;
  margin-right: 10rem;
  align-items: center;
}

.title-box {
  display: flex;
  align-items: center;
  justify-content: space-between;
  background: rgba(0, 0, 0, 0.8);
  border-radius: 100px;
  padding: 1.5rem;
  box-shadow: 0 4px 10px rgba(0, 0, 0, 0.2);
  max-width: 80%;
  min-width: 400px;
}

.profile-pic {
  width: 150px;
  height: 150px;
  border-radius: 50%;
  margin-left: 1.5rem;
  object-fit: cover;
}
```

```
.title-text {
  text-align: center;
  padding-left: 1rem;
}

.title-text h1 {
  margin: 0;
  font-size: 1.8rem;
  color: #f0caca;
}

.title-text p {
  margin: 0;
  color: #a07d7d;
  font-size: 1rem;
}
```

While were at it we can use a similar approach and set up our other sections. We'll want to add a class for a box in each section like this,

```
    <section id="about" class="about-section">
      <div class="about-box">
          <h1>About Me</h1>
        </div>
      </div>
    </section>
```

Then we can add some initial CSS styling,

```
.about-section,
.work-section,
.projects-section,
.contact-section {
  display: flex;
  justify-content: center;
  width: 100%;
}

.about-box,
.work-box,
.project-box,
.contact-box {
  background: rgba(0, 0, 0, 0.8);
  border-radius: 20px;
  padding: 1rem;
  box-shadow: 0 4px 10px rgba(0, 0, 0, 0.2);
  width: 95%;
  color: #fff;
  box-sizing: border-box;
  text-align: left;
}
```

About Section

12/03/2024

There are three somewhat tricky things I want to accomplish in the about section that we may use in the other sections. First we need to split the box into two columns so that we can have text on the left and images on the right. Then, I want to load in the text using JavaScript so that our html doesn't get overloaded with text. Finally, I want to add a component that allows you to flip through images on the right side of the text.

Columns in the box

To Accomplish this we can use flex. Each of our "columns" will be set to flex: 1 to ensure they take up equal space in the box. Additionally we can easily use media queries to set our text and images to display vertically when the screen size is narrow.

Load text file

Conveniently we can use the fetch api to do this. This also allows for easy error handling in cases that the file does not load. When you pass a relative file path like 'about.txt' to fetch, it looks for the file relative to the location of our html file being served. E.g, since our site.html is in the root folder, fetch('about.txt') looks for the file at oursite.com/about.txt

```
function loadTextFromFile(filePath, targetSelector) {
  fetch(filePath)
    .then((response) => {
      if (!response.ok) {
        throw new Error("Failed to fetch file: " + response.statusText);
      }
      return response.text();
    })
    .then((text) => {
      const targetElement = document.querySelector(targetSelector);
      if (targetElement) {
        targetElement.textContent = text;
      } else {
        console.error(`Target element not found: ${targetSelector}`);
      }
    })
    .catch((error) => {
      console.error("Error loading text from file:", error);
    });
}
```

Now we need a way to know when to load the text so let's add an event listener. We can listen for when the page finishes loading. The browser will send a 'load' flag on the window object when everything has loaded.

```
window.addEventListener("load", () => {
  loadTextFromFile("about.txt", ".about-paragraph");
});
```

Image carousel

I thought it would be cool to show images related to the about section. The user shoudl be able to flip through the images. To accomplish this we can code in the images to our html section under two class types, current-image and hidden-image. the carousel effect will be created by setting the class value of each image, e.g. the image we want to see is set to current and the rest are set to hidden. To flip through the images we will create a previous button and a next button. We can handle this logic in JavaScript by creating event listeners to get the images and buttons, detect button clicks and define a function to set the current and hidden images.

```
// Image carousel
document.addEventListener("DOMContentLoaded", () => {
  const images = document.querySelectorAll(".about-image img");
  const prevBtn = document.querySelector(".prev-btn");
  const nextBtn = document.querySelector(".next-btn");

  let currentIndex = 0;

  // Function to update image visibility
  function updateImages() {
    images.forEach((img, index) => {
      img.classList.toggle("current-image", index === currentIndex);
      img.classList.toggle("hidden-image", index !== currentIndex);
    });
  }

  // Event listeners for navigation buttons
  prevBtn.addEventListener("click", () => {
    currentIndex = (currentIndex - 1 + images.length) % images.length;
    updateImages();
  });

  nextBtn.addEventListener("click", () => {
    currentIndex = (currentIndex + 1) % images.length;
    updateImages();
  });

  updateImages();
});
```

Side-quest

I had a cool idea that the title section could scroll up at a slower rate than the about section until they are touching. As it scrolls up the opacity should decrease until the component "disappears". I think this will make the site feel a little more dynamic to the user and could also be used for a parallaxing background at some point!

We need to get seom eitems and values using query selectors, the title box, about section, the window scroll Y value and the upward bound of the about section. As we scroll the view we can explicitly translate the title box Y value at a slower rate as well as decrease the opacity. This way once the title box meets the about section the title box will be fully transparent and move with the about section. Initially this worked but broke my navigation bar. After some troubleshooting I found that the I neededto explicitly set the title box translate to zero when not scrolling.

```
// dynamic scrolling
document.addEventListener("scroll", () => {
  const titleBox = document.querySelector(".title-box");
  const aboutSection = document.querySelector("#about");
  const scrollY = window.scrollY;
  const aboutTop = aboutSection.getBoundingClientRect().top + window.scrollY;
  const slowScrollRate = 0.5;

  // move title-box if it's above the viewport and not yet touching about-section
  if (scrollY < aboutTop) {
    // Adjust position adn opacity
    titleBox.style.transform = `translateY(${scrollY * slowScrollRate}px)`;
    const opacity = 1 - scrollY / aboutTop;
    titleBox.style.opacity = Math.max(opacity, 0);
  } else {
    titleBox.style.transform = "translateY(0)";
    titleBox.style.opacity = 0;
  }
});
```

Project Section

12/06/2024

For the projects section I want to highlight two personal projects, maybe include this website as a third. I want to organize each project similarly to how I did the previous section except with multiple stacked sections. I should just be able to copy the main components from the about section and repurpose them. I'll have text on the left and an interactive component on the right. That said, The interactive coponent will differ with each section.

To accomplish stacking the different projet sections I am going to introduce some grid organization into the projects section. Withe simple CSS we can achieve a nice staked layout within our projects section.

CSS

```
.grid-container {
  display: grid;
  grid-template-columns: 1fr;
  gap: 2rem;
  width: 100%;
}

.grid-item {
  display: grid;
  grid-template-columns: 1fr 1fr;
  gap: 2rem;
  background: rgba(42, 15, 59, 0.8);
  padding: 2rem;
  border-radius: 20px;
  color: #ffe8e8;
  align-items: center;
}
```

HTML

```
    <section id="projects" class="projects-section">
      <div class="project-box">
        <div class="grid-container">
          <h1 class="project-box-title">Personal Projects</h1>

          <!-- Project 1 -->
          <section class="grid-item">
            <!-- Project 1 First Column: Project Overview Text -->
            <div class="project-text">
              <h1 class="project-title">LunaLight - A Retro Game Project,</h1>
              <p id="project1-paragraph" class="project-paragraph">
                Loading...
              </p>
            </div>
            <!-- Project 1 Second Column: Video -->
            <div class="project1-video-container">
              <h1 class="project-title">LunaLight - Teaser</h1>
            </div>
          </section>

          <!-- Project 2 -->
          <section class="grid-item">
            <!-- Project 2 First Column: Project Overview Text -->
            <div class="project-text">
              <h1 class="project-title">
                EightBiterator - A Retro Game Melody Generator,
              </h1>
              <p id="project2-paragraph" class="project-paragraph">
                Loading...
              </p>
            </div>
            <!-- Project 2 Second Column: Images -->
            <div class="project2-form-container">
              <h1 class="project-title">EightBiterator</h1>
            </div>
          </section>
        </div>
      </div>
    </section>
```

Now let's get into the interactive project components

Project 1 - My Retro Game

The first project highlight will be for my personal retro game project and include an imbedded you tube video as the interactive component. To help with this I'll be referencing som eof the infomation here: HTML YouTube Videos.

So we'll be using a component called `iframe` which allows us to embed video and set many useful attributes.

As it turns out, if you click share on a YouTube video there is on option for "embed"! You can simply copy and paste it into your HTML!

Let's break down some of the less obvious attributes we are copying and embedding, for starters, it does use iframe.

```
<iframe
  width="560"
  height="315"
  src="https://www.youtube.com/embed/J_KjVaEGNk4?si=O0-429xP-P2r7JSx&autoplay=1&loop=1&playl
  title="YouTube video player"
  frameborder="0"
  allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-pi
  referrerpolicy="strict-origin-when-cross-origin"
  allowfullscreen>
</iframe>
```

Ok, firstly the I have set some important functionality within the URL itself,

Source URL `src="https://www.youtube.com/embed/J_KjVaEGNk4?si=O0-429xP-P2r7JSx&autoplay=1&lo`

- `autoplay=1`: Makes the video play automatically.
- `loop=1`: Enables continuous looping of the video.
- `playlist=OUR_VIDEO_ID`: Ensures looping by identifying the video as a playlist of one.
- `mute=1`: Starts the video muted to satisfy browser autoplay policies.

Allow list `allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture; web-share"`

- `accelerometer`: Allows device motion data (e.g., for interactive videos).
- `autoplay`: Enables automatic playback of the video.
- `clipboard-write`: Allows the iframe to write to the clipboard.
- `encrypted-media`: Permits encrypted content to play.
- `gyroscope`: Allows access to gyroscope data for interactive experiences.
- `picture-in-picture`: Allows the video to play in a smaller floating window outside the iframe.
- `web-share`: Supports web sharing features.

Referrer Policy `referrerpolicy="strict-origin-when-cross-origin"`

- `strict-origin-when-cross-origin`: Sends only the origin (not the full URL) as the referrer when navigating cross-origin, enhancing privacy.

Project 2 - EightBiterator

In this section I want to showcase a really cool program I built that generates eight-bit retro game music melody samples. A user can input various attributes to customize the meody and EightBiterator will generate a melody based on those parameters.

To showcase this I would like to create a user form that allows the user to input the parameters and the hit a button that says "generate". As the program runs it will give some indication as to what it is doing and play each waveform. It will then prompt the user if they woudl like to save the generated wav file.

This is actually going to be quite difficult I think... To be continued...

Previous Work Section

12/07/2024

For this section I am going to keep things relatively simple. I also think ths will be a good case for the use of cards. There will be three cards that give a brief overview of my experiences at a few of my favorite past work experiences. In addition each card wil act as a button to open a link to the relevant linked-in page for each of the work experiences.

```
<section id="work" class="work-section">
  <div class="work-box">
    <h2 class="work-section-title">Previous Work</h2>
    <div class="work-grid">
      <!-- Airship Card 1 -->
      <div class="work-card">
        <img src="images/airship.png" alt="Airship Software Icon" class="work-card-image">
        <h3 class="work-card-title">Software Development Intern</h3>
        <p id="airship-text" class="work-card-text">Loading...</p>
      </div>
      <!-- Jama Card 2 -->
      <div class="work-card">
        <img src="images/jama.png" alt="Jama Software Icon" class="work-card-image">
        <h3 class="work-card-title">Software Development Intern</h3>
        <p id="jama-text" class="work-card-text">Loading...</p>
      </div>
      <!-- Intel Card 3 -->
      <div class="work-card">
        <img src="images/intel.png" alt="Intel Icon" class="work-card-image">
        <h3 class="work-card-title">Lead Engineering Technician</h3>
        <p id="intel-text" class="work-card-text">Loading...</p>
      </div>
    </div>
  </div>
</section>
```

To make each card a link I can add an `<a>` tag element,

```
<a href="link_to_linkedin" target="_blank" class="work-card-link">

...rest of card html

</a>
```

and add the following CSS to create a nice effect when hovering above each card.

```
.work-card-link {
  text-decoration: none;
  color: inherit;
  display: block;
  transition: transform 0.2s ease-in-out;
}

.work-card-link:hover .work-card {
  box-shadow: 0 6px 12px rgba(0, 0, 0, 0.3);
}
```

Contact Section

12/07/2024

Here the user should be prompted to enter their email and a message if they would like to collaborate or work.

We will use a simple form that allows input for an email and a message. The form will have a submit button and reset button a well a validation. Submitting the message should validate user input and send an email to my preferred email with their contact info and message. Resetting should clear the form.

So how do we send an email from the form? We also need to think about deployment to github pages. Here is a good stack about email options when deploying to GH pages.

- Stack Overflow - Email from GitHub Pages

In this various people discuss options and honestly I thinkthe best option would be to use a third party service.

Here are some options:

- Formspree - Free for < 50 emails per month - a little more concise
- Postmark - Free for < 100 emails per month - pretty robust functionality

I am going to try Formspree. Creating an account and setting up a form server is super easy!

Once set up it gives the server URL and method:

`action="https://formspree.io/f/xwpkjpal"`

`method="POST"`

We can now build our `html` form section and include our email server configuration.

```
<section id="contact" class="contact-section">
  <div class="contact-box">
    <h2 class="section-title">Contact</h2>
    <form id="contact-form" method="POST" action="https://formspree.io/f/xwpkjpal" class="co
      <label for="email">Your Email:</label>
      <input type="email" id="email" name="email" placeholder="Enter your email" required>
      <label for="message">Your Message:</label>
      <textarea id="message" name="message" rows="5" placeholder="Enter your message" requi

      <div class="form-buttons">
        <button type="submit">Submit</button>
        <button type="reset">Clear</button>
      </div>
    </form>
  </div>
</section>
```

This works really well just as html however there are some things that I think need to be fixed.

- Successful submission redirects to a formspree confirmation screen.
- When returning to our site, the form is not cleared

Formspree includes an option to disable the redirect if you upgade to a paid account.. we don't want to do that. Instead we can fix these issues by taking control of what happens on submit action via JavaScript. We can submit the form using the fetch API and catch the response. We will want to handle potential errors as well as communciate to the user success or fail. We will use a simple alert message for both. On success, we will reset the form.

```
// On contact submit
document.getElementById("contact-form").addEventListener("submit", async function (event) {
  // Take control of form submission
  event.preventDefault();
  const form = event.target;
  const formData = new FormData(form);
  try {
    // Submit the form data using the Fetch API and catch response
    const response = await fetch(form.action, {
      method: form.method,
      body: formData,
      headers: {
        Accept: "application/json",
      },
    });
    if (response.ok) {
      // Success! give user a message
      alert("Thank you! Your message has been sent.");
      form.reset(); // Clear the form
    } else {
      // Fail :( give user a message
      alert("Oops! There was a problem submitting your form.");
    }
  } catch (error) {
    // Log any errors
    console.error("Error:", error);
    alert("There was a problem submitting your form. Please try again later.");
  }
});
```

Project Section Continued

12/08/2024

Capstone

I've decided to include my capstone project as one of the personal proejcts in this section. This will involve a video of the demo that I will share as a YouTube video and then embed here in the Website. We can set this up excatly the same way as our first project!

EightBiterator Continued

Although this would be really cool I've decided it is outside the scope of this project :(.

Parallaxing Background

12/08/2024

Another fun idea I had that is defintely within the scope is to create a parallax background. I think this will give a really cool user experience and highlight my love of art and game development.

As the user scrolls to the bottom of the page there will be layered background elements that move at different speeds giving the effect that the view is descending into a pixel art view of the background. The parallaxing will give a depth and movement effect that will be fun for the user experience.

We have already done some of the work for this with our dynamic scrolling.

The plan

We are going to create additional *background* layers that will be layered on top of the actual background. We will update our JavaScript scroll event listener to control the scroll speed of each of these layers. Finally we will use CSS to ensure the layers are styled correctly to get our parallax effect.

First I tried adding additional layers to the body however this proved difficult to control the position on various screen sizes.

Ultimately I decided to add another section to the bottom of the page that will include the different parallax layers. This will be our "credits" section and as th euser scrolls down into it the different layers will come into view with paralax effect giving a sense of depth.

We will control the different scroll rates using JavaScript. We just need to add a few things to what we had before.

```
// dynamic scrolling
  document.addEventListener("scroll", () => {
    const navBar = document.querySelector(".navbar");
    const titleBox = document.querySelector(".title-box");
    const aboutSection = document.querySelector("#about");
    const contactSection = document.querySelector("#contact");
    const creditsSection = document.querySelector("#credits");
    const parallaxlayer1 = document.querySelector(".layer-1");
    const parallaxlayer2 = document.querySelector(".layer-2");
    const parallaxlayer3 = document.querySelector(".layer-3");

    const scrollY = window.scrollY;
    const aboutTop = aboutSection.getBoundingClientRect().top + scrollY;
    const contactTop = contactSection.getBoundingClientRect().top + scrollY;
    const sectionTop = creditsSection.offsetTop;
    const slowScrollRate = 0.5;

    if (scrollY < aboutTop) {
      const opacity1 = scrollY / aboutTop;
      titleBox.style.transform = `translateY(${scrollY * slowScrollRate}px)`;
      titleBox.style.opacity = Math.max(1 - opacity1, 0);
    } else {
      titleBox.style.transform = "translateY(0)";
      titleBox.style.opacity = 0;
    }

    if (scrollY > contactTop) {
      const opacity2 = (scrollY - contactTop) / (document.body.scrollHeight - contactTop);
      navBar.style.opacity = Math.max(1 - opacity2, 0.5);
    } else {
      navBar.style.opacity = 1;
    }

    const relativeScrollY = sectionTop - scrollY;
    parallaxlayer1.style.transform = `translateY(${relativeScrollY * 0.2}px)`;
    parallaxlayer2.style.transform = `translateY(${relativeScrollY * 0.4}px)`;
    parallaxlayer3.style.transform = `translateY(${relativeScrollY * 0.6}px)`;
  });
```

Deployment

12/11/2024

To deploy our website we are going to use GitHub pages. Deployment through pages is straightforward and easy.

On GitHub in your repository

1. Make repository public.
2. Navigate to Settings -> Pages
3. Choose "Deploy form branch"
4. Set to main and /root
5. Click save.

There are further customizations and tools you can use but for now we will keep it at it's simplest form.

Once deployed, you will recieve a link to your deployed website. If changes are made to the repository Github will automatically rebuild and deploy your website with the updated changes.

Troubelshooting

For our website, initial deployment was not displaying our background image... what the heck? We are setting our background image in our CSS file in the body element. This is the only image being set this way and every other asset is loading. Here is the error we get when inspecting the page.

```
Failed to load resource: the server responded with a status of
404 () Moon.jpg:1
```

Here is our current CSS where we are setting the background:

```
body {
  margin: 0;
  font-family: Arial, sans-serif;
  background-image: url("/images/Moon.jpg");
  background-size: cover;
  background-position: left;
  background-repeat: no-repeat;
  background-attachment: fixed;
  min-width: 768px;
}
```

Failed troubleshooting attempts:

- verified the file path and can not seem to find a problem there..
- tried re-placing the file.. still not working.
- tried converting the file type, `png` -> `jpg`
- tried using absolute path `../images/Moon.jpg`

Possible Solution:

- Moved background styling out of the CSS file and into the HTML file to align with working images.

```
<body
    style="
      background-image: url('images/Moon.png');
      background-size: cover;
      background-repeat: no-repeat;
      background-attachment: fixed;
    "
  >
```

This worked.. phew..