

## HTML/CSS/Javascript

### HTML:

- The foundation of any website serves as its structural framework. HTML uses tags to mark up content, telling browsers how to interpret and display the information. It defines:
  - The basic structure and content of web pages
  - Text elements like headings, paragraphs, and lists
  - Links between pages
  - Media (images, videos, etc)
  - Forms for user input

### Javascript:

- Adds interactivity and dynamic functionality to websites. It is a programming language that runs in the browser, allowing websites to respond to user actions and create interactive user experiences. It defines:
  - User interaction handling (clicks, scrolls, form submissions)
  - Content updates without page reloads
  - Form validation and data processing
  - Animations and visual effects
  - API communication to fetch or send data
  - Creating and manipulating HTML elements
  - Building complex web applications

### CSS:

- CSS controls the visual presentation and layout of HTML elements. Without CSS, websites would be plain text documents with basic formatting. CSS makes these structures into appealing designs that improve the user experience. It also allows for separation of content from presentation, making maintenance easier. It defines:
  - Colors, fonts, and text formatting
  - Layout and positioning of elements
  - Responsive design (adapting to different screen sizes)
  - Animations and transitions
  - Visual effects and decorations
  - Consistent styling across multiple pages

### Example:

For my example of the use of these three concepts, I'm including the DCS Website I created.

**Website Link:** <https://chriskenny16.github.io/DCS.Website.Final/index.html>

```
1  TYPE html>
2  . lang="en">
3  >
4  meta charset="UTF-8">
5  meta name="viewport" content="width=device-width, initial-scale=1.0">
6  title>Alumni Network - Digital and Computational Studies</title>
7  !-- Bates official CSS -->
8  link rel="stylesheet" href="https://www.bates.edu/wp-content/themes/b/bates-framework/styles/theme.css?ver=5.19.4">
9  link rel="stylesheet" href="https://www.bates.edu/wp-content/themes/b/bates-framework/style.css?ver=3.6.0">
10 !-- Local CSS -->
11 link rel="stylesheet" href="css/main.css">
12 link rel="stylesheet" href="css/responsive.css">
13 link rel="stylesheet" href="css/placeholder.css">
```

In this HTML file, here you can see the links to the different CSS files used for Styling.

```
!-- Alumni Network Section -->
section id="alumni-network" class="">
  <div class="">
    <h2 class="">Alumni Network</h2>
    <div class="">
      <h3>Bates Alumni Working in DCS Positions</h3>
      <div class="">
        <div class="">
          <h4>Working in Software Development</h4>
        </div>
        <div class="">
          <div class="">
            
          </div>
          <div class="">
            <h4>Jane Doe '18, PHD</h4>
            <p class="">Software Engineer at Google</p>
            <p class="">Jane works on Google's core search algorithms, applying techniques from
            <a href="#" class="">Contact</a>
          </div>
        </div>
        <div class="">
          <div class="">
            
          </div>
          <div class="">
            <h4>Michael Chen '20</h4>
            <p class="">Full-Stack Developer at Shopify</p>
            <p class="">Michael helps build and maintain Shopify's e-commerce platform. He credi
            <a href="#" class="">Contact</a>
          </div>
        </div>
      </div>
    </div>
  </div>
```

This HTML code shows a section of a website dedicated to Bates alumni working in DCS. It is organized with a main section with ID alumni-network, A heading Alumni Network (h2), A subsection for Bates Alumni Working in DCS Positions (h3), A category for Working in Software Development (h4), and two alumni profiles. Each profile includes a placeholder profile image, their name and graduation year, their current job title and company, a brief description of their work, and a contact link.

```

.opportunity-item {
  display: flex;
  margin-bottom: 30px;
  background-color: #f9f9f9;
  border-radius: 8px;
  box-shadow: 0 3px 10px rgba(0,0,0,0.1);
  overflow: hidden;
}

.company-logo {
  display: flex;
  justify-content: center;
  align-items: center;
  background-color: #8e1919;
  color: white;
  padding: 30px;
  font-size: 2.5rem;
  min-width: 120px;
}

.opportunity-details {
  padding: 25px;
  flex: 1;
}

.opportunity-header {
  display: flex;
  justify-content: space-between;
  align-items: flex-start;
  margin-bottom: 10px;
}

.opportunity-header h4 {
  color: #333;
  font-size: 1.3rem;
  margin: 0;
}

.opportunity-location {
  color: #666;
  font-size: 0.9rem;
}

```

This CSS file is styling components for the career opportunity page, where:

- .opportunity-item - Styles each job/opportunity card
- .company-logo - Styles a company logo area
- .opportunity-details - Styles the main content area
- .opportunity-header - Styles the header section of each listing
- .opportunity-header h4 - Styles the h4 heading within headers
- .opportunity-location - Styles the location text

```

initializeNavigation();
...
function initializeOpportunityFilters(): void
initializeOpportunityFilters();
});

// Navigation functionality
function initializeNavigation() {
  // Get all navigation links
  const navLinks = document.querySelectorAll('.main-nav li a');

  // Add click event listeners
  navLinks.forEach(link => {
    link.addEventListener('click', function(e) {
      // Remove active class from all links
      navLinks.forEach(l => l.parentElement.classList.remove('active'));

      // Add active class to clicked link
      this.parentElement.classList.add('active');

      // If the link points to an ID on the page, smooth scroll to it
      const href = this.getAttribute('href');
      if (href.startsWith('#') && href.length > 1) {
        e.preventDefault();
        const targetId = href.substring(1);
        const targetElement = document.getElementById(targetId);

        if (targetElement) {
          window.scrollTo({
            top: targetElement.offsetTop - 100,
            behavior: 'smooth'
          });
        }
      }
    });
  });
}
}

```

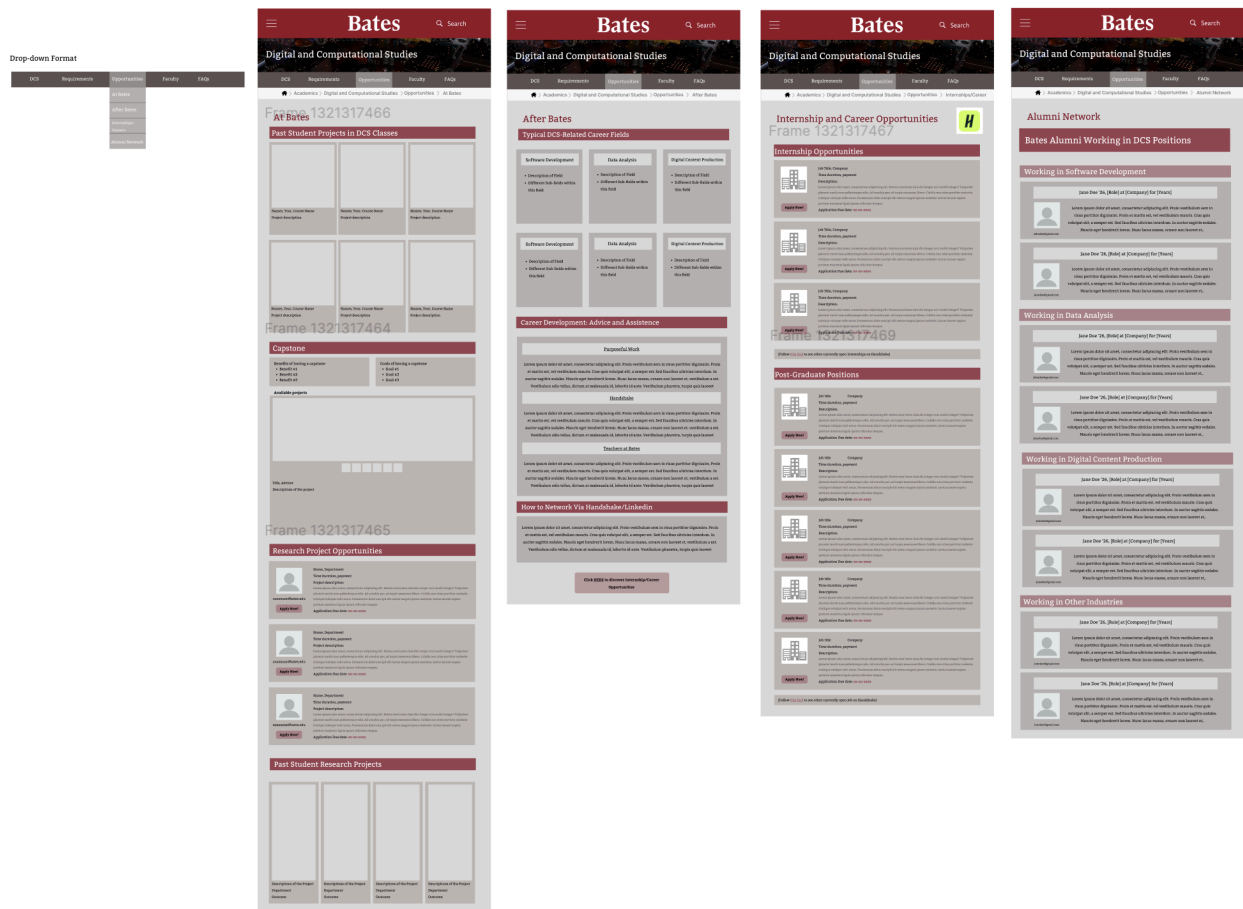
This JavaScript code is initializing and setting up navigation functionality for a website.

- The code starts by adding an event listener that runs when the DOM content is fully loaded (DOMContentLoaded). This ensures all HTML elements are available before JavaScript tries to interact with them.
- Inside this event listener, it calls four initialization functions:
  - initializeNavigation()
  - initializeSearch()
  - initializeProfileButtons()
  - initializeOpportunityFilters()
- The initializeNavigation() function:
  - Gets all navigation links in the main navigation menu (all <a> elements inside <li> elements in .main-nav)
  - Adds a click event listener to each navigation link
- When a link with an internal page reference is clicked, it:
  - Prevents the default jump behavior
  - Extracts the target element ID from the href
  - Finds the target element on the page
  - Smoothly scrolls to it with a 100px offset from the top using window.scrollTo()

## Figma

Figma is a great way to begin the brainstorming process for a website. Through creating a prototype of the DCS Major website (after Bates opportunities) we were able to clearly think through and picture our ideas before we got to the implementation aspect - which was implementation much less overwhelming. I think that this is a great way to start the process of building a website, where you can picture exactly how the website will look/flow before getting into the grunt work.

## Example:



## Ssh & Scp

Ssh:

- Ssh (secure shell) provides secure access to remote computers. It is used to securely log into remote servers and execute commands, as it creates an encrypted connection between your local machine and remote servers.

Scp:

- Scps (secure copy protocol) are built on top of Ssh and used for securely transferring files between computers. It allows you to to copy files to/from remote servers without setting up an FTP server, and uses the same authentication and security as SSH.

### Example

Making an alias that creates a shortcut that simplifies my Git and SSH commands:

Code:

- Create/edit SSH config file: `nano ~/.ssh/config`
- Add the GitHub repository configuration:
  - Host dcs325-charlie
    - HostName github.com
    - User git
    - IdentityFile ~/.ssh/id\_ed25519
    - PreferredAuthentications publickey
- Save the file: `Ctrl + O, [Enter]`
- Set up proper permissions: `chmod 600 ~/.ssh/config`
- Update the Git remote:
  - `cd ~/dcs325_w2025_final_charlie`
  - `git remote set-url origin dcs325-charlie`

Now, when I run commands such as `git push origin main`, Git will use this SSH alias to connect to GitHub. Instead of using `git@github.com:blawson-bates/dcs325_w2025_final_charlie.git`, I can use `dcs325-charlie`. The alias automatically uses my SSH key when connecting to GitHub, so I don't need to specify it each time. It ensures that I am always using the same settings when connecting to a particular server, and that all of my connection settings are stored in one place.

### **Cursor/Firebase/Tailwind/ShadCN**

#### Tailwind CSS:

Tailwind is a CSS framework that allows developers to build custom designs directly in their HTML markup by applying pre-defined utility classes. Rather than writing custom CSS styles, you apply small, single-purpose classes directly to HTML elements.

- Provides low-level utility classes for styling (like `flex`, `pt-4`, `text-center`)
- Allows for rapid UI development with minimal custom CSS
- You can make it super customizable through configuration files
- Has a responsive design built-in

### ShadCN/UI:

ShadCN/UI is a collection of reusable UI components built with Tailwind CSS. It's not like a traditional component library that you install as a package, but rather a set of components that you can copy and customize for your projects.

- Provides pre-built, customizable components like buttons, cards, and dialogs
- Styled with Tailwind CSS
- Components are copied into a project rather than installed
- Gives you full control to modify components for specific needs

### Cursor:

Cursor is an AI-powered code editor designed specifically for programming with AI assistance. It's built on top of Visual Studio Code but integrates AI capabilities directly into the development workflow. It can directly edit your files as well as work with your computer's terminal to download specific packages or create/move files. It's a very helpful tool in building a website, but often makes errors that you need to manually correct.

### Firebase for Backend

Firebase is Google's platform for developing mobile and web applications. It provides a comprehensive backend-as-a-service solution that eliminates the need to manage servers for many web applications. Using firebase allowed me to create a website that had an authentication aspect, where the user can create an account and input information. All of the information is stored within firebase and is accessible. It also ensures that only people who are authenticated can change/input any information.

### Example:

Using cursor, I set up a website that displays a login page. If you click sign up, it takes you to another page that allows you to create an account. Below is an image of the account pages as well as a snapshot of part of the code that was used to create it:

## Login

Enter your email below to login to your account

Email

Password

[Forgot your password?](#)

Don't have an account? [Sign up](#)

By continuing, you agree to our [Terms of Service](#) and [Privacy Policy](#).

## Create an account

Enter your information to create an account

First name

Last name

Email

Password

Already have an account? [Sign in](#)




```

1 import {
2   GoogleAuthProvider,
3   signInWithPopup,
4   signOut,
5   onAuthStateChanged,
6   User,
7   createUserWithEmailAndPassword,
8   signInWithEmailAndPassword
9 } from "firebase/auth";
10 import { auth } from "../firebase";
11
12 // Sign in with Google
13 export const signInWithGoogle = async (): Promise<User> => {
14   try {
15     const provider = new GoogleAuthProvider();
16     const result = await signInWithPopup(auth, provider);
17     return result.user;
18   } catch (error) {
19     console.error("Error signing in with Google", error);
20     throw error;
21   }
22 };
23
24 // Sign up with email and password
25 export const createUser = async (email: string, password: string): Promise<User> => {
26   try {
27     const userCredential = await createUserWithEmailAndPassword(auth, email, password);
28     return userCredential.user;
29   } catch (error) {
30     console.error("Error creating user:", error);
31     throw error;
32   }
33 };
34
35 // Sign in with email and password
36 export const signInWithEmail = async (email: string, password: string): Promise<User> => {
37   try {
38     const userCredential = await signInWithEmailAndPassword(auth, email, password);
39     return userCredential.user;
40   } catch (error) {
41     console.error("Error signing in with email:", error);
42     throw error;
43   }
44 };

```

Once you create an account, it stores the information in firebase. You can then use the same credentials to login.

<div> <input type="text" value="Search by email address, phone number, or user UID"/> <div> Add user </div> </div>				
Identifier	Providers	Created ↓	Signed In	User UID
bon@gmail.com		Apr 20, 2025	Apr 20, 2025	yK5hZ8sSdsN7OYqu5EvWu3K...
<div> <div>Rows per page:</div> <div>50</div> <div>1 – 1 of 1</div> </div>				

After logging in, you are brought to a page that has a welcome message as well as prompts the user to add more information. Below is an image of this page as well as part of the code used to create it:

Student Dashboard

Sign Out

Profile Information

User Info

Major Experience

To-Do List

Settings

Welcome, C!

bon@gmail.com

Your User Information

Name

Last Name

Major

Student ID

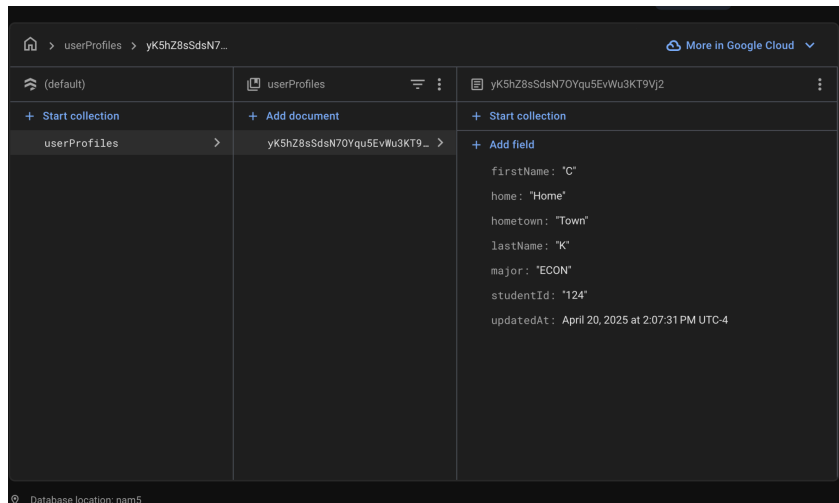
Home

Hometown

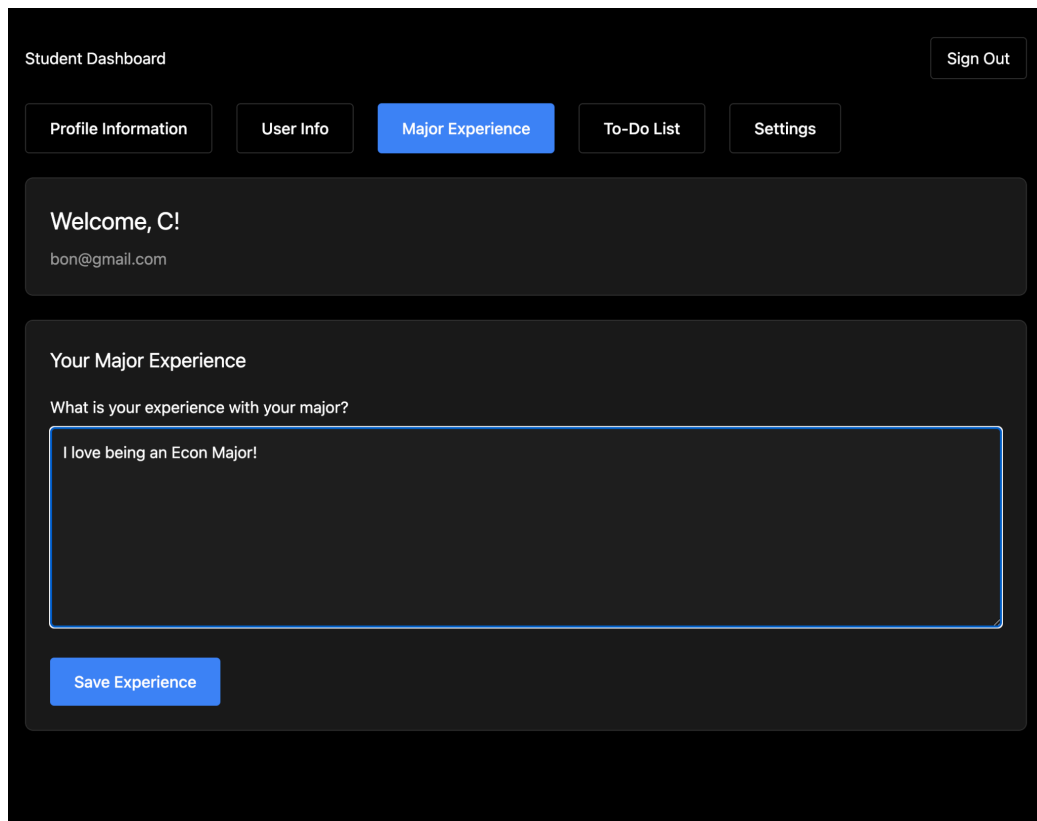
Save User Info

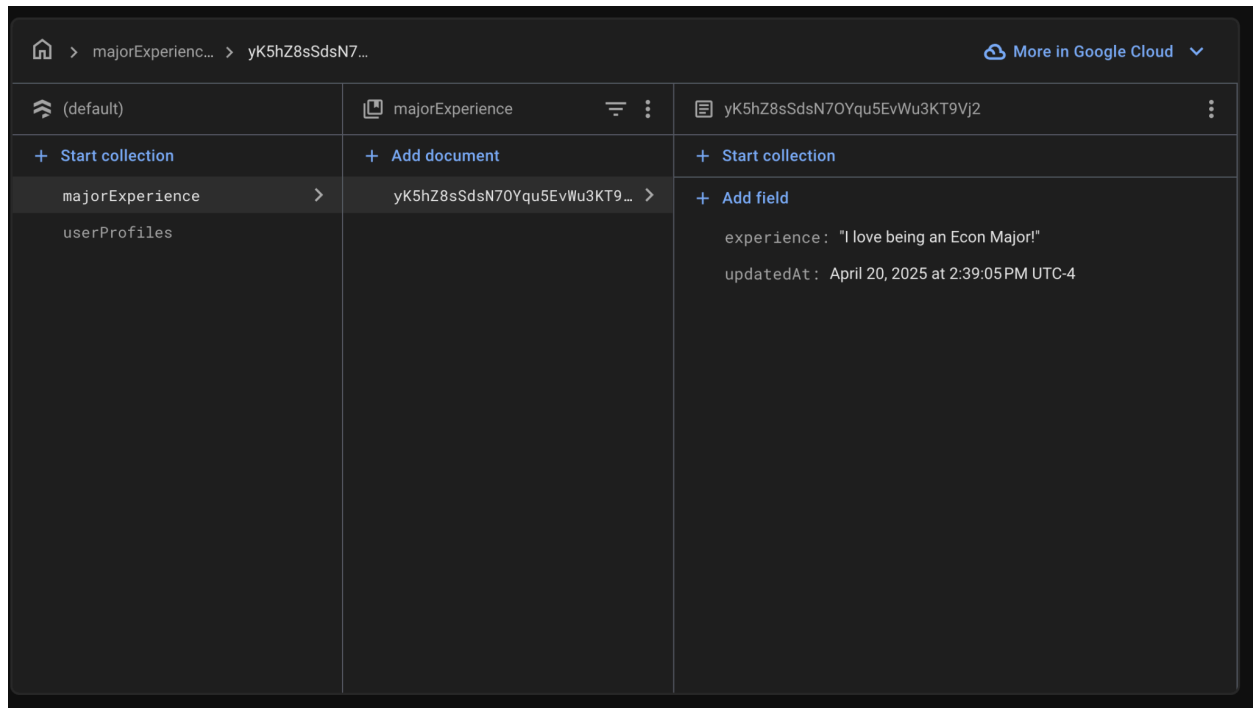
```
16 type UserInfo = {
17   name: string;
18   lastName: string;
19   major: string;
20   studentId: string;
21   home: string;
22   hometown: string;
23   updatedAt: Timestamp;
24 };
25
26 type DashboardProps = {
27   user: User;
28   onSignOut: () => void;
29 };
30
31 type NavItem = "profile" | "userInfo" | "todos" | "settings";
32
33 // Add these styles at the beginning of the file, outside the component
34 const spinnerKeyframes = `
35 @keyframes spin {
36   0% { transform: rotate(0deg); }
37   100% { transform: rotate(360deg); }
38 }
39 `;
40
41 export function Dashboard({ user, onSignOut }: DashboardProps) {
42   const [profile, setProfile] = useState<UserProfile>({
43     firstName: "",
44     lastName: "",
45     major: "",
46     studentId: "",
47     home: "",
48     hometown: "",
49     updatedAt: Timestamp.now(),
50   });
51
52   const [userInfo, setUserInfo] = useState<UserInfo>({
53     name: "",
54     lastName: "",
55     major: "",
56     studentId: "",
57     home: "",
58     hometown: "",
59     updatedAt: Timestamp.now(),
60   });
61 }
```

Once you press “Save User Info,” it is stored within firestore, as shown in the image below:



Additionally, I added a tab titled “Major Experience” that allows you to input text relating to your major, and saves it to firebase, as shown in the images below:





To conclude, through working with these different elements, I was able to create a website that prompts a user login, creates a user profile, and stores that profile in firebase. I was able to create a landing page for the user as well as create tabs where the user can input different types of information and store it on the backend. Firebase also separates the information based on which tab you are on, so for Major Experience, you can directly see what the user input was (rather than the user profile information).

This project used Tailwind and ShadCN/UI, illustrated through

- A components.json file in the project root.
- Tailwind CSS configured (with tailwind.config.js), which ShadCN/UI requires
- There are dependencies in package.json:
  - @radix-ui/react-label
  - @radix-ui/react-slot
  - Class-variance-authority
  - Tailwind-merge
  - Tailwindcss-animate

## Bootstrap/React

### Bootstrap:

A front-end CSS framework that provides pre-designed components and styling to help developers build responsive websites quickly.

- Bootstrap uses a 12-column grid system that automatically adjusts to screen size, making mobile-friendly design easier.
- It includes UI components like navigation bars, cards, forms, and buttons that keep a consistent appearance.
- Provides utility classes for spacing, sizing, flex layouts, and other common styling needs.
- Some Bootstrap components (such as dropdowns, carousels) include JavaScript
- While Bootstrap has a default style, it can be customized

### React:

A JavaScript library for building user interfaces, particularly single-page applications where UI updates are frequent.

- React organizes UIs into reusable, self-contained components that manage their own state.
- Creates a lightweight copy of the actual DOM in memory, which allows it to efficiently update only the parts of the actual DOM that need to change.
- React uses JSX, a syntax extension that lets you write HTML-like code within JavaScript
- Data flows down from parent components to children, making applications more predictable and easier to debug.

### Example

For this project, I created a chess game website featuring a modern, dark-themed UI that combines both Tailwind CSS and Bootstrap framework elements. The site includes: a navigation bar, A responsive two-column layout on larger screens (three column grid), an interactive chess board where users can play against computer moves (through drag and dropping), and game controls and statistics on the left sidebar. It implements bootstrap by:

- A responsive Bootstrap navbar with dropdown support
- Bootstrap cards with various styling
- Progress bars showing game progress
- Bootstrap buttons with different variants
- A responsive footer with Bootstrap grid
- Bootstrap icons integration

Here are some of the react components as well:

```
// The main App component
function App() {
  // Component code here
}

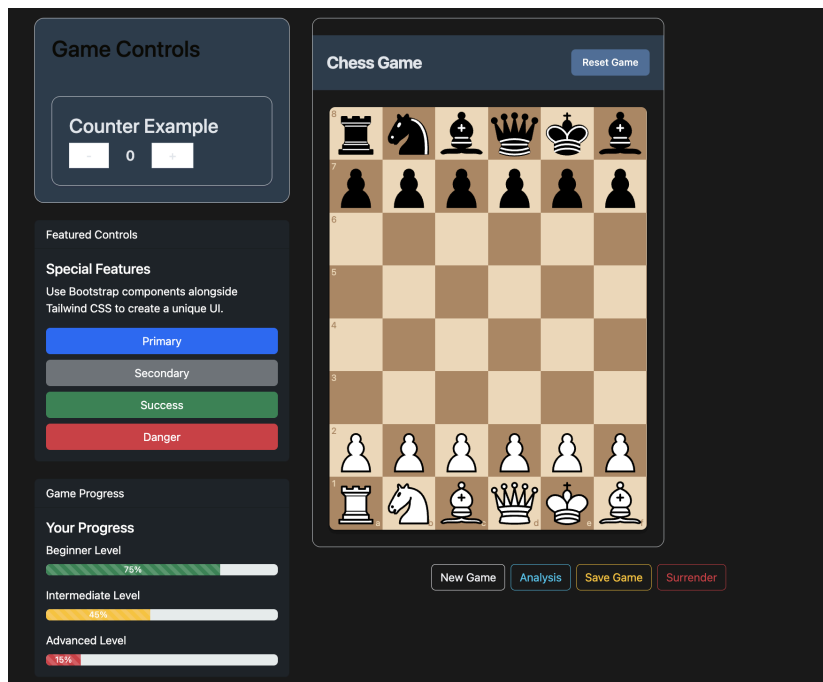
// Imported ModernChessGame component
<ModernChessGame width={600} />

// UI components from shadcn/ui
<Card className="p-6 bg-[#2c3e50] border-[#2d3748] mb-4">
  <Button variant="outline" onClick={() => setCount((count) => count - 1)}>

// Using useState hook to create and manage state
const [count, setCount] = useState(0)

// State update in event handlers
onClick={() => setCount((count) => count - 1)}
onClick={() => setCount((count) => count + 1)}
```

Below is an image of the website. I had trouble implementing a full chess board, but this chess board is still responsive and prompts a computer move after the user moves.



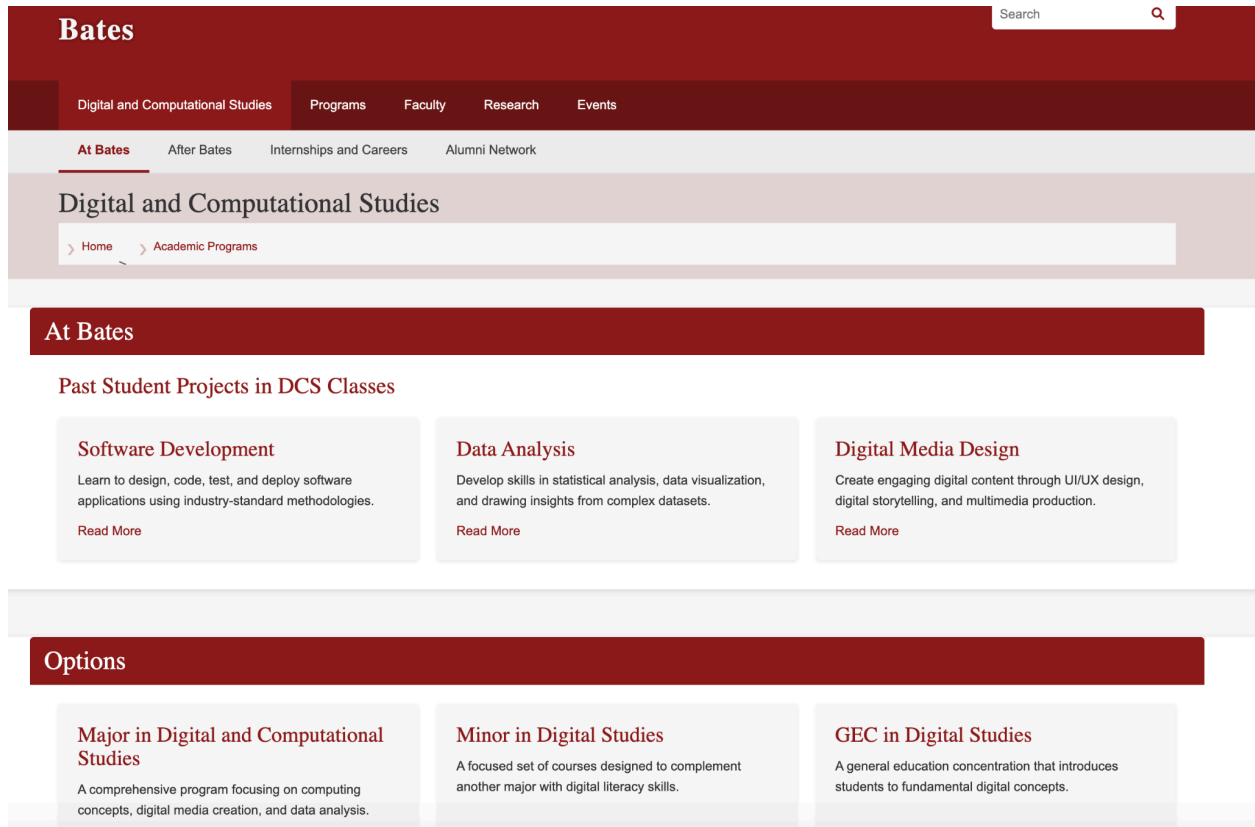
## **Design For User Experience (Krug)/Accessibility**

Krug's usability principles provide guidance for creating good, user friendly websites. These principles help designers and developers create interfaces that are intuitive and easy to navigate. Their role in web development includes:

- Simplifying user decision making, creating websites that require minimal cognitive effort, allowing users to navigate intuitively rather than consciously thinking through each step.
- It informs decisions about layout, navigation, content hierarchy, and visual elements to create websites that align with how users actually behave online.
- When websites follow these principles, users are more likely to have a positive experience, accomplish their goals, and return to the site.
- By reducing friction in the user journey, Krug's principles can help improve conversion rates for business websites.

Through understanding the impact of accessibility in creating a website, I was guided by some important features that ensure a website can be accessible. For example, for someone with visual impairments using a screen reader, properly labeled images and form fields mean the difference between being able to complete an online job application or not. About 15% of the world's population lives with some form of disability. This highlights the importance of also having the ability to navigate through a website without using the mouse. Features like keyboard navigation benefit not just people with motor disabilities but also help users who prefer keyboard shortcuts. Furthermore, the color scheme of a website has some important implications - people who may be colorblind might have a challenging time viewing specific colors or colors that crash. Thus, it is important to create a website that addresses such.

## Example:



With this homepage for my Bates DCS Website, I tried to follow Krug's suggestions as closely as possible. Highlighted here, you can see the simple and easy layout, where the user can see the title that directly lets them know where they are on the website, as well as provides breadcrumbs for them to understand how they got there. There are four tabs at the top that the user can choose from that clearly state where they would like to go. Additionally, each time you hover over a clickable button, the color darkens, indicating to the user that it is clickable. It allows the user to make minimal cognitive effort to navigate the site. Additionally, the layout is simple and structured, highlighting subtitles that show the user what subject they are looking at immediately.

Additionally, during the creation of this website, I made sure to keep in mind many of the accessibility implications as discussed above. This led me to create a website without crashing colors, allow for the functionality of the site through only using the keyboard, and have explicit, simple titles that clearly illustrate where the user is and what they are looking at.



## Prompt Responses:

1. What was your experience in using React+Vite for building web apps compared to "rolling your own" using HTML, CSS, & JavaScript?

My experience using react+Vite was great. I thought that it allowed for a much more customizable site, where I could create reusable components that made development more efficient. The component-based architecture of React made it easier to organize my code and maintain consistency across different parts of the application. With Vite, I could see changes in real-time without refreshing the page, which significantly accelerated my development workflow. Unlike rolling my own with HTML, CSS, and JavaScript, React+Vite eliminated much of the repetitive DOM manipulation and event handling code I would normally write. State management was more intuitive, and I didn't have to worry about manually updating different parts of the UI when data changed. There was definitely a learning curve to understand React, once I grasped these fundamentals, I found that I could build more complex features in less time than with JavaScript.

2. What are the primary takeaways you had from reading Krug's book and your corresponding analysis of sites?

To add to my discussion above regarding Krug, I would say that the main takeaways are: Usability trumps aesthetics - sites should be self-evident without requiring users to figure them out. Users scan, don't read - design for scanning with clear visual hierarchies and concise content. Minimize cognitive load - reduce the mental effort required to navigate and use a website. Test with real users early and often - even simple usability testing with a few people reveals important issues And lastly, eliminate unnecessary choices - fewer options lead to better user experiences and decisions.

3. What is the importance of accessibility (give a few explicit examples), and what steps can (and should) you take in assessing the accessibility of your site?

Also highlighted in my discussion above. I'd say that the importance of accessibility can be seen through having good heading structure and alt text for images which can allow blind users to navigate content effectively. Ensuring all interactive elements can be accessed without a mouse (helps users with motor impairments). Good contrast between text and background accommodates users with vision impairments. Allowing text to be enlarged without breaking layouts also helps users who need larger text. Lastly, providing text alternatives for audio and video content supports deaf and hard-of-hearing users. The steps for assessing the accessibility of a site are: Use automated tools websites that evaluate accessibility, checking color contrast, and testing with screen readers.

4. In what ways did the different design sprints, and use of Figma, help you in thinking about what an end product should look like and how it should function?

I thought figma was a great tool for visualizing what you want a site to look at before getting into the grunt work of your project. I allowed you to simply design the ideas you have and see how it would look on a website. Having the pages side by side allowed me to conceptualize how the website would flow from page to page, making it easy to implement my ideas when working with the code. Working on this in a group made the implementation aspect much easier, where we could also divide the sections through seeing them next to each other.

5. What takeaways do you have from working with AI/LLMs through Cursor (or similar) in building web applications?

I think that AI can act as a great tool for building websites. Through working with the cursor, I definitely noticed that it makes a lot of mistakes. It's hard to implement tailwind with cursor, as the different versions conflict with the AI model. It will never do exactly what you want it to do - through each prompt I have cursor, it addressed my question, but did not stylize it in the way I was visualizing. This led to me going back and manually changing many of the aspects it created. The best way to use AI to code, in my opinion, is by allowing it to help you with the initial structure of a site - it does a pretty good job at organizing/linking together files. It also is good at determining what the problem is in your code and highlighting suggestions to fix it. I don't think that at the current point, it can replace a human's need to understand the code - you still have to do a lot of work yourself to create a great website - but it definitely makes the process a bit faster. I also had a lot of trouble using it to connect to firebase - I had to do basically all of that manually.

6. What was your favorite thing (or deemed most useful) that we covered this semester, and why?

My favorite part of the semester was creating the final Bates DCS website - I thought it was great to conceptualize what we have learned in the class to create a website that would actually go towards a goal. Understanding the basics (HTML, CSS, Javascript) was great, but it was cool to see how you could go much further with it through using react. I also enjoyed how we got exposure to using AI as a tool, but also how it's still required to have a background in web development in order to implement AI. This was my first time learning how to navigate through the terminal, which is something I will definitely use in the future, as well as my first time using github, which will be super helpful for the rest of my time at Bates/career.

7. What do you wish we had covered, or had covered in more detail, and why?

I wish we had covered backend integration with Firebase a little more. While we gained a solid foundation in creating frontend interfaces, and had one project where we used firebase, I would have liked to go into this a little bit more. Understanding how to implement user authentication, manage server side state, and create full-stack applications would help bridge the gap between creating visually appealing pages and building fully functional web applications that can store and retrieve data.