CORFO
BecasCapital Humano

Adalid
aspasia
LA FORMACIÓN DE TU FUTURO

MÓDULO 2

DESARROLLO DE LA INTERFAZ DE USUARIO WEB

Manual del Módulo 2: Desarrollo de la Interfaz de Usuario Web

Introducción

El Módulo 2 es esencial para que los estudiantes desarrollen las competencias necesarias en la construcción de interfaces de usuario modernas, responsivas y estéticamente agradables. A través de este módulo, los participantes aprenderán a utilizar HTML5 y CSS3 de manera eficiente, apoyándose en frameworks de estilos y metodologías de desarrollo que permiten crear aplicaciones web escalables y mantenibles. También se explorarán conceptos avanzados como preprocesadores CSS, el protocolo HTTP y la seguridad web, lo cual es vital para el desarrollo front-end en un entorno profesional.

1. El Proceso de Desarrollo de un Producto Visual

1.1 Diferencias entre Front-End, Back-End y Full-Stack

- **Front-End**: Se refiere a la parte visual de la aplicación, con la que interactúan los usuarios. Implica el uso de tecnologías como HTML, CSS y JavaScript para construir interfaces que sean no solo funcionales, sino también atractivas y accesibles. El front-end se enfoca en la experiencia del usuario (UX) y la interfaz de usuario (UI).

- **Back-End**: Involucra la lógica del servidor, bases de datos y APIs. Es la parte del desarrollo que se ocupa de cómo funcionan las cosas detrás de la interfaz de usuario, gestionando datos, autenticación de usuarios, y las reglas de negocio de la aplicación.

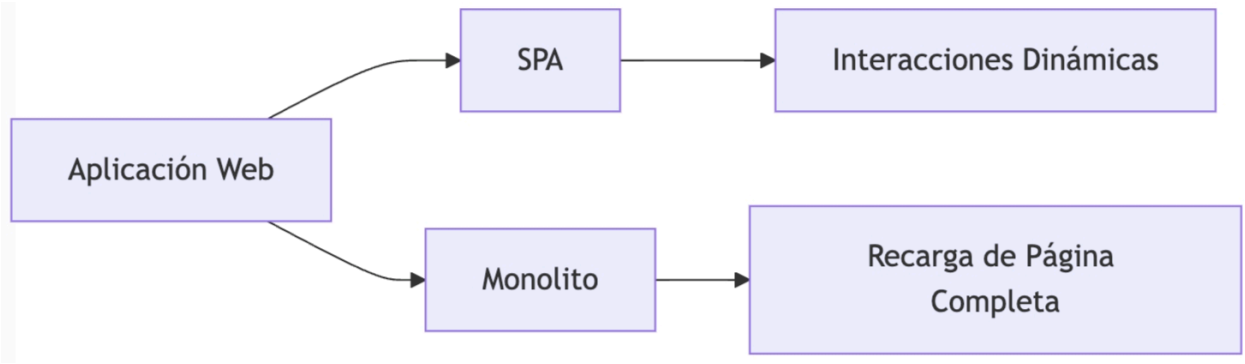
- **Full-Stack**: Un desarrollador full-stack tiene conocimientos tanto de front-end como de back-end. Este perfil permite crear aplicaciones completas, desde la interfaz de usuario hasta la lógica del servidor y la gestión de bases de datos.

Ejemplo Práctico: En una aplicación de comercio electrónico, el front-end maneja la presentación de productos, mientras que el back-end gestiona las transacciones y almacena la información del usuario en una base de datos. Un desarrollador full-stack podría encargarse de ambas partes.

1.2 Aplicaciones SPA vs. Monolitos

- **SPA (Single Page Application)**: Es un tipo de aplicación web que carga una única página HTML y actualiza dinámicamente el contenido según las interacciones del usuario, sin necesidad de recargar la página completa. Esto mejora la experiencia del usuario al ofrecer una navegación más rápida y fluida.

- **Monolitos**: Son aplicaciones tradicionales donde cada interacción del usuario con la web implica cargar una nueva página del servidor. Aunque menos eficiente que las SPA en términos de experiencia de usuario, los monolitos son más sencillos de implementar en términos de arquitectura.



Ejemplo Práctico: Gmail es un ejemplo de SPA, donde toda la aplicación funciona dentro de una sola página, mientras que un sitio web tradicional con varias páginas es un ejemplo de un monolito.

1.3 Renderización Cliente y Servidor

- **Renderización en el Servidor**: El servidor genera la página HTML completa y la envía al cliente (navegador). Es ideal para mejorar el SEO y la velocidad de carga inicial de la página.

- **Renderización en el Cliente**: El cliente (navegador) utiliza JavaScript para construir la página HTML después de recibir datos del servidor. Esto permite interacciones más dinámicas, aunque puede resultar en una carga inicial más lenta.

Ejemplo Real: Un sitio de noticias podría utilizar renderización del servidor para entregar rápidamente el contenido principal al usuario, mientras que las secciones interactivas, como los comentarios, podrían renderizarse en el cliente.

1.4 El Proceso de Ideación de un Producto Digital

- **Ideación**: Es la fase inicial donde se conceptualiza el producto, se identifican las necesidades del usuario y se define el problema que se va a resolver. Incluye actividades como brainstorming, creación de mapas de empatía y la definición de las principales funcionalidades del producto.

Ejemplo Práctico: Crear un mapa de empatía para una aplicación de gestión de tareas, identificando quiénes son los usuarios, qué necesidades tienen, y cómo la aplicación puede satisfacerlas.

1.5 El Proceso de Implementación de un Producto Digital

- **Implementación**: Se refiere a la fase donde las ideas se convierten en realidad. Incluye el desarrollo del producto utilizando tecnologías de front-end y back-end, pruebas de calidad, y despliegue. La implementación debe alinearse con los objetivos definidos durante la ideación.

Ejemplo Práctico: Desarrollar el prototipo de una aplicación web para gestionar tareas, utilizando HTML5 y CSS3 para la interfaz, y Node.js para el back-end.

1.6 El Rol del Diseñador UX/UI

- **Diseñador UX/UI**: Se enfoca en crear interfaces que sean intuitivas y fáciles de usar, y que proporcionen una experiencia positiva al usuario. El diseñador UX/UI trabaja en la estructura, diseño visual, y usabilidad del producto, asegurando que cada elemento cumpla una función y esté optimizado para la mejor experiencia posible.

Ejemplo Real: En una aplicación de banca móvil, el diseñador UX/UI sería responsable de que el proceso de transferencia de dinero sea fácil de entender y rápido de ejecutar.

1.7 El Rol del Desarrollador Front-End

- **Desarrollador Front-End**: Implementa las ideas del diseñador UX/UI utilizando HTML, CSS y JavaScript. Se asegura de que la interfaz funcione correctamente en diferentes navegadores y dispositivos, y que la experiencia del usuario sea fluida y sin problemas.

Ejemplo Práctico: Codificar una página de inicio basada en un diseño de UX/UI, utilizando un framework CSS como Bootstrap para garantizar la responsividad.

1.8 Importancia de las Guías de Estilos y Representaciones Visuales en la Maquetación

- **Guías de Estilos**: Son documentos que contienen directrices sobre el uso de colores, tipografías, iconografía, y componentes de interfaz en un producto digital. Son esenciales para mantener la coherencia visual y funcional en toda la aplicación.

- **Representaciones Visuales**: Incluyen wireframes, mockups y prototipos que ayudan a visualizar cómo se verá y funcionará la interfaz antes de comenzar a codificar.

Ejemplo Práctico: Crear una guía de estilos para una aplicación web, definiendo los colores principales, secundarios, tipografías y la estructura de los componentes.

2. El Lenguaje de Marcas

2.1 Qué es Lenguaje de Marcas

- **Lenguaje de Marcas**: Un lenguaje de marcas como HTML se utiliza para estructurar el contenido de una página web. Utiliza etiquetas para definir los distintos elementos del contenido, como encabezados, párrafos, imágenes y enlaces.

Ejemplo de Código:

```
<!DOCTYPE html>

<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Gestión de Tareas</title>
</head>
```

```
<body>
    <header>
        <h1>Mi Aplicación de Tareas</h1>
    </header>

    <section>
        <ul>
            <li>Comprar comestibles</li>
            <li>Estudiar HTML</li>
            <li>Desarrollar proyecto final</li>
        </ul>
    </section>

</body>
</html>
```

2.2 Qué es el DOM

- **DOM (Document Object Model)**: Es una representación en árbol de un documento HTML. Cada nodo en el árbol representa un elemento del documento. El DOM permite que los lenguajes de programación, como JavaScript, interactúen y manipulan la estructura, el contenido y el estilo de las páginas web.

Ejemplo Práctico: Utilizar JavaScript para cambiar el texto de un elemento en el DOM:

```
document.querySelector('h1').innerText = 'Nueva Tarea';
```

2.3 Acerca de HTML4 y Anteriores

- **HTML4**: Fue la versión predominante de HTML durante varios años antes de la llegada de HTML5. Aunque proporcionaba una estructura básica para las páginas web, carecía de elementos semánticos y de las funcionalidades avanzadas que ofrece HTML5.

Ejemplo: En HTML4, la estructura de una página web no diferenciaba claramente entre las diferentes secciones de contenido (como `header`, `nav`, `article`), lo que hacía más difícil para los motores de búsqueda y lectores de pantalla entender la organización del contenido.

2.4 HTML5 y sus Diferencias sobre HTML Tradicional

- **HTML5**: Introdujo nuevos elementos semánticos (`<header>`, `<footer>`, `<section>`, `<article>`) que mejoran la accesibilidad y SEO, soporte para video y audio sin plugins adicionales, y APIs nuevas como `localStorage` y `sessionStorage`.

Ejemplo de Código:

```
<article>
    <header>
        <h2>Título del Artículo</h2>
    </header>

    <p>Contenido del artículo aquí...</p>

    <footer>
        <p>Escrito por: Autor</p>
    </footer>
</article>
```

2.5 HTML5 en Dispositivos Móviles

- **HTML5 en Móviles**: Permite el desarrollo de aplicaciones web que son compatibles con dispositivos móviles gracias a su soporte para elementos responsivos y APIs que mejoran la experiencia móvil, como la geolocalización y el almacenamiento local.

Ejemplo Práctico: Crear una página web que se adapte a pantallas móviles utilizando etiquetas semánticas y el atributo `viewport` en el HTML.

2.6 Qué es XHTML y Cómo se Usa

- **XHTML (Extensible HyperText Markup Language)**: Es una reformulación de HTML como una aplicación XML. XHTML exige una sintaxis más estricta que HTML, como la

necesidad de cerrar todas las etiquetas y utilizar letras minúsculas para los nombres de las mismas.

Ejemplo de Código:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Mi Página XHTML</title>
</head>

<body>
    <p>Este es un documento XHTML</p>
</body>
</html>
```

3. Metodologías para la Organización y Modularización de Estilos

3.1 Ventajas de Utilizar una Metodología

- **Organización y Modularización**: Usar metodologías en CSS como BEM o SMACSS facilita la organización del código, lo que resulta en un mantenimiento más sencillo y la posibilidad de escalar proyectos sin que el código CSS se vuelva inmanejable.

Ejemplo Práctico: Estructurar un proyecto utilizando BEM para garantizar la modularidad de los estilos:

```
<div class="menu">
    <ul class="menu__list">
        <li class="menu__item menu__item--active">Inicio</li>
        <li class="menu__item">Acerca de</li>
```

```
        <li class="menu__item">Contacto</li>
    </ul>
</div>
```

3.2 Metodologías más Utilizadas (BEM, OOCSS, SMACSS)

- **BEM (Block, Element, Modifier)**: Se enfoca en la creación de componentes reutilizables y evita la duplicación de código. Define un sistema claro de nombres para las clases en CSS.

- **OOCSS (Object-Oriented CSS)**: Separa la estructura del contenido, permitiendo que los estilos sean más reutilizables.

- **SMACSS (Scalable and Modular Architecture for CSS)**: Proporciona una metodología flexible para el desarrollo de grandes proyectos de CSS, centrada en la modularización y escalabilidad.

Ejercicio Práctico: Convertir un diseño simple en una estructura BEM para practicar la modularización de estilos.

4. Preprocesadores CSS

4.1 Qué es un Preprocesador CSS

- **Preprocesadores CSS**: Como Sass o LESS, permiten escribir CSS de manera más eficiente y estructurada, utilizando características avanzadas como variables, anidación, mixins y funciones. Facilitan el mantenimiento del código CSS y mejoran su reutilización.

Ejemplo de Código con Sass:

```
$primary-color: #3498db;
```

```
body {
    font-family: 'Arial', sans-serif;
    background-color: $primary-color;
}

.button {
    background-color: darken($primary-color, 10%);
    padding: 10px 20px;
    border-radius: 5px;

    &:hover {
        background-color: lighten($primary-color, 10%);
    }
}
```

4.2 Importancia del Preprocesador en el Desarrollo Front-End

- **Ventajas**: Los preprocesadores CSS como Sass permiten estructurar el código de manera modular, reutilizar componentes y aplicar estilos de manera más eficiente, lo que es crucial para el desarrollo de proyectos grandes y complejos.

Ejercicio Práctico: Crear un archivo Sass para un proyecto de página web, utilizando variables y mixins para estandarizar los estilos.

4.3 Preprocesadores más Utilizados (SASS, LESS)

- **SASS**: Ofrece características avanzadas como la anidación, mixins, herencia, y más. Es uno de los preprocesadores más utilizados en la industria.

- **LESS**: Similar a Sass, pero con una sintaxis más ligera. LESS es especialmente popular en proyectos que utilizan frameworks como Bootstrap.

Ejercicio Práctico: Implementar un pequeño proyecto utilizando Sass para experimentar con sus características avanzadas.

5. Conceptos Avanzados del Desarrollo Web

5.1 Protocolo HTTP

- **Qué es el Protocolo HTTP**: HTTP (HyperText Transfer Protocol) es el protocolo utilizado para la transmisión de datos en la web. Permite que los navegadores soliciten y reciban contenido desde los servidores web.

- **Dónde se Utiliza el Protocolo HTTP**: HTTP es la base de la comunicación en la web, utilizado en todas las interacciones entre el cliente y el servidor para la carga de páginas, imágenes, scripts y más.

- **Entendiendo los Métodos y sus Usos**:
  - **GET**: Solicita datos del servidor (por ejemplo, para cargar una página web).
  - **POST**: Envía datos al servidor (por ejemplo, al enviar un formulario).
  - **PUT**: Actualiza recursos en el servidor.
  - **DELETE**: Elimina recursos en el servidor.

- **Qué se Entiende por Endpoint**: Un endpoint es una URL específica en la API de un servidor que maneja solicitudes HTTP.

- **Interpretando los Códigos de Retorno**:
  - **200 OK**: Solicitud exitosa.
  - **404 Not Found**: El recurso solicitado no se encuentra.
  - **500 Internal Server Error**: Error en el servidor.

Ejemplo Práctico: Utilizar `fetch` en JavaScript para realizar una solicitud GET y procesar la respuesta:

```
fetch('https://api.example.com/data')
    .then(response => response.json())
    .then(data => console.log(data))
    .catch(error => console.error('Error:', error));
```

5.2 Qué es un CDN y Cómo se Usa

- **CDN (Content Delivery Network)**: Es una red de servidores distribuidos que entregan contenido a los usuarios según su ubicación geográfica, lo que reduce el tiempo de carga de las páginas web.

Ejemplo Práctico: Utilizar un CDN para cargar una librería JavaScript como jQuery desde una ubicación cercana al usuario:

```
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
```

5.3 Implementando la Seguridad con HTTPS

- **HTTPS**: Es la versión segura de HTTP, que utiliza SSL/TLS para encriptar la comunicación entre el cliente y el servidor, protegiendo la integridad y confidencialidad de los datos.

Ejemplo Real: Configurar un sitio web para que utilice HTTPS en lugar de HTTP, asegurando que todas las comunicaciones sean seguras.

5.4 Entendiendo el Encoding de Páginas

- **Encoding**: Se refiere a la codificación de caracteres en una página web. UTF-8 es el encoding más común, permitiendo que se representen una amplia gama de caracteres de diferentes lenguas.

Ejemplo de Código:

```
<meta charset="UTF-8">
```

5.5 Acerca de Cookies y su Uso en los Navegadores

- **Cookies**: Son pequeños archivos almacenados en el navegador del usuario que contienen datos específicos de una página web, como preferencias del usuario o información de inicio de sesión.

Ejemplo Práctico: Crear una cookie con JavaScript:

```
document.cookie = "username=JohnDoe; expires=Fri, 31 Dec 2024 12:00:00 UTC; path=/";
```

5.6 Compatibilidad del Desarrollo Web en los Distintos Navegadores

- **Compatibilidad entre Navegadores**: Asegurar que una página web funcione correctamente en todos los navegadores principales es crucial. Esto implica probar la aplicación en diferentes entornos y utilizar técnicas como polyfills para garantizar el soporte en navegadores más antiguos.

Ejemplo Práctico: Utilizar Autoprefixer para añadir prefijos CSS automáticamente y garantizar la compatibilidad entre navegadores.

5.7 Entendiendo el LocalStorage y SessionStorage como Almacenaje Local de los Datos en un Navegador

- **LocalStorage y SessionStorage**: Son APIs de almacenamiento local que permiten a las aplicaciones web guardar datos en el navegador del usuario. `localStorage` almacena datos sin fecha de expiración, mientras que `sessionStorage` almacena datos solo durante la sesión del navegador.

Ejemplo Práctico: Guardar y recuperar datos usando `localStorage`:

```
localStorage.setItem('username', 'JohnDoe');

let user = localStorage.getItem('username');

console.log(user); // Output: JohnDoe
```

5.8 Aplicaciones de Consola: Otra Forma de Consumir Aplicaciones Web (Lynx, Wget, Curl)

- **Aplicaciones de Consola**: Herramientas como Lynx, Wget y Curl permiten acceder a páginas web y consumir APIs desde la línea de comandos, lo cual es útil para desarrolladores que necesitan automatizar solicitudes o trabajar en entornos sin interfaz gráfica.

Ejemplo Práctico: Usar `curl` para hacer una solicitud GET a una API:

```
curl https://api.example.com/data
```

6. Hojas de Estilo

6.1 Acerca de CSS y su Uso en la Interfaz de Usuario Web

- **CSS (Cascading Style Sheets)**: Es el lenguaje utilizado para definir la presentación de un documento HTML, controlando aspectos como colores, fuentes, y layout. CSS permite separar la estructura del contenido (HTML) de su presentación, facilitando la creación de interfaces de usuario atractivas y consistentes.

Ejemplo de Código:

```css
body {
    font-family: 'Open Sans', sans-serif;
    background-color: #f4f4f4;
}

h1 {
    color: #333;
    text-align: center;
}
```

6.2 Ventajas de Usar CSS

- **Ventajas de CSS**: CSS permite aplicar estilos de manera consistente en todo el sitio web, reduce la redundancia en el código, y facilita la separación de preocupaciones al permitir que el HTML se enfoque en la estructura y el CSS en la presentación.

Ejemplo Práctico: Usar una hoja de estilo externa para aplicar estilos consistentes en varias páginas de un sitio web.

6.3 Desde CSS1 hasta CSS3

- **CSS1**: Introdujo las bases de la estilización en la web.

- **CSS2**: Añadió características como el soporte para diferentes tipos de medios (pantallas, impresoras) y la capacidad de definir layouts más complejos.

- **CSS3**: Introdujo nuevas características como transiciones, transformaciones, animaciones, y soporte para flexbox y grid layout, que permiten diseñar interfaces de usuario más complejas y dinámicas.

Ejemplo Práctico: Crear una animación simple en CSS3:

```
.button {
    background-color: #3498db;
    transition: background-color 0.3s ease;
}

.button:hover {
    background-color: #2980b9;
}
```

6.4 Aspectos Avanzados de CSS

- **Patrones**: Técnicas repetibles que pueden ser aplicadas en el diseño web para resolver problemas comunes de presentación.

- **Bloques de Declaración**: Conjunto de reglas CSS que se aplican a un selector particular.

- **Uso de Estilos**: Estrategias para aplicar CSS a elementos HTML de manera eficiente y organizada.

- **Fuentes**: Controlar la tipografía es esencial para la experiencia de usuario. CSS permite utilizar fuentes web, controlar el tamaño, peso, y estilo de las fuentes.

- **Especificidad**: Define qué estilos se aplican cuando múltiples selectores coinciden con un elemento. La especificidad se calcula en función de los tipos de selectores utilizados.

- **Herencia**: Algunos estilos se heredan de elementos padres a hijos en el DOM, lo que puede simplificar la aplicación de estilos.

- **Posicionamiento**: CSS ofrece varias formas de posicionar elementos en la página, como `static`, `relative`, `absolute`, `fixed`, y `sticky`.

Ejemplo Práctico: Utilizar flexbox para centrar un elemento en la página:

```
.container {
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
}
```

6.5 Limitaciones de CSS

- **Limitaciones**: Aunque poderoso, CSS tiene limitaciones, como la falta de lógica condicional, lo que a menudo requiere soluciones creativas o el uso de preprocesadores para superar estos desafíos.

6.6 Soporte en Distintos Navegadores Web

- **Soporte CSS**: No todos los navegadores implementan las propiedades CSS de la misma manera. Es importante probar los diseños en varios navegadores y utilizar herramientas como Autoprefixer para añadir los prefijos necesarios.

Ejemplo Práctico: Añadir prefijos para asegurar la compatibilidad:

```
.container {
    display: -webkit-flex;
    display: -ms-flexbox;
    display: flex;
}
```

6.7 CSS4 y el Futuro de las Hojas de Estilo

- **CSS4**: Aunque CSS4 no es una especificación oficial, el término se utiliza para describir la evolución continua de CSS con nuevas funcionalidades y características que seguirán mejorando la capacidad de diseñar interfaces web más complejas.

7. Responsividad

7.1 Qué es Responsividad

- **Responsividad**: Es la capacidad de un sitio web para adaptarse a diferentes tamaños de pantalla y dispositivos, ofreciendo una experiencia de usuario consistente sin importar si se accede desde un ordenador de escritorio, una tableta, o un teléfono móvil.

Ejemplo Práctico: Crear un layout responsivo utilizando media queries:

```
@media (max-width: 768px) {
    .container {
        flex-direction: column;
    }
```

```
}
```

7.2 Por Qué Desarrollar Aplicaciones Web Responsivas

- **Importancia**: Con el aumento del uso de dispositivos móviles, es crucial que las aplicaciones web sean responsivas para garantizar que todos los usuarios tengan una experiencia óptima sin importar el dispositivo que utilicen.

Ejemplo Práctico: Rediseñar un sitio web existente para hacerlo responsivo utilizando Bootstrap.

7.3 Impacto de la Responsividad en la Experiencia de Usuario

- **Impacto**: Un sitio web responsivo mejora la experiencia del usuario al facilitar la navegación y el acceso a la información sin la necesidad de hacer zoom o desplazarse horizontalmente.

7.4 Cómo Funciona la Adaptabilidad a Distintos Anchos de Pantalla

- **Adaptabilidad**: Se logra utilizando media queries y unidades flexibles como `%`, `vw`, `vh`, y `rem` para ajustar el layout y los elementos de la página a diferentes tamaños de pantalla.

Ejemplo Práctico: Implementar un diseño fluid grid que se ajuste a cualquier tamaño de pantalla.

7.5 Diseño Web Responsivo vs. Diseño Adaptativo

- **Diseño Responsivo**: Se adapta automáticamente a cualquier tamaño de pantalla utilizando un único diseño flexible que cambia según el tamaño del viewport.

- **Diseño Adaptativo**: Utiliza diferentes layouts predefinidos que se cargan según el tamaño de pantalla del dispositivo.

Ejemplo Práctico: Comparar los resultados de un diseño responsivo y un diseño adaptativo en una variedad de dispositivos.

7.6 Cómo Crear Aplicaciones Web Responsivas

- **Técnicas**: Utilizar media queries, flexbox, grid layout, y unidades relativas para crear layouts que se adapten a diferentes tamaños de pantalla.

7.7 Los Bloques de Construcción del Diseño Web Responsivo

- **Bloques**: Incluyen el uso de fluid grids, flexible images, y media queries para crear un diseño responsivo eficaz.

7.8 Puntos de Ruptura Responsivos Comunes

- **Puntos de Ruptura**: Son los anchos de pantalla en los cuales el layout del sitio web cambia para mejorar la legibilidad y la usabilidad en diferentes dispositivos.

Ejemplo Práctico: Definir puntos de ruptura para un sitio web utilizando Bootstrap.

7.9 Unidades y Valores del CSS para el Diseño Responsivo

- **Unidades Responsivas**: Utilizar unidades como `%`, `vw`, `vh`, y `rem` permite crear layouts más flexibles y adaptables a diferentes tamaños de pantalla.

Ejemplo Práctico: Usar `vw` y `vh` para definir un contenedor que ocupe un porcentaje de la ventana gráfica:

```
.container {
    width: 80vw;
    height: 50vh;
}
```

8. Modelo de Cajas

8.1 Qué es el Modelo de Cajas. ¿Existen otros modelos?

- **Modelo de Cajas**: Es el concepto básico de CSS que describe cómo se estructura el contenido en una página web. Cada elemento HTML se representa como una caja que tiene márgenes, bordes, rellenos, y el contenido.

Ejemplo de Código:

```
.box {
    width: 300px;
    padding: 20px;
    border: 5px solid black;
    margin: 15px;
}
```

8.2 Propiedades que Componen el Modelo de Cajas

- **Propiedades**: Incluyen `margin`, `border`, `padding`, y `content`. Estas propiedades controlan el espaciado y el tamaño de los elementos en una página web.

8.3 Tipos de Cajas: Diferencias entre Elementos de Bloque y Elementos de Línea

- **Bloques**: Elementos que ocupan todo el ancho disponible y comienzan en una nueva línea (`div`, `h1`, `p`).

- **Línea**: Elementos que ocupan solo el ancho necesario y no obligan a un salto de línea (`span`, `a`, `img`).

8.4 Inspeccionando Elementos con el Navegador para Identificar las Cajas

- **Herramientas de Desarrollo**: Los navegadores modernos tienen herramientas de desarrollo que permiten inspeccionar y modificar el modelo de cajas de cualquier elemento en tiempo real.

Ejemplo Práctico: Utilizar las herramientas de desarrollo de Chrome para inspeccionar y modificar el modelo de cajas de un elemento en una página web.

9. Posicionamiento de Elementos y Visualización

9.1 Conceptos Básicos de Propiedades de Posicionamiento

- **Posicionamiento**: CSS permite posicionar elementos en la página utilizando diferentes técnicas: `static`, `relative`, `absolute`, `fixed`, y `sticky`.

Ejemplo Práctico: Crear un menú de navegación fijo en la parte superior de la pantalla usando `position: fixed`.

9.2 Tipos de Posicionamiento (Normal, Relativo, Absoluto, Fijo, Flotante)

- **Normal**: El flujo estándar de documentos en HTML.
- **Relativo**: Posiciona el elemento en relación con su posición normal.
- **Absoluto**: Posiciona el elemento en relación con su contenedor más cercano que tenga un `position: relative`.
- **Fijo**: Fija el elemento en una posición específica en la ventana gráfica.
- **Flotante**: Utiliza la propiedad `float` para permitir que los elementos floten a la izquierda o derecha de su contenedor.

Ejemplo Práctico: Crear una tarjeta de información que flote a la derecha de un contenedor utilizando `float`.

9.3 Visualización

- **Visualización**: CSS controla cómo se muestran los elementos en la página, con propiedades como `display`, `visibility`, y `opacity`. Estas propiedades permiten ocultar elementos, cambiar su visibilidad, y modificar cómo se distribuyen en el layout.

Ejemplo Práctico: Usar `display: none` para ocultar un elemento en un tamaño de pantalla específico y `display: block` para mostrarlo nuevamente en un tamaño mayor.

10. Layout

10.1 Qué es un Layout

- **Layout**: Es la disposición y estructura de los elementos en una página web. El layout determina cómo se presentan los contenidos, controlando su tamaño, posición, y relación entre sí.

10.2 Tipos de Layout (Fluido, Fijo, Elástico, Absoluto)

- **Fluido**: Los elementos del layout se redimensionan proporcionalmente según el tamaño de la ventana gráfica.

- **Fijo**: Los elementos del layout tienen un tamaño fijo y no cambian con el tamaño de la ventana gráfica.

- **Elástico**: Los elementos cambian su tamaño en función de una unidad relativa como `em` o `rem`.

- **Absoluto**: Los elementos se posicionan de manera fija y no dependen del flujo normal del documento.

Ejemplo Práctico: Crear un layout fluido utilizando unidades porcentuales para los anchos de los elementos.

10.3 Ventajas y Desventajas

- **Ventajas y Desventajas**: Cada tipo de layout tiene sus propias ventajas y desventajas en términos de flexibilidad, mantenibilidad, y compatibilidad. Es importante seleccionar el tipo de layout adecuado según el proyecto y los requisitos de diseño.

11. El Preprocesador SASS

11.1 Qué es Sass y Por Qué Utilizarlo

- **Sass**: Es un preprocesador CSS que extiende las capacidades de CSS con características como variables, anidación, mixins, herencia, y más. Sass facilita la escritura de CSS más limpio, mantenible y modular.

Ejemplo Práctico: Definir una variable en Sass y usarla para cambiar el color de fondo de varios elementos:

```
$bg-color: #f4f4f4;

body {
    background-color: $bg-color;
}

.header {
    background-color: darken($bg-color, 10%);
}
```

11.2 Conocer Sass y Aprovechar las Ventajas en el Proceso de Construcción de un Sitio Web

- **Ventajas**: Sass permite organizar el CSS en múltiples archivos parciales, que luego se combinan en un solo archivo CSS. Esto mejora la organización y el mantenimiento del código.

11.3 Instalar Sass y Linters para Editores de Código

- **Instalación de Sass**: Se puede instalar Sass mediante Node.js o usando la versión de Ruby. Además, se recomienda configurar linters para asegurarse de que el código Sass sigue las mejores prácticas.

Ejemplo Práctico: Instalar Sass usando npm:

```
npm install -g sass
```

11.4 Conocer Patrón 7-1

- **Patrón 7-1**: Es una convención para organizar los archivos de Sass en siete carpetas (base, components, layout, pages, themes, abstracts, vendors) más un archivo principal que importa todos estos módulos.

Ejemplo de Estructura:

```
styles/
├── abstracts/
├── base/
├── components/
├── layout/
├── pages/
├── themes/
├── vendors/
```

```
└── main.scss
```

11.5 Sintaxis, Flujo de Trabajo y Buenas Prácticas Usando Sass

- **Sintaxis**: Sass admite dos sintaxis: `.scss` (similar a CSS) y `.sass` (una sintaxis más concisa). La mayoría de los desarrolladores prefieren `.scss` por su similitud con CSS.

- **Flujo de Trabajo**: Utilizar Sass en combinación con herramientas como Gulp o Webpack para compilar automáticamente el CSS cada vez que se guarda un archivo.

- **Buenas Prácticas**: Es importante seguir buenas prácticas como usar variables para colores y tamaños, modularizar el código y evitar la anidación excesiva.

Ejemplo Práctico: Configurar un flujo de trabajo básico con Gulp para compilar Sass:

```
const gulp = require('gulp');
const sass = require('gulp-sass')(require('sass'));

gulp.task('sass', function() {
    return gulp.src('styles/**/*.scss')
        .pipe(sass().on('error', sass.logError))
        .pipe(gulp.dest('./css'));
});

gulp.task('watch', function() {
    gulp.watch('styles/**/*.scss', gulp.series('sass'));
});
```

11.6 Utilización de Sass desde Línea de Comandos

- **Línea de Comandos**: Sass puede compilarse directamente desde la línea de comandos usando el comando `sass`.

Ejemplo Práctico: Compilar un archivo `.scss` a `.css`:

```
sass styles/main.scss css/main.css
```

11.7 Instalación de Plugins de Sass en Editores de Texto para Automatización de Tareas

- **Plugins**: Editores como Visual Studio Code tienen extensiones que permiten la compilación automática de Sass y el linting para garantizar que el código siga las mejores prácticas.

11.8 Uso de Variables para Reutilización de Código

- **Variables**: Permiten definir valores reutilizables como colores, fuentes, y tamaños, facilitando cambios globales en el proyecto.

Ejemplo Práctico:

```
$primary-color: #3498db;
$font-stack: 'Helvetica, sans-serif';

body {
    color: $primary-color;
    font-family: $font-stack;
}
```

11.9 Elementos Anidados y Namespaces

- **Anidación**: Sass permite anidar selectores dentro de otros selectores, lo que facilita la organización del código CSS.

- **Namespaces**: Se utilizan para agrupar estilos relacionados y evitar conflictos de nombres.

Ejemplo Práctico:

```
nav {
    ul {
        list-style: none;
    }

    li {
        display: inline-block;
        margin-right: 10px;
    }
}
```

11.10 Manejo de Parciales e Imports

- **Parciales**: Son archivos Sass que contienen partes del código CSS y que se importan en el archivo principal para compilarse juntos.

- **Imports**: La directiva `@import` se utiliza para incluir parciales en el archivo principal.

Ejemplo Práctico: Crear un archivo parcial `_buttons.scss` e importarlo en `main.scss`:

```
// _buttons.scss

.button {
    padding: 10px 20px;
    background-color: $primary-color;
}
```

```
// main.scss

@import 'buttons';
```

11.11 Manejo de Mixins e Includes

- **Mixins**: Permiten definir bloques de código CSS que se pueden reutilizar en diferentes partes del proyecto.

- **Includes**: La directiva `@include` se utiliza para aplicar un mixin a un selector.

Ejemplo Práctico: Crear un mixin para aplicar bordes redondeados a un elemento:

```
@mixin rounded($radius) {
    border-radius: $radius;
}

.box {
    @include rounded(10px);
}
```

12. Modularización de Estilos

12.1 Por Qué Modularizar

- **Modularización**: Organizar el CSS en módulos ayuda a mantener el código limpio, fácil de mantener, y escalable. Permite trabajar en proyectos grandes sin que el CSS se vuelva inmanejable.

12.2 Ventajas de Modularizar los Estilos

- **Ventajas**: Incluyen mejor mantenimiento, reutilización de código, y colaboración más eficiente en equipos grandes.

12.3 Utilización de Metodologías de Modularización

- **Metodologías**: Existen diversas metodologías como BEM, OOCSS, y SMACSS que ayudan a modularizar y organizar el código CSS.

13. Metodologías de Modularización

13.1 Qué es una Metodología de Modularización

- **Modularización**: Es una estrategia para dividir el código CSS en módulos reutilizables y manejables. Esto permite que los desarrolladores trabajen en diferentes partes de un proyecto sin causar conflictos.

13.2 Para Qué Sirve una Metodología

- **Utilidad**: Las metodologías de modularización sirven para organizar el código de manera que sea fácil de mantener, escalar y colaborar en proyectos de gran envergadura.

13.3 Ventajas de Usar una Metodología

- **Ventajas**: Incluyen una estructura clara, menos conflictos de nombres, y un código más mantenible y escalable.

13.4 Identificando Algunas Metodologías y sus Características (BEM, OOCSS, SMACSS, ITCSS)

- **BEM**: Se enfoca en la creación de componentes reutilizables con una convención de nombres clara.

- **OOCSS**: Promueve la separación de estructura y estilo.

- **SMACSS**: Se centra en la modularización y escalabilidad del CSS.

- **ITCSS**: Proporciona un enfoque escalonado para la organización del código CSS, optimizando la carga de los estilos.

14. Metodología BEM

14.1 En Qué Consiste Esta Metodología

- **BEM**: Significa Block, Element, Modifier. Es una metodología que facilita la creación de componentes reutilizables y un código CSS más limpio y mantenible.

14.2 Conceptos Claves de BEM

- **Block**: Es el contenedor principal de un componente.
- **Element**: Son las partes individuales de un bloque.
- **Modifier**: Son variaciones o estados del bloque o sus elementos.

14.3 Convención de Nombres

- **Nombres en BEM**: Utiliza una convención de nombres que facilita la lectura y mantenimiento del código.

Ejemplo Práctico:

```
<div class="menu">
    <ul class="menu__list">
        <li class="menu__item menu__item--active">Inicio</li>
        <li class="menu__item">Acerca de</li>
        <li class="menu__item">Contacto</li>
    </ul>
</div>
```

14.4 Cómo Estructurar un Proyecto con BEM

- **Estructura**: Organizar los archivos CSS siguiendo la convención de BEM facilita la escalabilidad y el mantenimiento del proyecto.

Ejemplo de Estructura de Archivos:

```
styles/
├── blocks/
│   ├── _menu.scss
│   ├── _button.scss
│   └── _header.scss
└── main.scss
```

15. Metodología OOCSS

15.1 En Qué Consiste Esta Metodología

- **OOCSS**: Object-Oriented CSS. Promueve la reutilización de código CSS mediante la separación de la estructura y la apariencia, enfocándose en la creación de componentes modulares.

15.2 Principios Básicos de OOCSS

- **Separación de Estructura y Estilo**: Mantener la estructura y los estilos separados permite que los componentes sean más flexibles y reutilizables.

Ejemplo Práctico:

```css
/* Estructura */

.box {
    padding: 20px;
    border: 1px solid #ccc;
}

/* Estilo */
```

```css
.theme-light .box {
    background-color: #fff;
    color: #333;
}

.theme-dark .box {
    background-color: #333;
    color: #fff;
}
```

16. Metodología SMACSS

16.1 En Qué Consiste Esta Metodología

- **SMACSS**: Scalable and Modular Architecture for CSS. Es una metodología que organiza el CSS en categorías para mejorar la modularización y escalabilidad.

16.2 Categorías de Elementos SMACSS

- **Base**: Contiene estilos globales y reset.
- **Layout**: Estructura básica de la página.
- **Module**: Componentes reutilizables.
- **State**: Modificaciones o variaciones de los módulos.
- **Theme**: Estilos específicos para diferentes temas o configuraciones visuales.

Ejemplo de Estructura SMACSS:

```
styles/
├── base/
├── layout/
├── modules/
├── state/
```

```
└── theme/
```

17. Frameworks CSS

17.1 Qué es un Framework CSS, Por Qué y Cuándo Utilizarlo

- **Framework CSS**: Un framework CSS proporciona un conjunto predefinido de estilos y componentes que facilitan la creación de interfaces de usuario consistentes y responsivas.

17.2 Alternativas de Frameworks Basados en CSS (Bootstrap, Foundation, Pure CSS, Bulma)

- **Bootstrap**: El framework CSS más popular, ofrece un amplio conjunto de componentes responsivos.
- **Foundation**: Similar a Bootstrap, pero con un enfoque en la accesibilidad.
- **Pure CSS**: Un framework ligero que proporciona solo lo esencial.
- **Bulma**: Un framework moderno y modular basado en flexbox.

17.3 Ventajas de Usar un Framework CSS vs. HTML Tradicional

- **Ventajas**: Los frameworks CSS aceleran el desarrollo al proporcionar componentes predefinidos y estilos consistentes, lo que reduce la necesidad de escribir código CSS personalizado desde cero.

17.4 Componiendo Estilos Propios

- **Personalización**: Aunque los frameworks CSS son útiles, es importante saber cómo personalizarlos y escribir estilos propios para que se ajusten a las necesidades específicas del proyecto.

Ejemplo Práctico: Personalizar un botón de Bootstrap utilizando variables Sass:

```
$btn-primary-bg: #5a67d8;
$btn-primary-border: #4c51bf;
@import 'bootstrap/scss/bootstrap';
```

18. Analizando un Framework CSS: Bootstrap

18.1 Qué es Bootstrap

- **Bootstrap**: Es un framework front-end que facilita la creación de sitios web y aplicaciones web responsivas. Proporciona una gran cantidad de componentes preconstruidos y una grilla flexible para el diseño del layout.

18.2 Ventajas

- **Ventajas de Bootstrap**: Bootstrap permite un desarrollo rápido, es altamente personalizable, y asegura la compatibilidad entre navegadores.

18.3 Filosofía Bootstrap

- **Responsividad**: Bootstrap está diseñado para ser responsivo desde el principio, utilizando una grilla que se adapta a diferentes tamaños de pantalla.

- **Principio Mobile First**: Bootstrap prioriza el diseño móvil, asegurando que las aplicaciones sean accesibles y funcionales en dispositivos móviles antes de escalar a pantallas más grandes.

Ejemplo Práctico: Crear un layout básico en Bootstrap utilizando la grilla responsiva:

```
<div class="container">
    <div class="row">
        <div class="col-sm-12 col-md-6">Columna 1</div>
        <div class="col-sm-12 col-md-6">Columna 2</div>
    </div>
</div>
```

18.4 Modificar y Extender Funcionalidad de Bootstrap con Sass

- **Extensión con Sass**: Bootstrap es altamente personalizable utilizando variables Sass. Esto permite modificar la apariencia de los componentes predeterminados sin necesidad de sobrescribir CSS.

18.5 Conociendo Layouts, Contenidos y Componentes de Bootstrap

- **Grillas**: El sistema de grillas de Bootstrap permite organizar el contenido de manera flexible y responsiva.

- **Botones**: Los botones en Bootstrap son altamente personalizables y vienen en varios estilos predefinidos.

- **Paneles, Tablas, Formularios, Navegación y Menús**: Bootstrap proporciona una amplia gama de componentes preconstruidos que facilitan la creación de interfaces consistentes y funcionales.

Ejemplo Práctico: Crear un formulario utilizando Bootstrap:

```
<form>
    <div class="form-group">
        <label for="email">Correo Electrónico</label>
        <input type="email" class="form-control" id="email"
placeholder="Ingrese su correo">
```

```
    </div>

    <button type="submit" class="btn btn-primary">Enviar</button>
</form>
```

18.6 Conociendo Componentes JavaScript

- **Alertas**: Utilizadas para mostrar mensajes de advertencia, información o éxito.
- **Breadcrumb**: Muestra la ubicación del usuario dentro de la jerarquía de la aplicación.
- **Card**: Un contenedor flexible que agrupa contenido y acciones sobre un único tema.
- **Carrusel**: Un componente que permite mostrar una serie de imágenes o contenido en un espacio reducido.
- **Tooltip**: Proporciona información adicional cuando el usuario pasa el cursor sobre un elemento.
- **Modal**: Un cuadro de diálogo que aparece sobre el contenido principal, ideal para formularios, alertas o confirmaciones.
- **Popover**: Similar al tooltip, pero más flexible y con más opciones de contenido.

Ejemplo Práctico: Implementar un modal en Bootstrap:

```
<button type="button" class="btn btn-primary" data-toggle="modal"
data-target="#myModal">Abrir Modal</button>

<div class="modal fade" id="myModal" tabindex="-1"
aria-labelledby="exampleModalLabel" aria-hidden="true">

    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title" id="exampleModalLabel">Título del
Modal</h5>
```

```
                <button type="button"

 class="close" data-dismiss="modal" aria-label="Close">
                    <span aria-hidden="true">&times;</span>
                </button>
            </div>

            <div class="modal-body">
                Contenido del modal...
            </div>

            <div class="modal-footer">
                <button type="button" class="btn btn-secondary"
data-dismiss="modal">Cerrar</button>
                <button type="button" class="btn btn-primary">Guardar
cambios</button>
            </div>
        </div>
    </div>
</div>
```

Material de Referencia

Libros:

- **"HTML & CSS: Design and Build Websites"** de Jon Duckett
  Un libro visualmente atractivo que ofrece una introducción completa a **HTML** y **CSS**, ideal para principiantes que buscan aprender las bases de desarrollo web. Con explicaciones claras y ejemplos visuales, es una excelente guía para aquellos que desean construir sitios web desde cero.

- **"CSS Secrets"** de Lea Verou
  En este libro, **Lea Verou** explora técnicas avanzadas de **CSS**, proporcionando soluciones creativas a problemas comunes de diseño web. Cada "secreto" revela trucos prácticos para mejorar la eficiencia y el rendimiento en la maquetación de sitios web modernos.

- **"Learning Web Design"** de Jennifer Niederst Robbins
  Un recurso completo para quienes desean aprender los fundamentos del diseño web, cubriendo **HTML5**, **CSS3**, y gráficos web. **Jennifer Niederst Robbins** explica con detalle cómo construir sitios web accesibles y optimizados, haciendo énfasis en las mejores prácticas del desarrollo web.

Enlaces a Recursos Online:

- **MDN Web Docs**
  La referencia definitiva para desarrolladores web, con documentación detallada sobre **HTML**, **CSS**, **JavaScript**, y **APIs web**. **MDN Web Docs** es una fuente confiable y precisa que ofrece ejemplos prácticos y guías exhaustivas para desarrolladores de todos los niveles.

- **CSS-Tricks**
  Un sitio web con tutoriales, artículos, y ejemplos prácticos sobre **CSS**, diseño responsivo, y técnicas avanzadas de front-end. **CSS-Tricks** es ideal para aprender soluciones innovadoras en el uso de **CSS** y explorar las últimas tendencias en diseño web.

- **A Complete Guide to Flexbox en CSS-Tricks**
  Una guía exhaustiva sobre cómo usar **Flexbox** para diseñar layouts responsivos y flexibles. Este recurso es esencial para aprender a dominar una de las herramientas más poderosas y versátiles en el diseño web moderno.

- **Can I Use**
  Un recurso útil para verificar la compatibilidad de diferentes características web en los principales navegadores. **Can I Use** ayuda a los desarrolladores a asegurarse de que las tecnologías que utilizan sean compatibles con los navegadores más utilizados por sus usuarios.

Videos Recomendados:

- **Flexbox in 20 Minutes**
  Un video tutorial que explica cómo utilizar **Flexbox** para crear layouts modernos y flexibles. En solo 20 minutos, aprenderás los fundamentos para alinear y distribuir elementos en una página de manera eficiente.

- **Responsive Web Design Introduction**
  Una introducción al diseño web responsivo que cubre los conceptos básicos para construir sitios que se adapten a diferentes tamaños de pantalla. Este video es ideal para quienes desean entender los principios clave del diseño adaptable.

- **Understanding Sass**
  Un curso rápido sobre cómo comenzar a usar **Sass**, el popular preprocesador de **CSS**, para mejorar la eficiencia y la modularidad en proyectos de desarrollo web. El video cubre la instalación, configuración y uso práctico de **Sass** en proyectos reales.

- **Bootstrap 5 Crash Course**
  Un curso rápido que enseña cómo utilizar **Bootstrap 5**, uno de los frameworks más populares para desarrollar aplicaciones web responsivas y modernas. El video cubre desde la configuración básica hasta el uso de componentes avanzados.

CORFO
BecasCapital Humano

Adalid
aspasia
LA FORMACIÓN DE TU FUTURO

MÓDULO 2

DESARROLLO DE LA INTERFAZ DE USUARIO WEB