DESARROLLO

WEB FRONT END

-INTERMEDIO

Autor de contenido

Andrés Fernando Pineda Guerra

EMPEZAR

Tabla de Contenido

Presentación

En el curso de desarrollador Full Stack como componente intermedio, podrán adquirir las habilidades y lenguajes necesarios para el desarrollo web, enfocándose en sus grandes pilares, como lo son Front End, Back End, Diseño y modelamiento de aplicaciones y documentación de código.
El curso trata temas emergentes tales como, la seguridad informática, desarrollo de aplicaciones móviles, gestión de base de datos, todo esto basado en la metodología Scrum. De la misma manera, se hace énfasis en el manejo de proyectos tanto en los módulos de desarrollo como los módulos de gestión de proyectos de TI.

Objetivos del curso (competencias)

Objetivo general

Formar a los participantes en el desarrollo web en todo el ciclo de vida del software, en donde adquieran los conocimientos básicos para implementar soluciones web.

Objetivo específico

- Conocer los conceptos y teoría básica del desarrollo web.

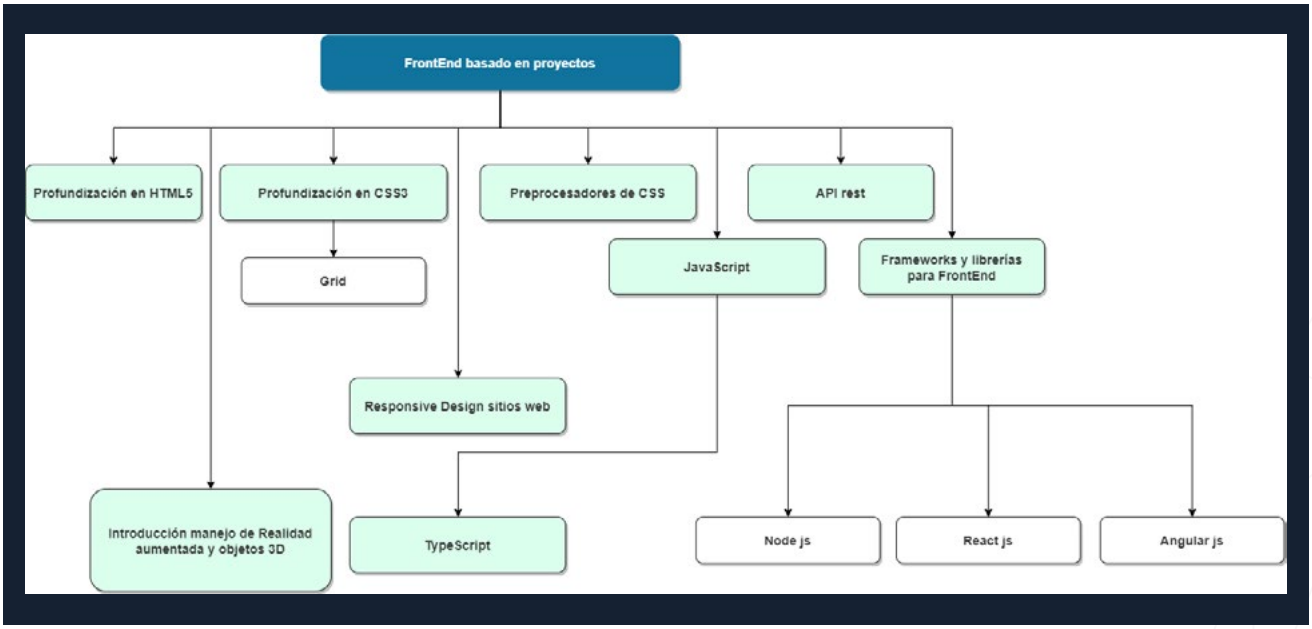
- Identificar y conocer los diferentes lenguajes de programación y herramientas para el desarrollo web.

- Aplicar las diferentes tecnologías web, tendencias y herramientas en el desarrollo de soluciones web enfocadas a proyectos.

- Diseñar, desarrollar e implementar soluciones web básicas en donde se integren los componentes de Front End, Back End, seguridad, redes y buenas prácticas utilizando metodologías ágiles.

- Identificar y conocer los conceptos básicos para el desarrollo móvil, así como aplicar su desarrollo en aplicaciones básicas.

Mapa de contenido de la unidad



Módulo 5

FrontEnd basado en proyectos

Ideas clave

Profundización en HTML5, principales etiquetas, estructuras, estructuras semánticas, manejo de imagen audio y video, elementos básicos de formularios, así como profundización de CSS3, sintaxis, propiedades y valores, selectores, flexbox, grid, manejo de preprocesadores SASS, LESS y Stylus con ejemplos.

Responsive design.

Profundización de Javascript y Typescript, declaraciones, ámbito de variables, tipos de datos, control de flujo, condicionales, bucles y manejo de API REST, REST FULL.

Frameworks, herramientas y librerías para el manejo de frontend, hamburgers, bootstrap, chart js, plotly js, google fonts, fontawesome, JQuery.

Modelado 3D y uso de realidad aumentada con herramientas de javascript.

5.1. Profundización en HTML5

HTML (Lenguaje de etiquetas de hipertexto) es un lenguaje de marcado que se utiliza para el desarrollo de páginas web. Los navegadores son los encargados de interpretar el código HTML y mostrarlo en la forma que lo hayamos desarrollado. Estos navegadores se rigen con los estándares de la W3C, lo cual nos permite realizar un desarrollo y estar seguros podrá ser interpretado en cualquier navegador.

HTML

Lenguaje marcado de hipertexto; Estructura de la página.

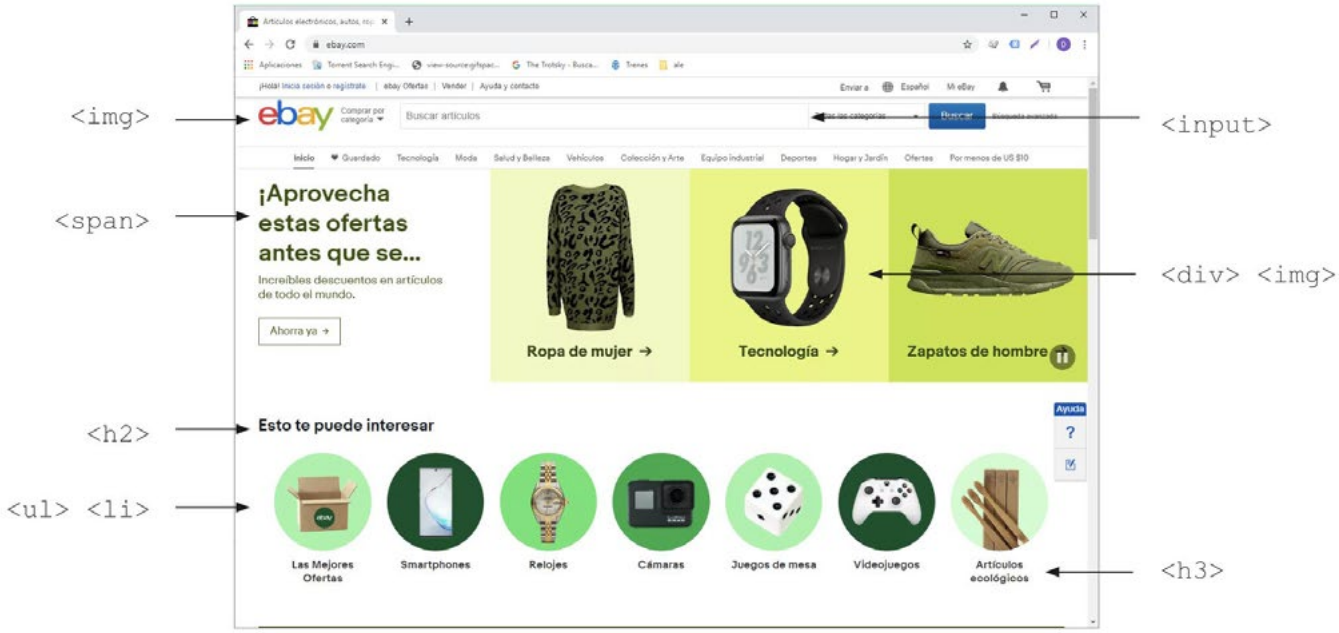
CSS

Hojas de estilo en cascada; Define los estilos de la página web.

JavaScript

Lenguaje de programación usado para agregar interacción al sitio web.

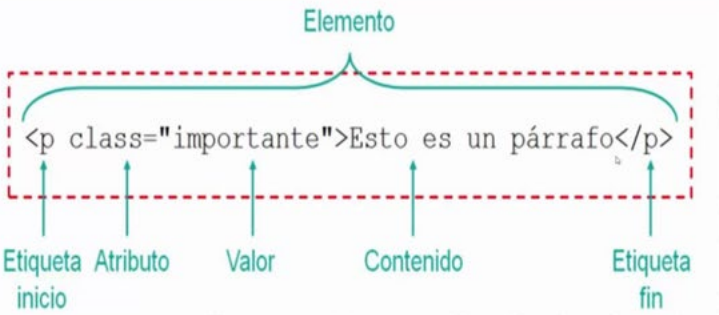

```
<!DOCTYPE html>
<html lang="en">
  <head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <title>Document</title>
  </head>
  <body>
  </body>
</html>
```



Principales etiquetas

<body>	<header>	<tfood>
<head>	<footer>	<source>
	<scrip>	<audio>
<a>	<link>	<select>
	<article>	<textarea>
 	<table>	<option>
<h1> ... <h6>	<hr>	<input>
<img>	<td>	<button>
	<tr>	<caption>
<p>	<thead>	<embed>
		<object>
<iframe>		<svg>
<style>	<i>	
<section>	<video>	

Estructura de una etiqueta



Cada elemento HTML puede tener o no atributos que definirán su comportamiento:
Básicos

id="texto"	Establece un identificador a cada elemento
class="texto"	Establece una clase CSS con la que es posible aplicar estilos a los elementos
title="texto"	Establece el título del elemento (Mejora la accesibilidad)

Los elementos HTML más utilizados son:

i<!–...–>	Define un comentario
<!DOCTYPE>	Define el tipo de documento
<a>	Define un hipervínculo
<article>	Define un artículo
<aside>	Define el contenido lateral del contenedor de una página
<audio>	Define contenido de sonido
<b>	Define texto en negrita

i<!–...–>	Define un comentario
<!DOCTYPE>	Define el tipo de documento
<a>	Define un hipervínculo
<article>	Define un artículo
<aside>	Define el contenido lateral del contenedor de una página
<audio>	Define contenido de sonido
<b>	Define texto en negrita
<bdi>	Aísla una parte del texto que puede tener un formato diferente del texto externo
<body>	Define el cuerpo del documento
 	Define un salto de línea
<button>	Define un botón clickeable
<canvas>	Se usa para dibujar gráficos en pantalla

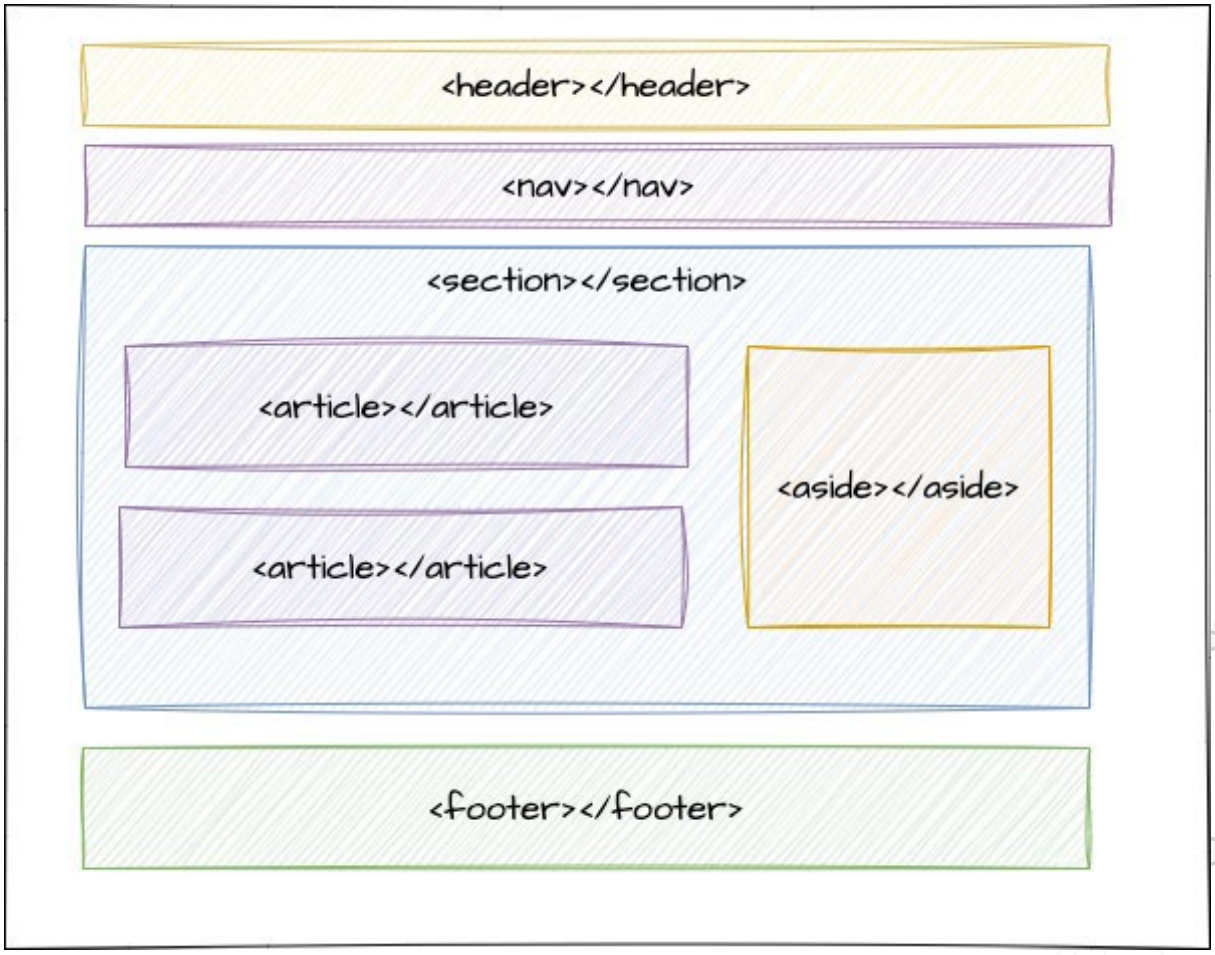
	Define una sección en un documento
<dt>	Define un término (un ítem) en una lista de definición
<footer>	Define el pie de página de un documento
<h1> a <h6>	Define encabezados o títulos
<head>	Define información acerca del documento
<header>	Define la sección de encabezado del documento
<html>	Define la raíz del documento
<i>	Define una parte del texto de modo alternativo
<iframe>	Define un frame en línea
<img>	Define una imagen
<input>	Define un control de entrada de texto
<label>	Define el rótulo para un elemento <input>

	Define un ítem de una lista
<menu>	Define la lista de un menú
<meta>	Define un metadato de un documento
<nav>	Define un link de navegación
	Define una lista ordenada
<p>	Define un párrafo
<script>	Define un script del lado cliente
<section>	Define una sección de un documento
<select>	Define un drop-down list
	Define una pequeña sección de un documento
<style>	Define un estilo para la información de un documento
<table>	Define una tabla

<tbody>	Define el cuerpo de una tabla
<td>	Define una celda en una tabla
<th>	Define una celda de encabezado en una tabla
<thead>	Agrupa los encabezados de una tabla
<title>	Define un título para el documento
<tr>	Define una fila en una tabla
	Define una lista desordenada
<var>	Define una variable
<video>	Define un vídeo o película

Estructuras semánticas

Utilizar etiquetas semánticas ayuda a los buscadores a detectar la jerarquización de contenido de nuestro sitio. Adicionalmente, facilita la lectura del documento HTML, con esto se optimiza el tiempo de desarrollo de nuevas características dentro del documento. Con el uso de etiquetas HTML semánticas los motores de búsqueda y las tecnologías de asistencia (como lectores de pantalla para usuarios con discapacidad visual) también pueden comprender mejor el contexto y el contenido del sitio web, lo que significa una mejor experiencia para los usuarios.



Lista de elementos HTML semánticos:

Tag/Elemento	**Definición**
<article>	El elemento representa una composición independiente en un documento, página, aplicación o sitio, que está destinado a ser distribuible o reutilizable de forma independiente.
<aside>	El elemento representa una parte de un documento cuyo contenido solo está indirectamente relacionado con el contenido principal del documento.

<figcaption>	El elemento representa un título o leyenda que describe el resto del contenido de su elemento padre <figure>
<figure>	El elemento representa contenido autónomo, potencialmente con un título opcional, que se especifica mediante el elemento <figcaption>.
<footer>	Normalmente contiene información sobre el autor de la sección, datos de derechos de autor o enlaces a documentos relacionados. Tambien puede contener información sobre el sitio web.
<header>	Representa contenido introductorio, normalmente un grupo de ayudas introductorias o de navegación.
<mark>	Representa texto que está marcado o resaltado para fines de referencia o notación debido a la relevancia
<nav>	Representa una sección de una página cuyo propósito es proporcionar enlaces de navegación, ya sea dentro del documento actual o hacia otros documentos.
<section>	Representa una sección independiente genérica de un documento, que no tiene un elemento semántico más específico para representarlo.

Elementos semánticos (MDN)

Imagen, Audio y video

Imagen
El tag de imagen <img> permite agregar una imagen a un documento HTML

```
<img src="mdn-logo-sm.png" alt="MDN">
```

Ejemplo tag de imagen. (MDN)

Atributos

- alt
- height
- width
- src

Audio

El tag audio se usa para insertar contenido de audio en un documento HTML o XHTML.

```
<audio src="https://developer.mozilla.org/@api/deki/files/2926/=AudioTest_(1).ogg"
       autoplay>
  Your browser does not support the <code>audio</code> element.
</audio>
```

Ejemplo tag de audio (MDN)

Atributos

- autoplay
- autobuffer
- src
- buffered
- controls

Audio

El elemento video se utiliza para incrustar vídeos en un documento HTML

```
<video src="videofile.ogg" autoplay poster="posterimage.jpg">
  Tu navegador no admite el elemento <code>video</code>.
</video>
```

Ejemplo tag de video (MDN)

Atributos

- autoplay
- controls
- height
- width
- src
- loop

Elementos básicos de formularios

Un formulario HTML es una sección del documento que contiene controles como campos de texto, campos de contraseña, checkboxes, radio buttons, etc. Este formulario facilita al usuario a ingresar los datos necesarios que serán enviados al servidor para ser procesados.

Ejemplo de formulario (Bootstrap Documentation)

Los tags más utilizados para crear formularios son:

- <form> Define un formulario HTML para ingresar entradas por el lado usado.
- <input> Define un campo de entrada.
- <textarea> Define un campo de entrada de texto multilínea.
- <label> Define un label para un campo de entrada
- <select> Define una lista de opciones
- <option> Define una opción dentro de select
- <button> Define un botón

5.2. Profundización en CSS3

Sintaxis

CSS Hojas de Estilo en Cascada (del inglés Cascading Style Sheets) es el lenguaje de estilos utilizado para describir la presentación de documentos. CSS permite describir cómo debe ser renderizado el elemento en la pantalla.

Cada instrucción css está compuesta por una propiedad y un valor. La propiedad es un identificador de la característica de ese elemento que se dea mutar y el valor es la característica que debe ser aplicada y manejada por el navegador.

Sintaxis CSS

```
p {
  color: red;
}
```

Sintaxis CSS

Propiedades más utilizadas y sus posibles valores:

Propiedad	**Definición**
align-content	Especifica la alineación de los elementos del contenedor flexible.
align-items	Especifica la alineación predeterminada de los elementos dentro del contenedor flexible.
align-self	Especifica la alineación de los elementos seleccionados dentro del contenedor flexible.
animation	Especifica las animaciones basadas en fotogramas.
background	Define una variedad de propiedades de fondo dentro de una declaración.
background-color	Define el color de fondo de un elemento.
background-image	Define la imagen de fondo de un elemento.
background-position	Define el origen de una imagen de fondo.

background-repeat	Especifica la imagen de fondo está en mosaico.
background-size	Especifica el tamaño de las imágenes de fondo.
border	Establece el ancho, el estilo y el color de los cuatro lados del borde de un elemento.
border-color	Establece el color del borde en los cuatro lados de un elemento.
border-left-color	Establece el color del borde izquierdo de un elemento.
border-left-style	Establece el estilo del borde izquierdo de un elemento.
border-left-width	Establece el ancho del borde izquierdo de un elemento.
border-radius	Define la forma de las esquinas de los bordes de un elemento.
border-spacing	Establece el espacio entre los bordes de las celdas de tabla adyacentes.
border-style	Establece el estilo del borde en los cuatro lados de un elemento.
bottom	Especifica la ubicación del borde inferior del elemento posicionado.
box-shadow	Aplica una o más sombras al cuadro del elemento.
color	Especifica el color del texto de un elemento.

display	Especifica cómo se muestra un elemento en pantalla.
flex	Especifica los componentes de una longitud flexible
flex-basis	Especifica el tamaño principal inicial del elemento flexible.
flex-direction	Especifica la dirección de los elementos flexibles.
flex-grow	Especifica cómo crecerá el elemento flexible en relación con los otros elementos dentro del contenedor flexible.
flex-wrap	Especifica si los artículos flexibles deben envolverse o no.
float	Especifica si una caja debe flotar o no.
font-family	Define una lista de fuentes para el elemento.
font-size	Define el tamaño de fuente para el texto.
height	Especifica la altura de un elemento.
justify-content	Especifica cómo se alinean los elementos flexibles a lo largo del eje principal del contenedor flexible después de que se hayan resuelto las longitudes flexibles y los márgenes automáticos.
left	Especifica la ubicación del borde izquierdo del elemento posicionado.

line-height	Establece la altura entre líneas de texto.
list-style	Define el estilo de visualización para una lista y elementos de lista.
list-style-image	Especifica la imagen que se utilizará como marcador de elemento de lista.
list-style-position	Especifica la posición del marcador de elemento de lista.
list-style-type	Especifica el estilo de marcador para un elemento de lista.
margin	Establece el margen en los cuatro lados del elemento.
max-height	Especifica la altura máxima de un elemento.
max-width	Especifica el ancho máximo de un elemento.
min-height	Especifica la altura mínima de un elemento.
min-width	Especifica el ancho mínimo de un elemento.
opacity	Especifica la transparencia de un elemento.
overflow	Especifica el tratamiento del contenido que desborda el cuadro del elemento.

overflow-x	Especifica el tratamiento del contenido que desborda el cuadro del elemento horizontalmente.
overflow-y	Especifica el tratamiento del contenido que desborda el cuadro del elemento verticalmente.
padding	Establece el espacio de relleno en los cuatro lados del elemento.

Selectores

Existe una gran variedad de selectores CSS para aplicar estilos a un sitio web. Estos selectores permiten elegir los tags HTML a los que se aplicarán las reglas CSS definidas. Los selectores más conocidos son:

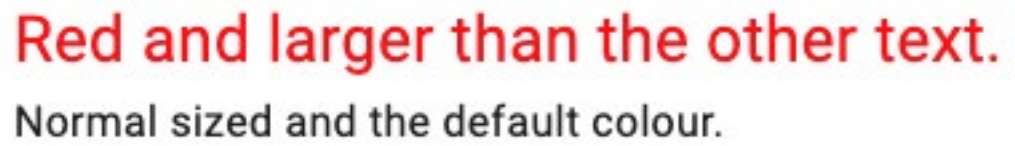
- universal -> *
- tag -> span
- class y id
- atributos -> a[href="https://example.com"]
- pseudo-clases y pseudo elementos -> Es un selector que permite elegir un tag HTML pero teniendo en cuenta su estado. -> :focus, :hover

```html
<article>
  <p>Rojo y más grande que el otro texto.</p>
  <p>Tamaño normal y el color predeterminado.</p>
</article>
```

Ejemplo código HTML

```css
article p:first-of-type {
  color: red;
  font-size: 1.5em;
}
```

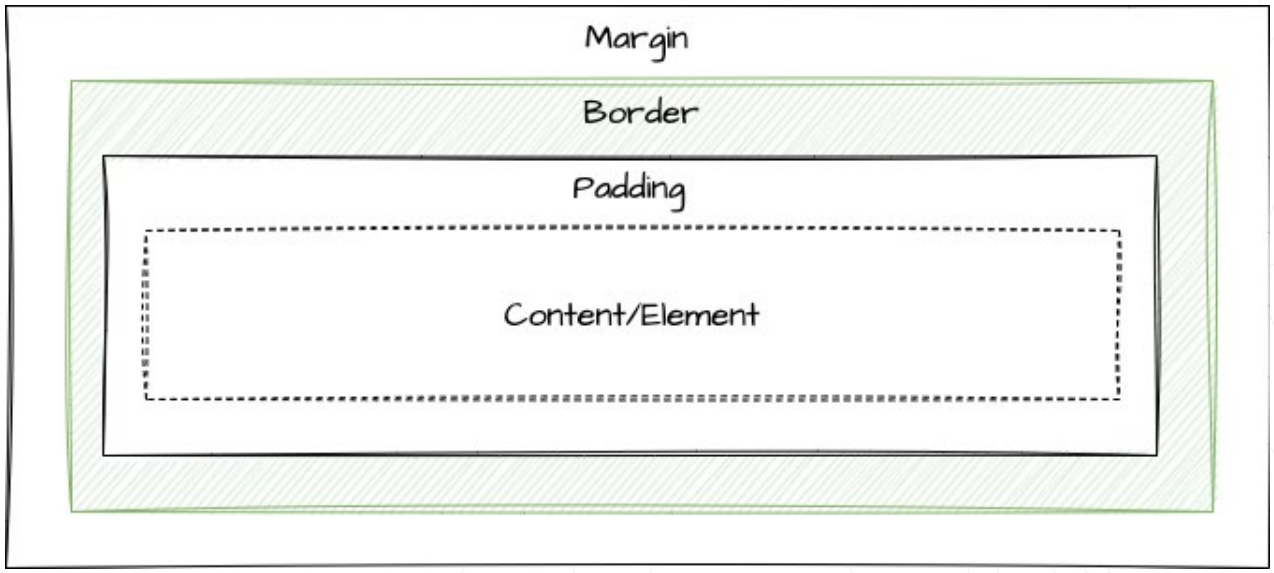
Ejemplo de selector CSS



Resultado visual de implementación de CSS

Flexbox y grid

El modelo de caja CSS es esencialmente un cuadro que envuelve cada elemento HTML. Consiste en: márgenes, bordes, relleno y el contenido real. La siguiente imagen ilustra el modelo de caja:



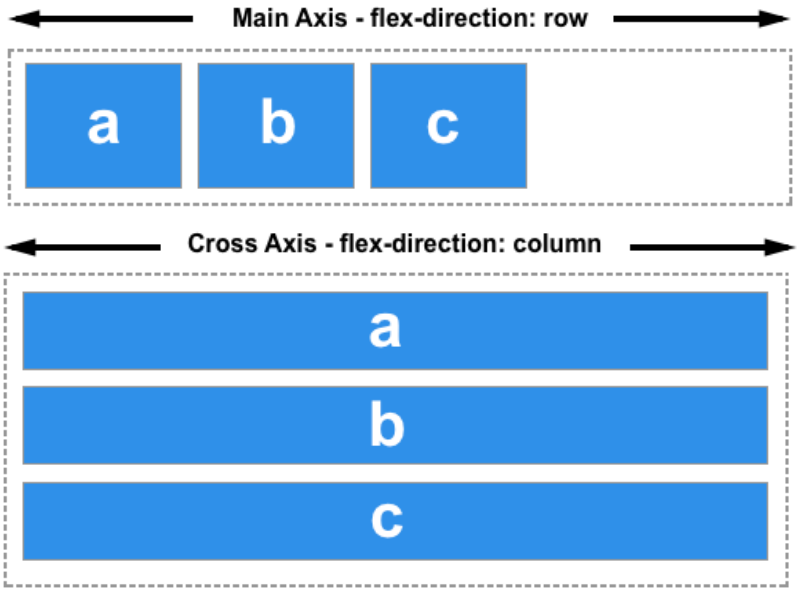
Modelo de caja en CSS

Content - El contenido de la caja, donde aparecen texto e imágenes.
Padding - Borra un área alrededor del contenido. El relleno es transparente.
Border - Un borde que rodea el relleno y el contenido.
Margin - Borra un área fuera del borde. El margen es transparente.

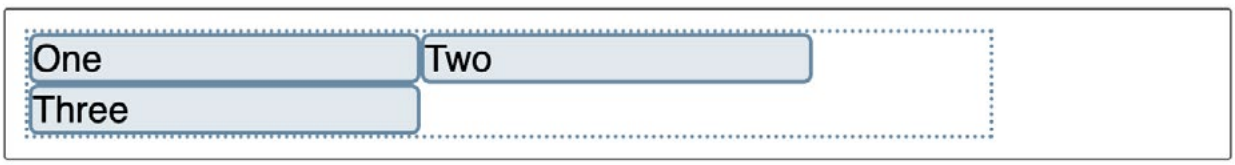
Flexbox, fue diseñado como un modelo de diseño unidimensional y como un método que podría ofrecer distribución espacial entre elementos en una interfaz y potentes capacidades de alineación. Este artículo ofrece un resumen de las características principales de flexbox, que exploraremos con más detalle en el resto de estas guías.

Se define por medio de flex-direction:

- row
- row-reverse
- column
- column-reverse



Flex direction (MDN)

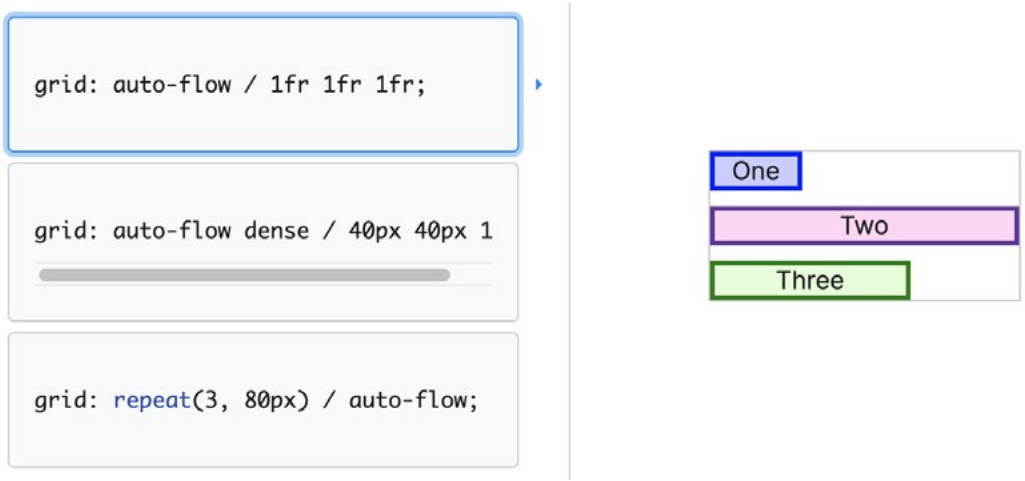



```
.box {
  display: flex;
  flex-flow: row wrap;
}
```

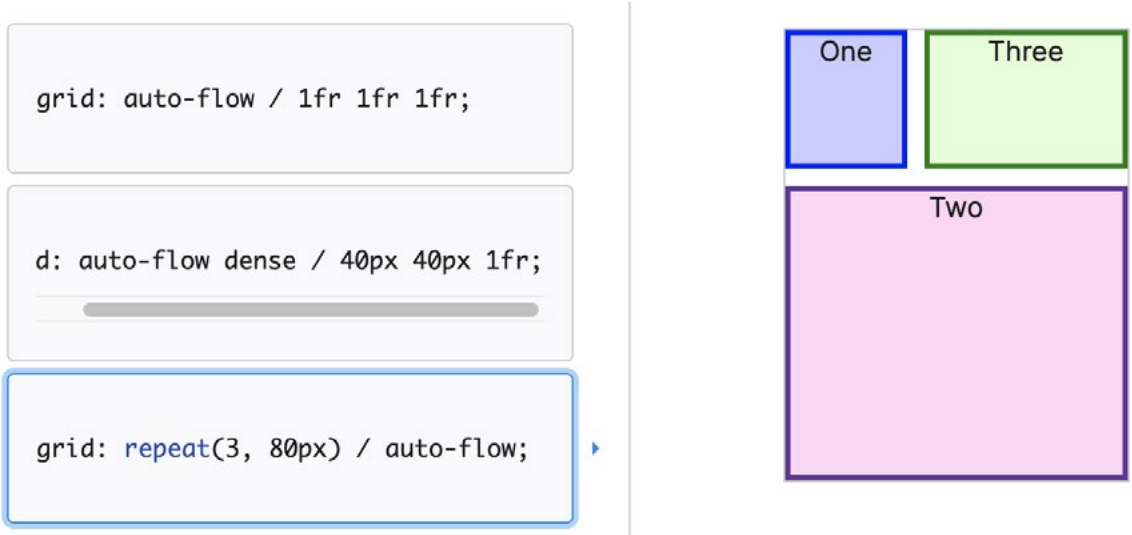
```
<div class="box">
  <div>One</div>
  <div>Two</div>
  <div>Three</div>
</div>
```

Flex direction (MDN)

Usando grid usted especifica un eje usando grid-template-rows o grid-template-columns, luego especifica cómo el contenido debe repetirse automáticamente en el otro eje usando las propiedades de cuadrícula implícitas:
grid-auto-rows, grid-auto-columns, and grid-auto-flow.



CSS Grid (MDN)



CSS Grid (MDN)

Especificidad de selectores

Si hay dos o más CSS reglas que apuntan al mismo elemento, el selector con el valor de especificidad más alto “ganará” y su declaración de estilo se aplicará a ese HTML elemento. Piense en la especificidad como una puntuación/rango que determina qué declaración de estilo se aplica en última instancia a un elemento.
La siguiente tabla muestra algunos ejemplos de cómo calcular los valores de especificidad:

Selector	Valor de Especificidad	Cálculo
p	1	1
p.test	11	1 + 10
p#demo	101	1 + 100
<p style=”color: pink; “ >	1000	1000
#demo	100	100
.test	10	10
p.test1.test2	21	1 + 10 + 10
#navbar p#demo	201	100 + 1 + 100
*	0	0 (se ignora el selector universal)

Ejemplo:

```
A: h1
B: h1#content
C: <h1 id="content" style="color: pink;">Heading</h1>
```

Animaciones

Es posible crear una secuencia de animaciones con CSS por medio de la propiedad animation y sus subpropiedades. Las siguientes subpropiedades permiten configurar el ritmo y la duración de cada animación, entre otros aspectos:

- animation-delay
- animation-direction
- animation-duration

Ejemplo:

```
p {
  animation-duration: 3s;
  animation-name: slidein;
}

@keyframes slidein {
  from {
    margin-left: 100%;
    width: 300%
  }

  to {
    margin-left: 0%;
    width: 100%;
  }
}
```

5.3. Preprocesadores de CSS (SASS, LESS)

Definición, historia, importancia

Los preprocesadores de CSS son lenguajes de secuencias de comandos que amplían las capacidades predeterminadas de CSS. Nos permiten usar la lógica en nuestro código CSS, como variables, anidamiento, herencia, mixins, funciones y operaciones matemáticas.

Cada preprocesador de CSS tiene su propia sintaxis que compilan en CSS normal para que los navegadores puedan representarlo en el lado del cliente. Actualmente, los tres preprocesadores CSS más populares y estables son Sass, LESS y Stylus; sin embargo, también hay muchos más pequeños. Todos los preprocesadores de CSS hacen cosas similares pero de una manera diferente y con sus propias sintaxis. Cada uno de ellos tiene algunas características avanzadas únicas para ellos y también su propio ecosistema (herramientas, marcos, bibliotecas).

Ventajas:

- - Facilitan la automatización de tareas repetitivas
- - Reducen la cantidad de errores y la sobrecarga de código
- - Crean fragmentos de código reutilizables
- - Garantizan la compatibilidad con versiones anteriores

LESS: Leaner Style Sheets

LESS se lanzó inicialmente tres años después de Sass, en 2009 por Alexis Sellier. Fue influenciado por Sass, por lo que implementa muchas de sus funciones, como mixins, variables y anidamiento. Curiosamente, LESS posterior también influyó en Sass, ya que la sintaxis SCSS más nueva se inspiró en la sintaxis de LESS.

El preprocesador LESS CSS es, de hecho, una biblioteca de JavaScript que amplía las funcionalidades predeterminadas de CSS. Como está escrito en JavaScript, necesitamos Node.js para instalar y ejecutar el compilador LESS. Sin embargo, también podemos compilar LESS sobre la marcha agregando archivos .less y el convertidor LESS a la sección <head> de nuestra página HTML.

Stylus

La primera versión de Stylus se lanzó un año después de LESS, en 2010, por TJ Holowaychuk, un antiguo desarrollador de Node.js. Stylus está escrito en Node.js para que los desarrolladores puedan integrarlo fácilmente en sus proyectos de Node. Fue influen-

ciado tanto por Sass como por LESS. Stylus combina las poderosas habilidades lógicas de Sass con la configuración fácil y directa de LESS.

Sass: Syntactically Awesome Style Sheet

Sass es el preprocesador CSS más antiguo, lanzado inicialmente en 2006. Sus creadores, Natalie Weizenbaum y Hampton Catlin, se inspiraron en el lenguaje de plantillas Haml, que agrega características dinámicas a HTML. Su objetivo era implementar una funcionalidad dinámica similar en CSS también. Entonces, se les ocurrió un preprocesador CSS y lo llamaron Hojas de estilo sintácticamente asombrosas. Sass permite a los desarrolladores frontend usar variables, declaraciones if/else, bucles for/while/each, herencia y otra lógica computacional en su código CSS.

Sass está escrito en Ruby, y originalmente también necesitaba Ruby para compilar el código, lo que disuadió a muchos desarrolladores de usarlo. Sin embargo, la introducción de la biblioteca LibSass dio un gran impulso a la utilización de Sass.

A nivel general, todos los preprocesadores de CSS tienen funcionalidades similares, como variables, mezclas, importación y anidamiento. Todos siguen el principio DRY y pueden realizar declaraciones condicionales, funciones y operaciones. Sin embargo, hay algunas diferencias importantes en sus funciones avanzadas que debemos considerar antes de elegir una sobre la otra. Sass y Stylus son más como lenguajes de programación, ya que tienen capacidades lógicas y de bucle avanzadas y nos permiten escribir funciones personalizadas. Además, Sass tiene un gran ecosistema y una comunidad activa, y los marcos frontend populares también lo usan. Por otro lado, Stylus tiene una sintaxis increíblemente flexible y puede integrarse fácilmente en proyectos de Node. Y aunque LESS tiene menos funciones basadas en lógica que Sass y Stylus, se puede compilar fácilmente en la interfaz, lo que lo hace excelente en una arquitectura sin servidor y también tiene excelentes funciones integradas.

Sintaxis

En esta sección revisaremos la sintaxis de SASS, el preprocesador de CSS más utilizado en la actualidad.

Sass tiene dos sintaxis. La extensión de archivo .sass utiliza la sintaxis anterior que se basa en la sangría y omite los puntos y comas y las llaves del código. La sintaxis más nueva y más utilizada que pertenece a la extensión de archivo .scss. Utiliza la sintaxis CSS estándar con llaves y punto y coma.

A continuación, se puede ver un ejemplo básico de la sintaxis de Sass/SCSS. El código simplemente declara dos variables, $primary-color y $primary-bg y las aplica al elemento body del documento HTML.

```
/* SCSS */

$primary-color: seashell;
$primary-bg: darkslategrey;

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

El mismo código con la sintaxis de Sass:

```
/* Sass */

$primary-color: seashell
$primary-bg: darkslategrey

body
color: $primary-color
background: $primary-bg
```

Ambos compilan al mismo CSS:

```
/* Compiled CSS */

body {
color:seashell;
background: darkslategrey;
}
```

Mientras que la sintaxis Sass más antigua es más rápida de escribir y más difícil equivocarse, la sintaxis SCSS más nueva es totalmente compatible con la sintaxis CSS normal.

Errores

Cuando Sass encuentra una sintaxis no válida en una hoja de estilo, el parseo o conversión a CSS fallará y se presentará un error al usuario con información sobre la ubicación de la sintaxis no válida y la razón por la que no lo era.

Es necesario tener en cuenta que esto es diferente a CSS, que especifica cómo recuperarse de la mayoría de los errores en lugar de fallar inmediatamente.

Estructura de una hoja de estilos

Al igual que CSS, la mayoría de las hojas de estilo de Sass se componen principalmente de reglas de estilo que contienen declaraciones de propiedades. Pero las hojas de estilo Sass tienen muchas más características que pueden existir junto con estas.

Declaraciones

Una hoja de estilo Sass se compone de una serie de declaraciones, que se evalúan para construir el CSS resultante. Algunas sentencias pueden tener bloques, definidos mediante { and }, que contienen otras sentencias. Por ejemplo, una regla de estilo es una declaración con un bloque. Este bloque contiene otras declaraciones, como declaraciones de propiedades.
En SCSS, las declaraciones están separadas por punto y coma (que son opcionales si la declaración usa un bloque). En la sintaxis con sangría, sólo están separados por saltos de línea.

- Declaraciones Universales: Estos tipos de declaraciones se pueden usar en cualquier parte de una hoja de estilo Sass:
- Declaraciones de variables como: $var: value
- Reglas de control de flujo: @if y @each
- Reglas de error, warning y debug como: @error, @warn, and @debug
- Declaraciones CSS: Estas declaraciones producen CSS. Se pueden usar en cualquier lugar excepto dentro de una @function:
- Reglas de estilo como: h1 { /* … */ }
- @media y @font-face
- Mixin usando @include
- La regla @at-root
- Sentencias de alto nivel: Estas declaraciones solo se pueden usar en el nivel superior de una hoja de estilo o anidadas dentro de una declaración CSS en el nivel superior:
- Carga de módulos usando @use
- Imports usando @Import
- Definiciones de mixins usando @mixin
- Definición de función usando @function

Expresiones

Una expresión es cualquier cosa que va en el lado derecho de una declaración de propiedad o variable. Cada expresión produce un valor. Cualquier valor de propiedad CSS válido también es una expresión Sass, pero las expresiones Sass son mucho más poderosas que los valores CSS simples. Se pasan como argumentos a mixins y funciones, se usan para controlar el flujo con la regla @if y se manipulan usando aritmética.

- Literales: Las expresiones más simples sólo representan valores estáticos.
- Números: como 12 o 100px
- Cadenas de texto: Pueden o no estar rodeadas por comillas como “Helvetica Neue” o bold
- Colores como #c6538c o blue.
- Booleanos, true o false
- Lista de valores, que puede estar separada por espacios o comas y pueden estar encerrados en corchetes ([ ]) o no, como: 1.5em 1em 0 2em, Helvetica, Arial, sans-serif, o [col1-start]
- Mapas que asocian valores y llaves: (“background”: red, “foreground”: pink)
- Operaciones: Sass define la sintaxis para una serie de operaciones:
- == y !=
- +, -, *, /
- <, <=, >, >=
- and, or y not
- +, - y /

Ejemplo:

```
// SCSS
.circle {
$size: 100px;
width: $size;
height: $size;
border-radius: $size * 0.5;
}

// Saas
.circle
$size: 100px
width: $size
height: $size
border-radius: $size * 0.5

// CSS resultante
.circle {
width: 100px;
height: 100px;
border-radius: 50px;
}
```

Nesting, mixins, funciones
Mixin e Include

Los mixins permiten definir estilos que se pueden reutilizar en toda la hoja de estilos. Facilitan evitar el uso de clases no semánticas como .float-left y distribuir colecciones de estilos en bibliotecas.
Los mixins se incluyen en el contexto actual usando la regla arroba @include, que se escribe @include <nombre> o @include <nombre>(<argumentos...>), con el nombre del mixin incluido.

Ejemplo:

```
@mixin reset-list {
margin: 0;
padding: 0;
list-style: none;
}

@mixin horizontal-list {
@include reset-list;

li {
display: inline-block;
margin: {
left: -2px;
right: 2em;
}
}
}

nav ul {
@include horizontal-list;
}
```

El CSS resultante es:

```
nav ul {
margin: 0;
padding: 0;
list-style: none;
}
nav ul li {
display: inline-block;
margin-left: -2px;
margin-right: 2em;
}
```

Los mixins también pueden tomar argumentos, lo que permite personalizar su comportamiento cada vez que se les llama. Los argumentos se especifican en la regla @mixin después del nombre del mixin, como una lista de nombres de variables entre paréntesis. Luego, el mixin debe incluirse con el mismo número de argumentos.

Ejemplo:

```
@mixin rtl($property, $ltr-value, $rtl-value) {
#{$property}: $ltr-value;

[dir=rtl] & {
#{$property}: $rtl-value;
}
}

.sidebar {
@include rtl(float, left, right);
}
```

Con argumentos opcionales, tenemos lo siguiente:

```
@mixin replace-text($image, $x: 50%, $y: 50%) {
text-indent: -99999em;
overflow: hidden;
text-align: left;

background: {
image: $image;
repeat: no-repeat;
position: $x $y;
}
}
```

```
.mail-icon {
@include replace-text(url("/images/mail.svg"), 0);
}
```

Interpolación

El nombre de una propiedad puede incluir interpolación, lo que hace posible generar propiedades dinámicamente según sea necesario. Es posible interpolar el nombre completo de la propiedad.

```
// SCSS
@mixin prefix($property, $value, $prefixes) {
@each $prefix in $prefixes {
-#{$prefix}-#{$property}: $value;
}
#{$property}: $value;
}

.gray {
@include prefix(filter, grayscale(50%), moz webkit);
}
```

Nesting

Muchas propiedades CSS comienzan con el mismo prefijo que actúa como una especie de espacio de nombres. Por ejemplo, font-family, font-size y font-weight comienzan con font-. Sass hace que esto sea más fácil y menos redundante al permitir que se aniden las declaraciones de propiedad. Los nombres de las propiedades externas se agregan a las internas, separados por un guión.

```
.enlarge {
font-size: 14px;
transition: {
property: font-size;
duration: 4s;
delay: 2s;
}

&:hover { font-size: 36px; }
}
```

CSS resultante:

```
.enlarge {
font-size: 14px;
transition-property: font-size;
transition-duration: 4s;
transition-delay: 2s;
}
.enlarge:hover {
 font-size: 36px;
}
```

Placeholder, import y partials

Placeholder selector

Sass tiene un tipo especial de selector conocido como “placeholder”. Se ve y actúa como un selector de clase, pero comienza con un % y no está incluido en la salida CSS.

```
.alert:hover, %strong-alert {
font-weight: bold;
}

%strong-alert:hover {
color: red;
}
```

CSS resultante:

```
.alert:hover {
font-weight: bold;
}
```

A diferencia de los selectores de clase, los placeholders no saturan el CSS si no están extendidos y no exigen que los usuarios de una biblioteca usen nombres de clase específicos para su HTML.

Ejemplo:

```
%toolbelt {
box-sizing: border-box;
border-top: 1px rgba(#000, .12) solid;
padding: 16px 0;
width: 100%;
```

```
&:hover { border: 2px rgba(#000, .5) solid; }
}

.action-buttons {
@extend %toolbelt;
color: #4285f4;
}

.reset-buttons {
@extend %toolbelt;
color: #cddc39;
}
```

CSS resultante:

```
.action-buttons, .reset-buttons {
box-sizing: border-box;
border-top: 1px rgba(0, 0, 0, 0.12) solid;
padding: 16px 0;
width: 100%;
}
.action-buttons:hover, .reset-buttons:hover {
border: 2px rgba(0, 0, 0, 0.5) solid;
}

.action-buttons {
color: #4285f4;
}

.reset-buttons {
color: #cddc39;
}
```

Los selectores de marcador de posición son útiles cuando se escribe una biblioteca Sass en la que cada regla de estilo puede usarse o no.

Import

Sass amplía la regla @import de CSS con la capacidad de importar hojas de estilo Sass y CSS, brindando acceso a mixins, funciones y variables y combinando CSS de múltiples hojas de estilo. A diferencia de las importaciones de CSS simples, que requieren que el navegador realice varias solicitudes HTTP a medida que representa su página, las importaciones de Sass se manejan por completo durante la compilación.

Ejemplo:

```
// foundation/_code.scss
code {
padding: .25em;
line-height: 0;
}

// foundation/_lists.scss
ul, ol {
text-align: left;

& & {
padding: {
bottom: 0;
left: 0;
}
}
}

// style.scss
@import ‘foundation/code’, ‘foundation/lists’;
```

Cuando Sass importa un archivo, ese archivo se evalúa como si su contenido apareciera directamente en lugar de @import. Todos los mixins, funciones y variables del archivo importado están disponibles, y todo su CSS se incluye en el punto exacto donde se escribió @import. Además, todos los mixins, funciones o variables que se definieron antes de la @importación (incluso de otras @importaciones) están disponibles en la hoja de estilo importada. El CSS resultante del anterior ejemplo es:

```
code {
padding: .25em;
line-height: 0;
}

ul, ol {
text-align: left;
}
ul ul, ol ol {
padding-bottom: 0;
padding-left: 0;
}
```

Partials

Es posible crear archivos Sass parciales que contengan pequeños fragmentos de CSS que puede incluir en otros archivos Sass. Esta es una excelente manera de modularizar su CSS y ayudar a que las cosas sean más fáciles de mantener. Un parcial es un archivo Sass nombrado con un guión bajo al principio. Puede nombrarlo algo como _parcial. scss. El guión bajo le permite a Sass saber que el archivo es solo un archivo parcial y que no debe generarse en un archivo CSS. Los parciales de Sass se usan con la regla @ use.

Ejemplo:

```
// _base.scss
$font-stack: Helvetica, sans-serif;
$primary-color: #333;

body {
font: 100% $font-stack;
color: $primary-color;
}

// styles.scss
@use ‘base’;

.inverse {
background-color: base.$primary-color;
color: white;
}
```

Estructuras de control

@if

La regla @if se escribe @if <expression> { ... }, y controla si un bloque se evalúa o no (incluida la emisión de cualquier estilo como CSS). La expresión generalmente devuelve verdadero o falso: si la expresión devuelve verdadero, el bloque se evalúa, y si la expresión devuelve falso, no lo es.

```
@mixin avatar($size, $circle: false) {
width: $size;
height: $size;
```

```
@if $circle {
border-radius: $size / 2;
}
}

.square-av {
@include avatar(100px, $circle: false);
}
.circle-av {
@include avatar(100px, $circle: true);
}
```

CSS resultante:

```
.square-av {
width: 100px;
height: 100px;
}

.circle-av {
width: 100px;
height: 100px;
border-radius: 50px;
}
@else
```

Una regla @if puede ser seguida opcionalmente por una regla @else, escrita @else { ... }. El bloque de esta regla se evalúa si la expresión @if devuelve falso.

```
$light-background: #f2ece4;
$light-text: #036;
$dark-background: #6b717f;
$dark-text: #d2e1dd;

@mixin theme-colors($light-theme: true) {
@if $light-theme {
background-color: $light-background;
color: $light-text;
} @else {
background-color: $dark-background;
color: $dark-text;
}
}
```

```
.banner {
@include theme-colors($light-theme: true);
body.dark & {
@include theme-colors($light-theme: false);
}
}
```

CSS resultante

```
.banner {
background-color: #f2ece4;
color: #036;
}
body.dark .banner {
background-color: #6b717f;
color: #d2e1dd;
}
```

@for

La regla @for, escrita @for <variable> de <expresión> a <expresión> { ... } o @for <variable> de <expresión> a <expresión> { ... }, cuenta hacia arriba o hacia abajo desde uno número (el resultado de la primera expresión) a otro (el resultado de la segunda) y evalúa un bloque para cada número intermedio. Cada número en el camino se asigna al nombre de variable dado. Si se usa to, se excluye el número final; si se usa through, está incluido.

```
$base-color: #036;

@for $i from 1 through 3 {
ul:nth-child(3n + #{$i}) {
background-color: lighten($base-color, $i * 5%);
}
}
```

CSS resultante:

```
ul:nth-child(3n + 1) {
background-color: #004080;
}

ul:nth-child(3n + 2) {
background-color: #004d99;
}
```

```
ul:nth-child(3n + 3) {
background-color: #0059b3;
}
@while
```

La regla @while, escrita @while <expresión> { ... }, evalúa su bloque si su expresión devuelve verdadero. Luego, si su expresión aún devuelve verdadero, evalúa su bloque nuevamente. Esto continúa hasta que la expresión finalmente retorna falso.

```
@use “sass:math”;

/// Divides `$value` by `$ratio` until it’s below `$base`.
@function scale-below($value, $base, $ratio: 1.618) {
@while $value > $base {
$value: math.div($value, $ratio);
}
@return $value;
}

$normal-font-size: 16px;
sup {
font-size: scale-below(20px, 16px);
}
```

El código CSS resultante:

```
sup {
font-size: 12.36094px;
}
```

5.4. Javascript

JavaScript (JS) es un lenguaje de programación ligero, interpretado, o compilado justo-a-tiempo (just-in-time) con funciones de primera clase. Si bien es más conocido como un lenguaje de scripting (secuencias de comandos) para páginas web, y es usado en muchos entornos fuera del navegador, tal como Node.js, Apache CouchDB y Adobe Acrobat JavaScript es un lenguaje de programación basada en prototipos, multiparadigma, de un solo hilo, dinámico, con soporte para programación orientada a objetos, imperativa y declarativa (por ejemplo programación funcional).

Sintaxis

Javascript

Es un lenguaje de programación que facilita la creación de contenido dinámico de páginas web. Es posible crear elementos para mejorar la interacción de los usuarios con las páginas web, como menús desplegables, gráficos animados y colores de fondo dinámicos.

Ventajas:

Simplicidad: tener una estructura simple hace que JavaScript sea más fácil de aprender e implementar, y también se ejecuta más rápido que otros lenguajes. Los errores también son fáciles de detectar y corregir.
Velocidad: JavaScript ejecuta scripts directamente dentro del navegador web sin necesitar un compilador.
Actualizaciones: el equipo de desarrollo de JavaScript y ECMA International actualizan y crean continuamente nuevos marcos y bibliotecas, lo que garantiza su relevancia dentro de la industria.

Sintaxis

- Comentarios:

```
// un comentario de una línea

/* este es un comentario
 * más largo, de varias líneas
 */
```

Comentarios en Javascript. (MDN)

Declaraciones

Las declaraciones de variables se realizan por medio de:

Palabra reservada	**Descripción**
var	Permite declarar una variable con ámbito global. Es opcional inicializar con un valor.
let	Permite declarar una variable con ámbito local de bloque. Es opcional inicializar con un valor.
const	Permite declarar una constante de solo lectura y ámbito de bloque.

Ámbito de variables

Las variables declaradas en javascript fuera de un bloque de código, se denomina una variable global. Si por el contrario la variable se declara dentro de un bloque se llama variable local.

```
if (true) {
  var x = 5;
}
console.log(x);  // x es 5
```

Ejemplo declaración de variable global. (MDN)

```
if (true) {
  let y = 5;
}
console.log(y); // ReferenceError: y no está definida
```

Ejemplo declaración de variable local. (MDN)

```
const PI = 3.14;
```

Ejemplo declaración de constantes. (MDN)

- Tipos de datos

Tipo de dato	**Posibles valores**
Boolean	true y false
null	Es una referencia que apunta a una dirección inexistente
undefined	Usualmente asignado a variables que solo han sido declaradas y no inicializadas

Number	**Número entero o decimal. Ejemplo: 56, 3.1416**
BigInt	Se trata de un número entero con precisión arbitraria.
String	Representa una secuencia de caracteres. Ejemplo: “Hola”
Arrays	
Object	Es utilizado para guardar una colección de datos definidos y entidades más complejas.

Control de flujo
Condicionales

```
if (condition_1) {
  statement_1;
} else if (condition_2) {
  statement_2;
} else if (condition_n) {
  statement_n;
} else {
  statement_last;
}
```

Estructura condicional. (MDN)

```
function checkData() {
  if (document.form1.threeChar.value.length == 3) {
    return true;
  } else {
    alert(
      "Introduce exactamente tres caracteres. " +
        `${document.form1.threeChar.value} no es válido.`
    );
    return false;
  }
}
```

Estructura condicional. (MDN)

Switch

```
switch (expression) {
  case label_1:
    statements_1
    [break;]
  case label_2:
    statements_2
    [break;]
    …
  default:
    statements_def
    [break;]
}
```

Estructura condicional switch. (MDN)

```
switch (fruittype) {
  case "Oranges":
    console.log("Las naranjas cuestan $0.59 la libra.");
    break;
  case "Apples":
    console.log("Las manzanas cuestan $0.32 la libra.");
    break;
  case "Bananas":
    console.log("Los plátanos cuestan $0.48 la libra.");
    break;
  case "Cherries":
    console.log("Las cerezas cuestan $3.00 la libra.");
    break;
  case "Mangoes":
    console.log("Los mangos cuestan $0.56 la libra.");
    break;
  case "Papayas":
    console.log("Los mangos y las papayas cuestan $2.79 la libra.");
    break;
  default:
    console.log(`Lo sentimos, no tenemos ${fruittype}.`);
}
console.log("¿Hay algo más que quieras?");
```

Ejemplo estructura switch. (MDN)

Manejo de errores

El manejo de errores con Javascript se realiza por medio de la declaración try...catch, el try es utilizado para definir un bloque de instrucciones que deben ser ejecutadas y en caso de que se presente algún error, se ejecutan las instrucciones definidas en el bloque catch.

```
function getMonthName(mo) {
  mo = mo - 1; // Ajusta el número de mes para el índice del arreglo
(1 = Ene, 12 = Dic)
  let months = [
    "Ene",
    "Feb",
    "Mar",
    "Abr",
    "May",
    "Jun",
    "Jul",
    "Ago",
    "Sep",
    "Oct",
    "Nov",
    "Dic",
  ];
  if (months[mo]) {
    return months[mo];
  } else {
    throw "InvalidMonthNo"; // aquí se usa la palabra clave throw
  }
}

try {
  // declaraciones para try
  monthName = getMonthName(myMonth); // la función podría lanzar una
excepción
} catch (e) {
  monthName = "unknown";
  logMyErrors(e); // pasar el objeto exception al controlador de
errores (es decir, su propia función)
}
```

Ejemplo de manejo de errores. (MDN)

Bucle for

```
for (let step = 0; step < 5; step++) {
  // Se ejecuta 5 veces, con valores del paso 0 al 4.
  console.log('Camina un paso hacia el este');
}
```

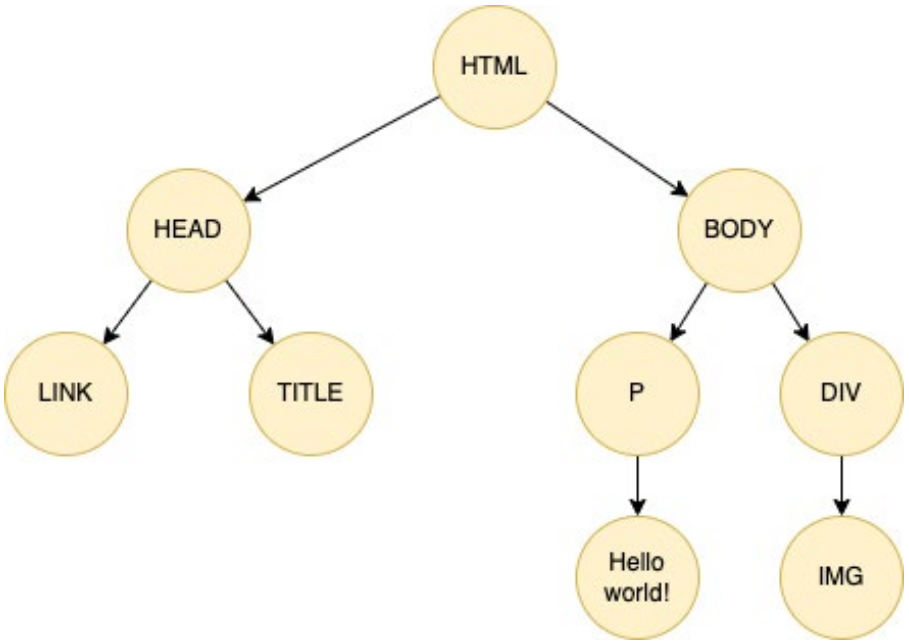
Ejemplo bucle for. (MDN)

While

```
let n = 0;
let x = 0;
while (n < 3) {
  n++;
  x += n;
}
```

DOM (Document Object Model)

DOM es una forma de representar la página web de una manera jerárquica estructurada para que sea más fácil para los programadores y usuarios navegar por el documento. Con DOM, podemos acceder y manipular fácilmente etiquetas, ID, clases, atributos o elementos de HTML utilizando comandos o métodos proporcionados por el objeto del documento. Usando DOM, JavaScript obtiene acceso a HTML y CSS de la página web y también puede agregar comportamiento a los elementos HTML. Así que, básicamente, Document Object Model es una API que representa e interactúa con documentos HTML.



Árbol de nodos del DOM

```
<html>
   <head>
      <link href="style.css" rel="stylesheet">
      <title>Title</title>
   </head>
   <body>
      <p>Hallo Welt!</p>
      <div><img src="bild.jpg"></div>
   </body>
</html>
```

Documento HTML

```
paragraphs = document.getElementsByTagName ("p");
// paragraphs[0] es el primer elemento <p>
// paragraphs[1] es el segundo elemento <p>, etc.
alert (paragraphs [0].nodeName);
```

Documento HTML

Eventos

La interacción de JavaScript con HTML se maneja a través de eventos que ocurren cuando el usuario o el navegador manipula una página. Cuando se carga la página, se llama un evento. Cuando el usuario hace clic en un botón, ese clic también es un evento.

Evento	Descripción
onchange	Se ha cambiado un elemento HTML
onclick	El usuario hizo clic en un elemento HTML
onmouseover	El usuario movió el mouse sobre un elemento HTML
onmouseout	El usuario alejó el mouse de un elemento HTML
onkeydown	El usuario presiona una tecla del teclado
onload	El navegador ha terminado de cargar la página

Timers - Timeout

setTimeout() es un método de javascript que permite ejecutar un bloque de código después de transcurrido un tiempo determinado. Puede verse como un temporizador para ejecución de código.

```
setTimeout(() => {console.log("this is the first message")}, 5000);
setTimeout(() => {console.log("this is the second message")}, 3000);
setTimeout(() => {console.log("this is the third message")}, 1000);

// Output:

// this is the third message
// this is the second message
// this is the first message
```

Ejemplo método setTimeOut. (MDN)

Almacenamiento en localstorage

El mecanismo localStorage proporciona un tipo de objeto de almacenamiento web que permite almacenar y recuperar datos en el navegador. Es posible almacenar y acceder a los datos sin vencimiento; los datos estarán disponibles incluso después de que un usuario cierre el sitio. El acceso al localStorage se da usando JavaScript.

JavaScript.

```
localStorage.setItem('myCat', 'Tom');
```

Ejemplo set item con localstorage. (MDN)

```
const cat = localStorage.getItem('myCat');
```

Ejemplo get item con localstorage. (MDN)

```
localStorage.removeItem('myCat');
```

Ejemplo remove item con localstorage. (MDN)

```
localStorage.clear();
```

Ejemplo clear con localstorage. (MDN)

Fetch API

Fetch API permite consultar y/o modificar datos proporcionados por recursos externos como APIs.

```
fetch('http://example.com/movies.json')
  .then(response => response.json())
  .then(data => console.log(data));
```

Ejemplo remove item con localstorage. (MDN)

```
var url = 'https://example.com/profile';
var data = {username: 'example'};

fetch(url, {
  method: 'POST', // or 'PUT'
  body: JSON.stringify(data), // data can be `string` or {object}!
  headers:{
    'Content-Type': 'application/json'
  }
}).then(res => res.json())
.catch(error => console.error('Error:', error))
.then(response => console.log('Success:', response));
```

Ejemplo remove item con localstorage. (MDN)

5.5. API rest

API Rest

Una API de REST, o API de RESTful, es una interfaz de programación de aplicaciones (API o API web) que se ajusta a los límites de la arquitectura REST y permite la interacción con los servicios web de RESTful.

Características básicas de REST.

- Sin estado (stateless). La información del cliente no se almacena en el servidor, en cambio se almacena en el lado del cliente.
- Cliente / Servidor. Separa las responsabilidades del cliente (Front End) y el servidor (Back End) operando de forma independiente.
- Caché. Alguna información del lado del servidor se puede guardar en el cliente, para mejorar el rendimiento del sitio web.
- Métodos HTTP. REST utiliza métodos como get, put, delete y post con el fin de realizar ciertas acciones en el aplicativo.

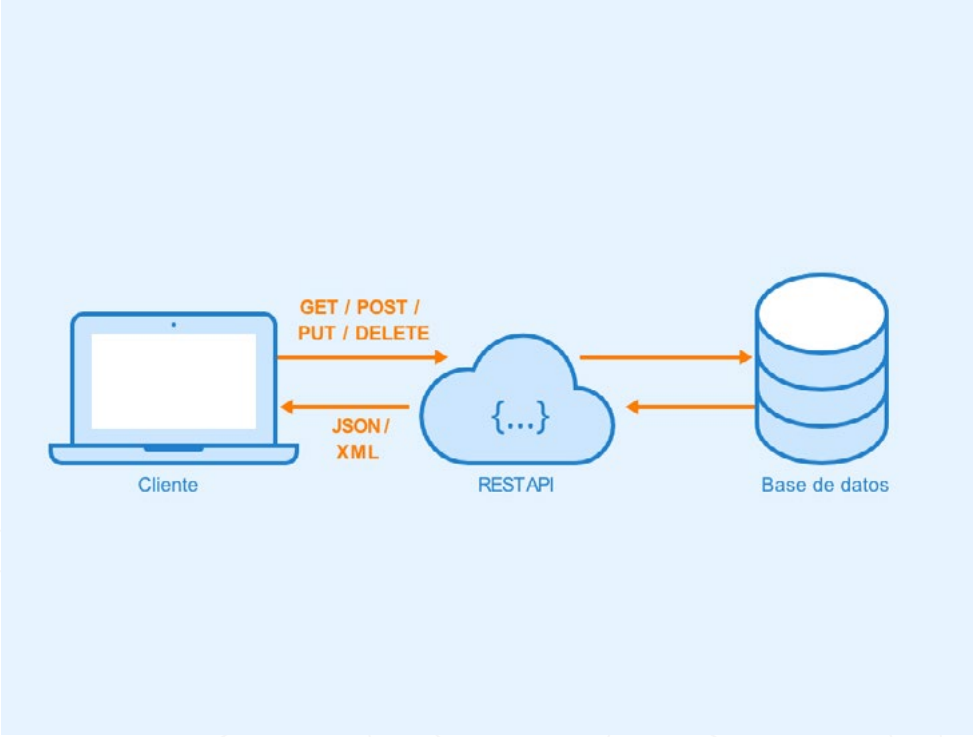


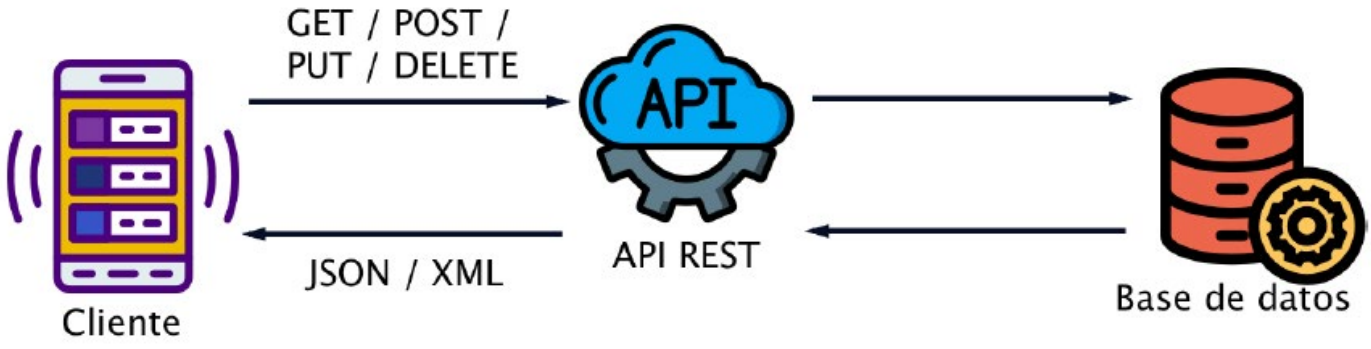
Figura 5. REST API. Fuente: https://www.seobility.net/es/wiki/API_REST.

Métodos HTTP.

- GET. Método utilizado para recuperar información. Devuelve una representación en XML o JSON y en código HTTP la respuesta OK (200) si todo sale bien o un error 404 o 400.
- POST. Método utilizado para crear nuevos recursos. Devuelve un estado HTTP 201 si todo sale bien en la ejecución.
- PUT. Método utilizado para actualizar o crear recursos. Devuelve un estado HTTP 200 si fue exitosa la actualización, o, un estado 201 si se crea con éxito el recurso.
- DELETE. Método utilizado para borrar recursos.

Las API son conjuntos de definiciones y protocolos que se utilizan para diseñar e integrar el software de las aplicaciones.Suele considerarse como el contrato entre el proveedor de información y el usuario, donde se establece el contenido que se necesita por parte del consumidor (la llamada) y el que requiere el productor (la respuesta).Por ejemplo, el diseño de una API de servicio meteorológico podría requerir que el usuario escribiera un código postal y que el productor diera una respuesta en dos partes: la primera sería la temperatura máxima y la segunda, la mínima.

En otras palabras, las API le permiten interactuar con una computadora o un sistema para obtener datos o ejecutar una función, de manera que el sistema comprenda la solicitud y la cumpla.



Cuando el cliente envía una solicitud a través de una API de RESTful, esta transfiere una representación del estado del recurso requerido a quien lo haya solicitado o al extremo. La información se entrega por medio de HTTP en uno de estos formatos: JSON (JavaScript Object Notation), HTML, XLT, Python, PHP o texto sin formato. JSON es el lenguaje de programación más popular, ya que tanto las máquinas como las personas lo pueden comprender y no depende de ningún lenguaje, a pesar de que su nombre indique lo contrario.

Algunas API, como SOAP o XML-RPC, imponen una estructura estricta a los desarrolladores. Sin embargo, las API REST se pueden desarrollar mediante el uso de prácticamente cualquier lenguaje de programación y son compatibles con una variedad de formatos de datos. El único requisito es que se ajusten a los siguientes seis principios de diseño de REST, también conocidos como restricciones de arquitectura:

Interfaz uniforme: Todas las solicitudes de API para el mismo recurso deben ser iguales, independientemente de la procedencia de la solicitud. La API REST debe asegurarse de que el mismo dato, como el nombre o la dirección de e-mail de un usuario, pertenezca a un único identificador uniforme de recurso (URI). Los recursos no deben ser demasiado grandes, sin embargo, deben contener toda la información que el cliente pueda necesitar.

Sin estado: Las API REST son sin estado, lo que significa que cada solicitud debe incluir toda la información necesaria para procesarla. En otras palabras, las API REST no requieren ninguna sesión en el lado del servidor. Las aplicaciones de servidor no pueden almacenar ningún dato relacionado con una solicitud de cliente.

Separación entre cliente y servidor: En el diseño de API REST, las aplicaciones de cliente y de servidor deben ser completamente independientes entre sí. La única información que la aplicación de cliente debe conocer es el URI del recurso solicitado. No puede interactuar con la aplicación de servidor de ninguna otra forma. Del mismo modo, una aplicación de servidor no debe modificar la aplicación de cliente más allá de entregarle los datos solicitados vía HTTP.

Arquitectura de sistema de capas: En las API REST, las llamadas y respuestas pasan por diferentes capas. Como regla general, no debe suponer que las aplicaciones de cliente y de servidor se conectan directamente entre sí. Puede haber una serie de intermediarios diferentes en el bucle de comunicación. Las API REST deben diseñarse para que ni el cliente ni el servidor puedan notar si se comunican con la aplicación final o con un intermediario.

Capacidad de almacenamiento en memoria caché: Siempre que sea posible, los recursos deben poder almacenarse en la memoria caché en el lado del cliente o el servidor. Las respuestas de servidor también necesitan contener información de si el almacenamiento en memoria caché está permitido para el recurso entregado. El objetivo es mejorar el rendimiento en el lado del cliente, al mismo tiempo que aumenta la escalabilidad en el lado del servidor.

Funcionamiento de las API REST

Las API REST se comunican a través de solicitudes HTTP para realizar funciones estándar de base de datos, como crear, leer, actualizar y suprimir registros (también conocidos como CRUD) dentro de un recurso. Por ejemplo, una API REST utilizará una solicitud GET para recuperar un registro, una solicitud POST para crearlo, una solicitud PUT para actualizarlo y una solicitud DELETE para suprimirlo. Todos los métodos HTTP se pueden utilizar en llamadas API. Una API REST bien diseñada es similar a un sitio web que se ejecuta en un navegador web con funcionalidad HTTP incorporada.

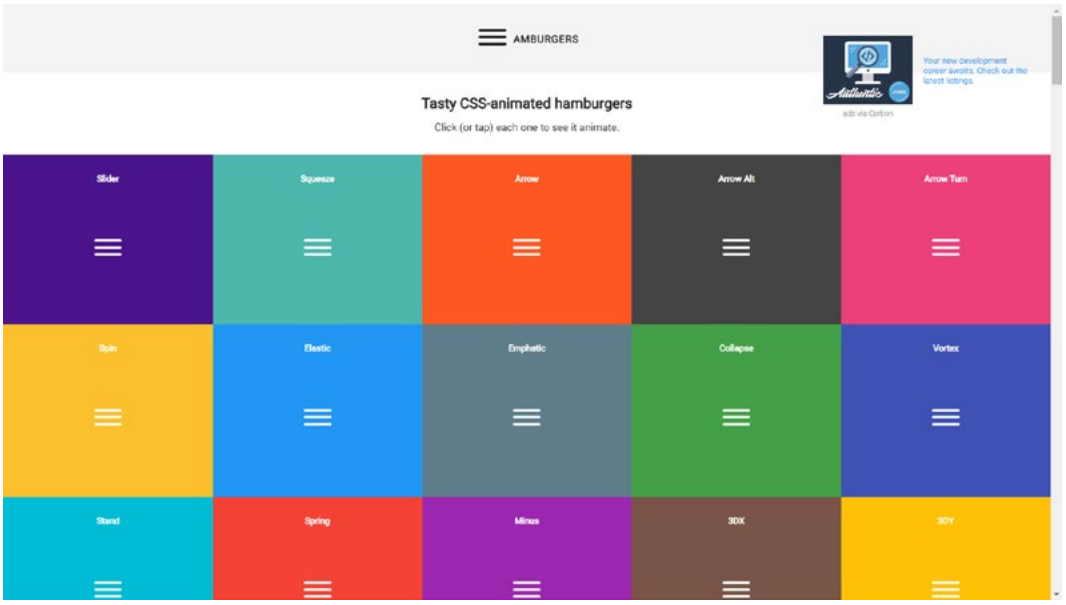
El estado de un recurso en un instante específico, o en una indicación de fecha y hora, se conoce como la representación de recursos. Esta información se puede entregar a un cliente en prácticamente cualquier formato, entre ellos JavaScript Object Notation (JSON), HTML, XLT, Python, PHP o un texto sin formato. JSON es popular debido a que es legible tanto por los seres humanos como por las máquinas y debido a que es independiente de un lenguaje de programación.

Las cabeceras y parámetros de solicitud también son importantes en las llamadas de API REST, puesto que incluyen información de identificador importante, como metadatos, autorizaciones, identificadores de recursos uniformes (URI), almacenamiento en memoria caché, cookies y más. Las cabeceras de solicitud y las cabeceras de respuesta, junto con los códigos de estado HTTP convencionales, se utilizan en las API REST bien diseñadas.

5.6. Framework y librerías para el FrontEnd.

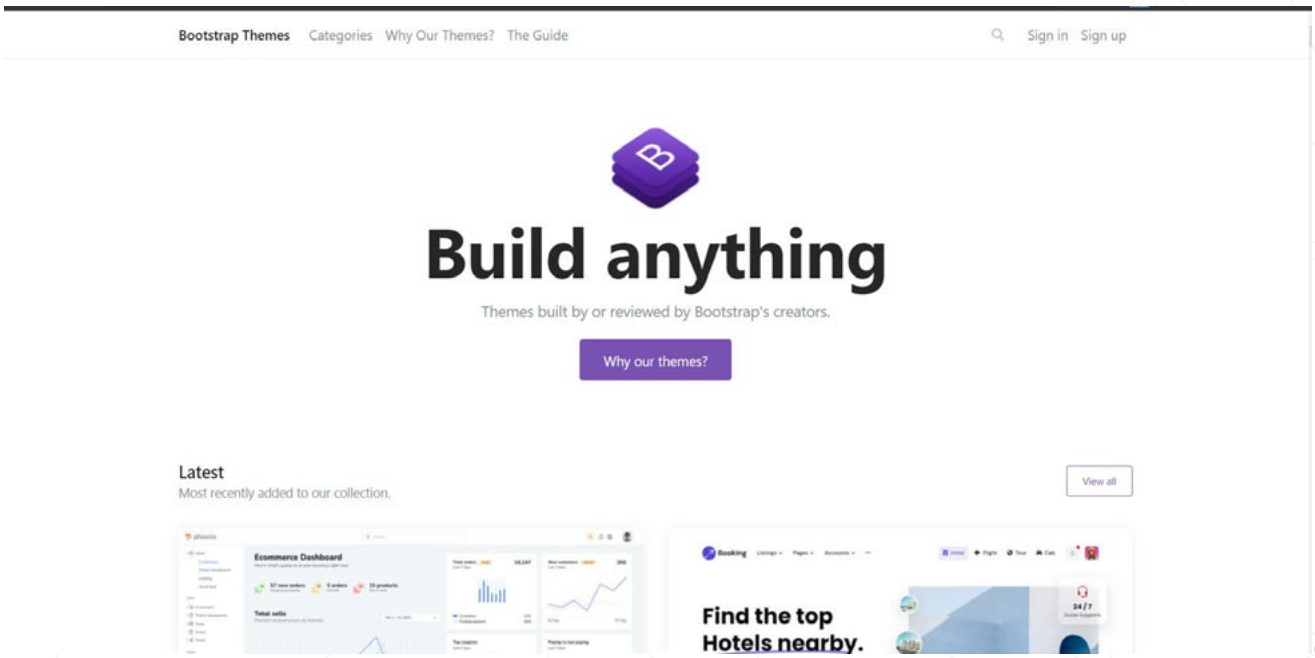
Framework y librerías para el FrontEnd.

Hamburgers: Es una de las librerías más utilizadas para el uso de menús hamburguesa dentro de los sitios web.
https://github.com/jonsuh/hamburgers

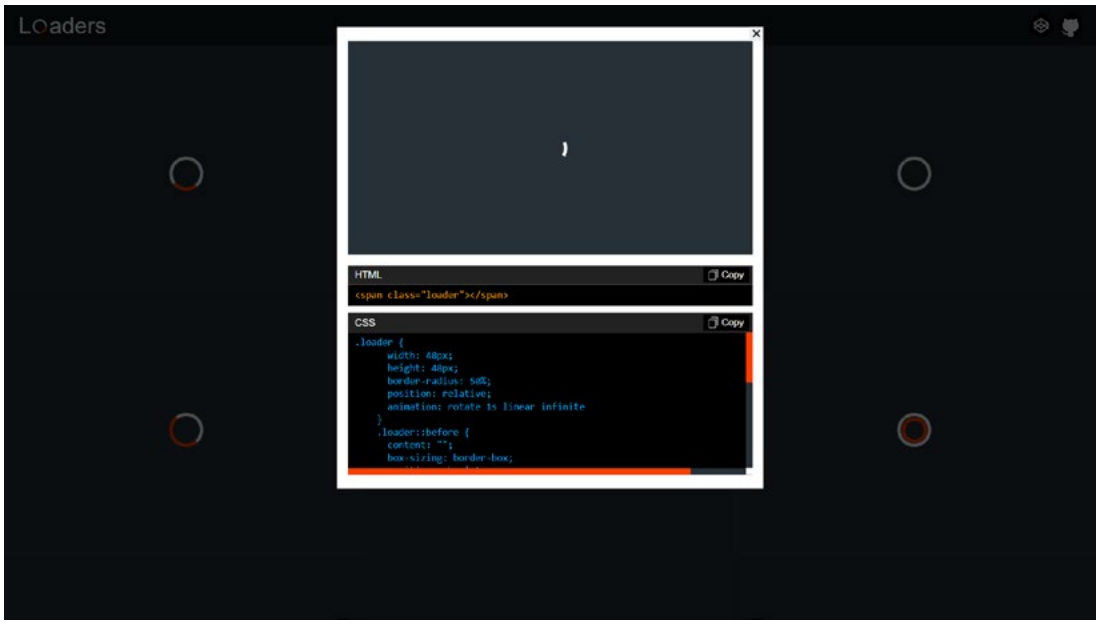


BootStrap: Es una biblioteca de diversas herramientas de código abierto el cual sirve para el diseño de sitios web y aplicaciones web. Contiene diversa cantidad de herramientas como formularios, botones, tipografías, iconos, menús de navegación, entre otros.

https://getbootstrap.com/



Loaders: Es una de las librerías más utilizadas para el uso de pre-carga dentro de los sitios web.



FontAwesome: FontAwesome tiene un conjunto de herramientas de fuentes e iconos, los cuales son fácilmente implementados en HTML, tiene una de las gamas más grandes de iconos para uso web.

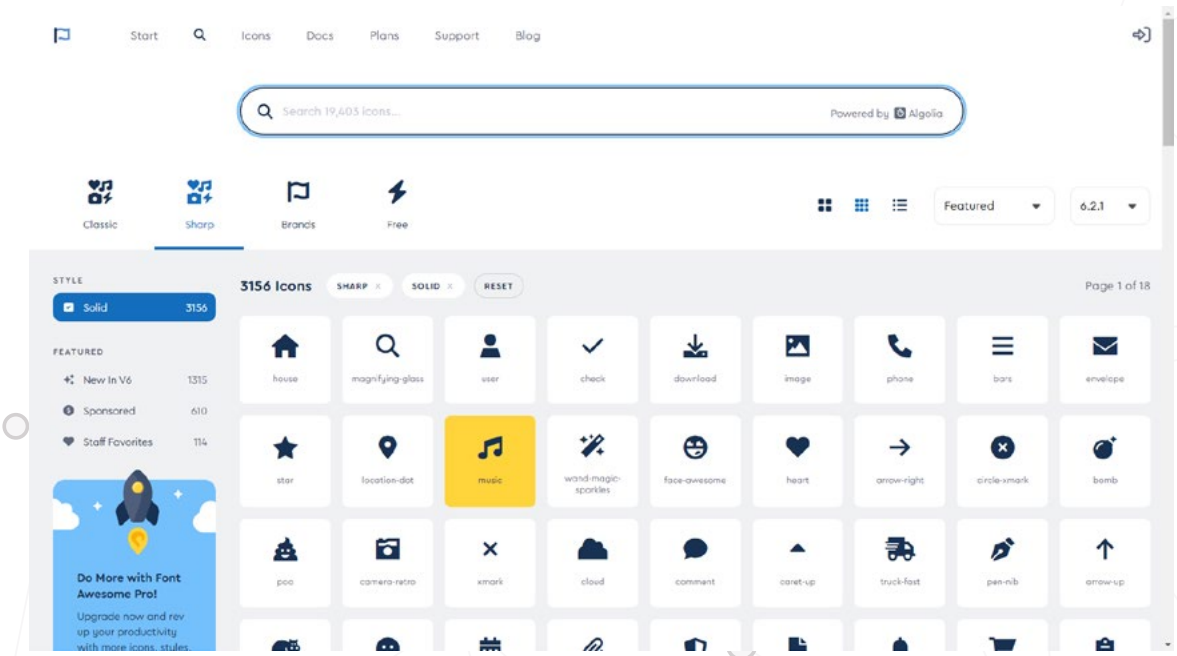
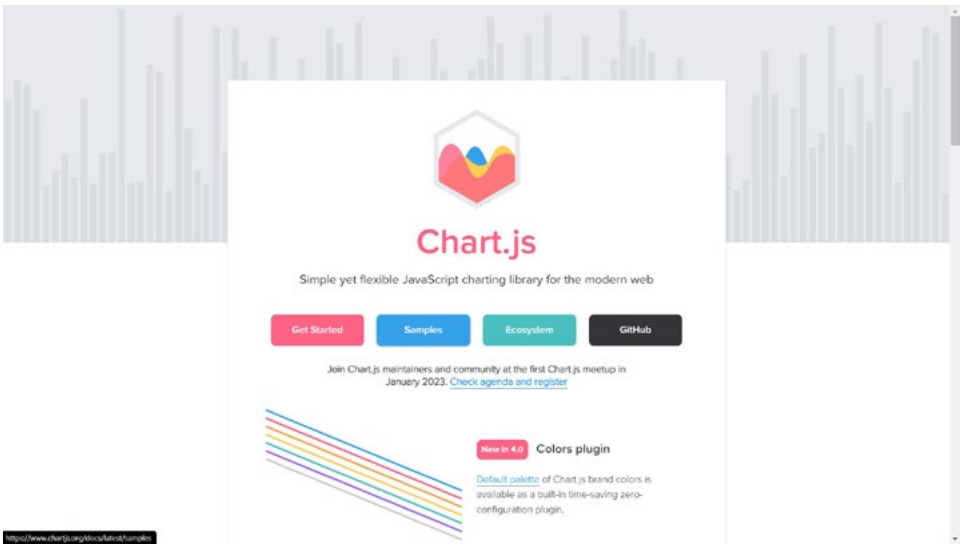
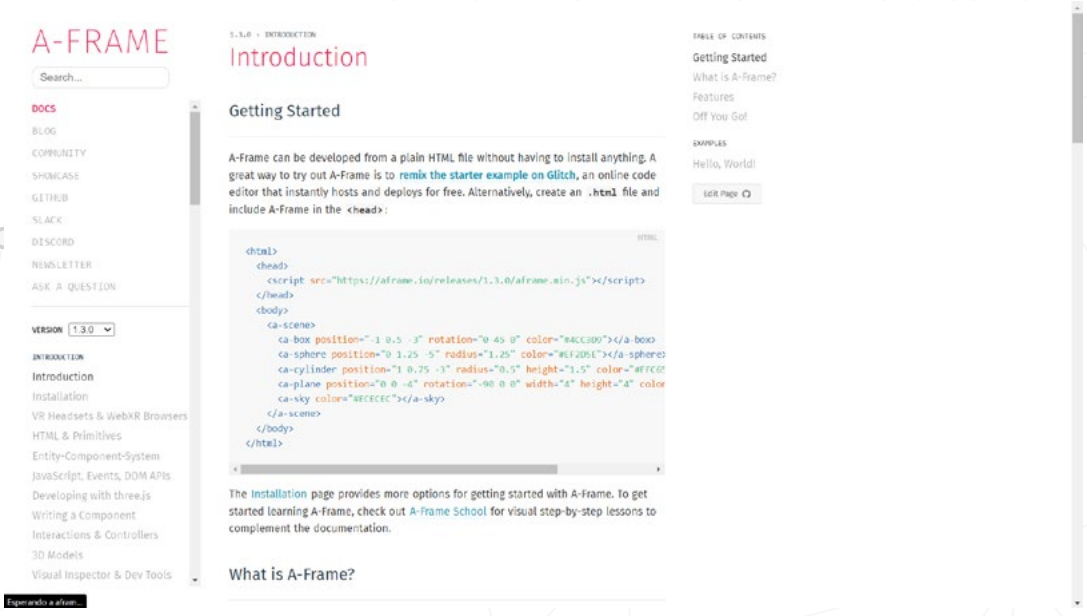


Chart JS: Es una librería de JavaScript, la cual permite implementar dentro de cualquier sitio web, diversos tipos de gráficas.



Aframe: Es un marco web para crear experiencias de realidad virtual (VR). A-Frame se basa en la parte superior de HTML, por lo que es fácil comenzar. Pero A-Frame no es solo un gráfico de escena 3D o un lenguaje de marcado; el núcleo es un poderoso mar-co de Esta obra está bajo una licencia: CC BY-NC-ND Atribución – No comercial – Sin derivar Consultar información relacionada en: Atribución – No comercial – Sin derivar entidad-componente que proporciona una estructura declarativa, extensible y compo-nible para three.js.

Originalmente concebido dentro de Mozilla y ahora mantenido por los co-creadores de A-Frame dentro de Supermedium, A-Frame fue desarrollado para ser una forma fácil pero poderosa de desarrollar contenido de realidad virtual. Como proyecto inde-pendiente de código abierto, A-Frame se ha convertido en una de las comunidades de realidad virtual más grandes.



JQuery: jQuery es una biblioteca de JavaScript rápida, pequeña y rica en funciones. Hace que cosas como el recorrido y la manipulación de documentos HTML, el manejo de eventos, la animación y Ajax sean mucho más simples con una API fácil de usar que funciona en una multitud de navegadores. Con una combinación de versatilidad y extensibilidad, jQuery ha cambiado la forma en que millones de personas escriben JavaScript.

Tablas dinámicas.

Las tablas son uno de los elementos más antiguos del lenguaje HTML que se ha mantenido a lo largo de los años. Incluso se llegó a emplear tanto para organizar contenido como para crear una layout. Por supuesto, esta es una práctica que ya no se emplea pues existen propiedades CSS que permiten la creación de layouts responsive. Sin embargo, las tablas se siguen empleando en la web para un único propósito: mostrar información de forma organizada.

Dynatable

Dynatable es una tabla interactiva que hace uso de jQuery, HTML5 y JSON. Con Dynatble puedes crear tablas que admiten características como paginación, clasificación y filtro de elementos.

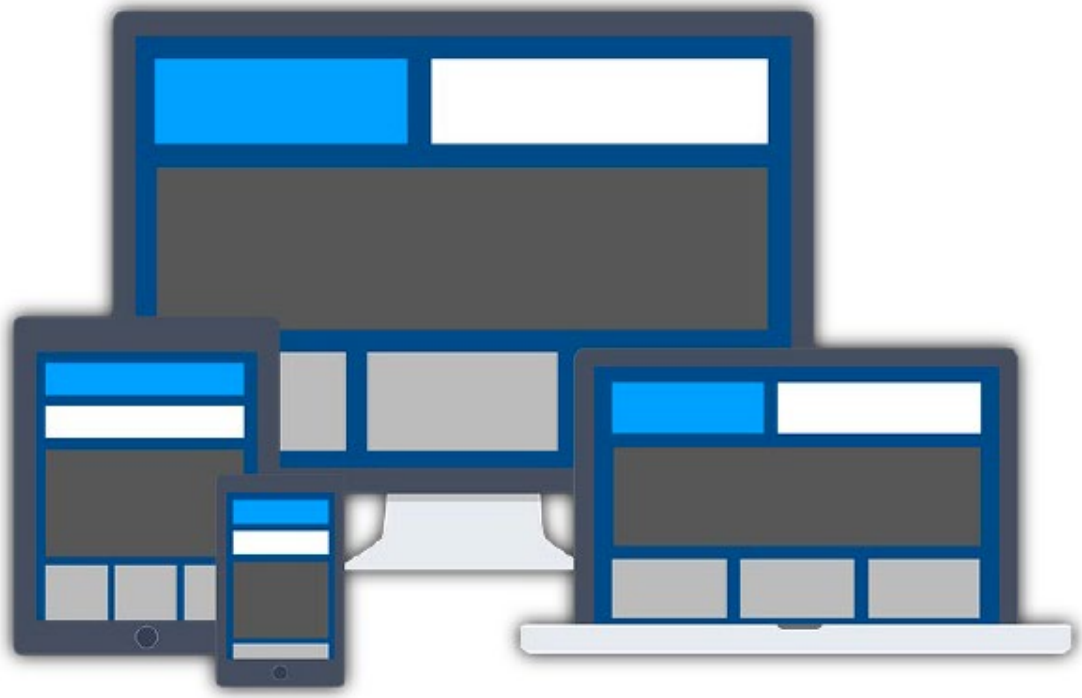
Dynatable escanea tablas HTML y estandariza los datos en elementos JSON. Al hacer este proceso cada colección JSON puede ser clasificada. Por último, renderiza los resultados en el DOM en la tabla respectiva.

Dynatable es un poderoso plugin dirigido a desarrolladores experimentados pues te da un amplio control sobre la renderización y operaciones de búsqueda.

5.7. Responsive design aplicación en sitios web

Responsive design es el enfoque que sugiere que el diseño y el desarrollo deben responder al comportamiento y al entorno del usuario en función del tamaño de la pantalla, la plataforma y la orientación.

Las páginas web se pueden ver usando muchos dispositivos diferentes: computadoras de escritorio, tabletas y teléfonos. El sitio web debe verse bien y ser fácil de usar, independientemente del dispositivo. Las páginas web no deben dejar de lado la información para adaptarse a dispositivos más pequeños, sino adaptar su contenido para adaptarse a cualquier dispositivo:



Uso de media Queries para el responsive design

Las reglas media queries (también denominadas MQ a veces) son un tipo de reglas de CSS que permiten crear un bloque de código que sólo se procesa en los dispositivos que cumplan los criterios especificados como condición:

```
@media screen and (*condición*) {
  /* reglas CSS */
  /* reglas CSS */
}

@media screen and not (*condición*) {
  /* reglas CSS */
  /* reglas CSS */
}
```

Con este método, especificamos que queremos aplicar los estilos CSS para tipos de medios concretos (screen: sólo en pantallas, en este caso) que cumplan las condiciones especificadas entre paréntesis. De esta forma, una estrategia aconsejable es crear reglas CSS generales aplicadas a todo el documento: colores, tipo de fuente, etc. y luego, las particularidades que se aplicarían sólo en el dispositivo en cuestión.

Aunque suele ser menos habitual, también se pueden indicar reglas @media negadas mediante la palabra clave not, que aplicará CSS siempre y cuando no se cumpla una determinada condición. Tambien pueden separarse por comas varias condiciones de media queries.

Tipo de medio	Significado
screen	Monitores o pantallas de ordenador. Es el más común.
print	Documentos de medios impresos o pantallas de previsualización de impresión.
speech	Lectores de texto para invidentes (Antes **aural**, el cuál ya está obsoleto).
all	Todos los dispositivos o medios. El que se utiliza **por defecto**.

5.8. Introducción manejo de Realidad aumentada y objetos 3D

Introducción manejo de Realidad aumentada y objetos 3D

Realidad Aumentada es un recurso tecnológico que ofrece experiencias interactivas al usuario a partir de la combinación entre la dimensión virtual y la física, con la utilización de dispositivos digitales.

La Realidad Aumentada (RA) asigna la interacción entre ambientes virtuales y el mundo físico, posibilitando que ambos se entremezclan a través de un dispositivo tecnológico como webcams, teléfonos móviles (IOS o Android), tabletas, entre otros.

En otras palabras, la RA inserta objetos virtuales en el contexto físico y se los muestra al usuario usando la interfaz del ambiente real con el apoyo de la tecnología. Este recurso

viene revolucionando la forma en que lidiamos con nuestras tareas (e incluso, las que les asignamos a las máquinas).

De ese modo, podemos afirmar que la Realidad Aumentada se caracteriza por:

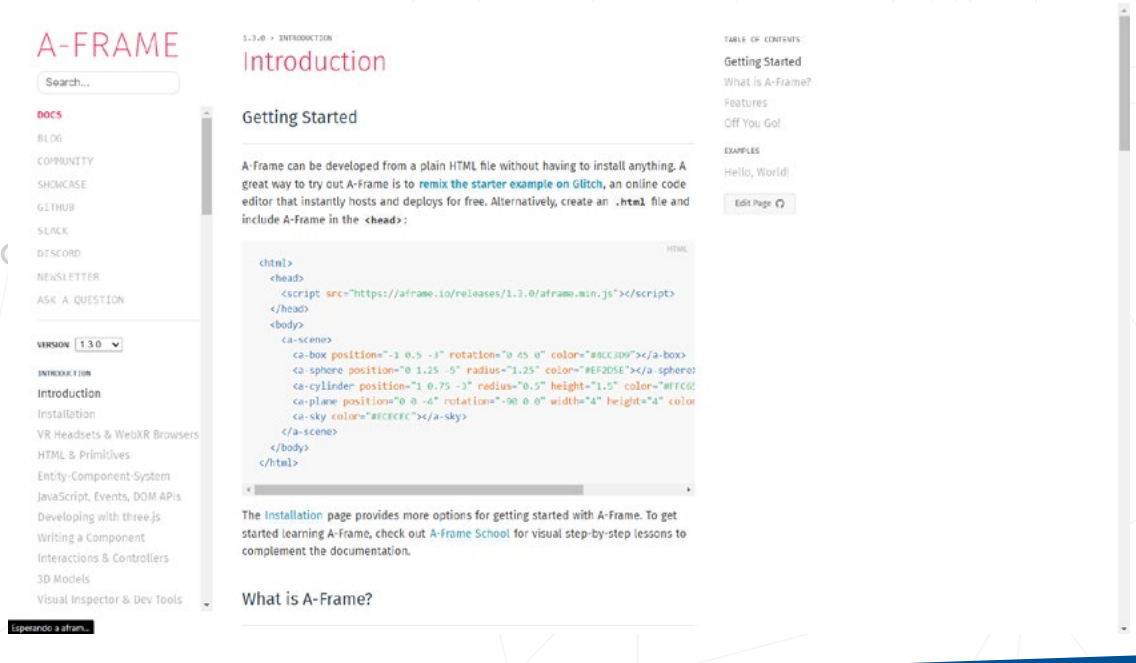
- combinar el mundo real y el virtual.
- Ofrecer una interacción en tiempo real.
- interactuar con todas las capacidades físicas del entorno (en tres dimensiones).

¿Has notado que nos referimos a la Realidad Virtual como parte integrante de la Realidad Aumentada? Eso porque RA y RV son dos conceptos distintos, aunque comúnmente confundidos, y que en muchos casos trabajan de forma conjunta.
Herramientas para el uso de realidad aumentada en sitios web.

Aframe

Es un marco web para crear experiencias de realidad virtual (VR). A-Frame se basa en la parte superior de HTML, por lo que es fácil comenzar. Pero A-Frame no es solo un gráfico de escena 3D o un lenguaje de marcado; el núcleo es un poderoso marco de entidad-componente que proporciona una estructura declarativa, extensible y componible para three.js.

Originalmente concebido dentro de Mozilla y ahora mantenido por los co-creadores de A-Frame dentro de Supermedium, A-Frame fue desarrollado para ser una forma fácil pero poderosa de desarrollar contenido de realidad virtual. Como proyecto independiente de código abierto, A-Frame se ha convertido en una de las comunidades de realidad virtual más grandes.

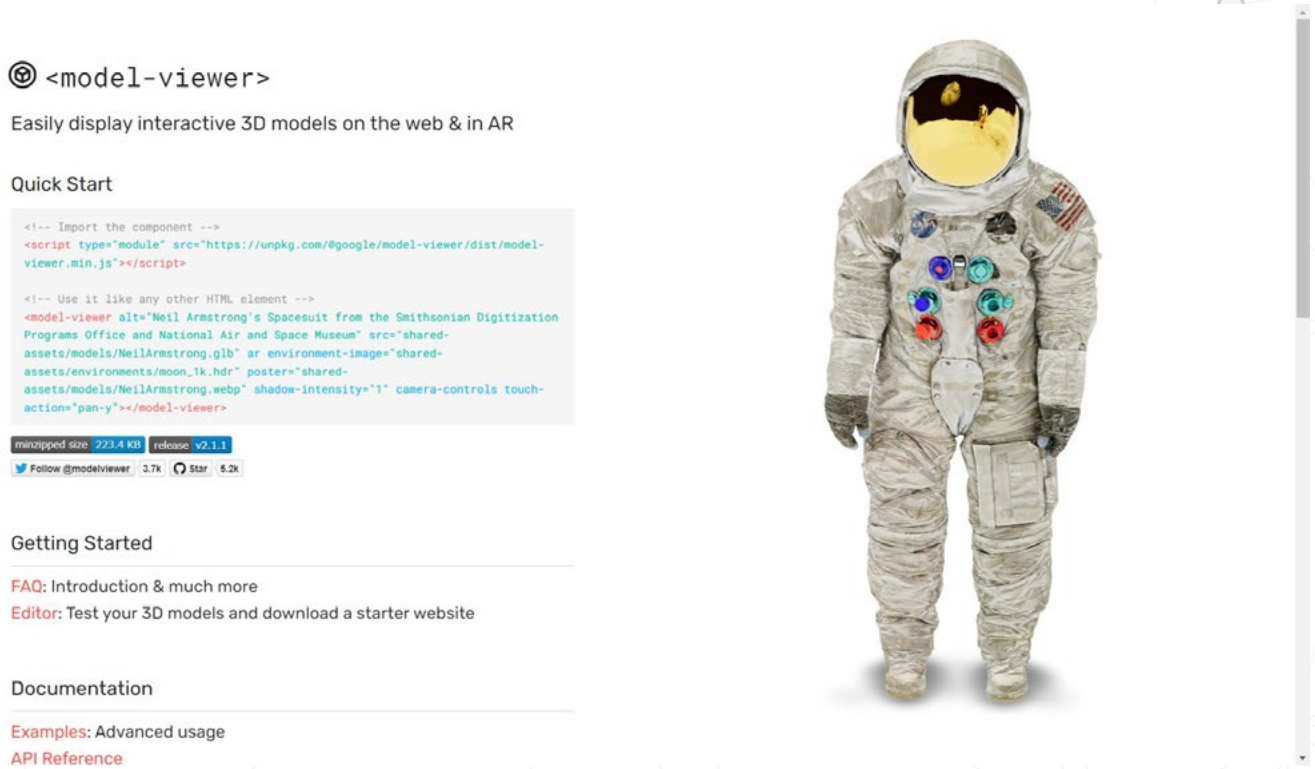


<model-viewer>

Es una librería que permite implementar diversos objetos interactivos 3D o modelos de realidad aumentada.

Para su funcionamiento se debe llamar el código en el head como script:

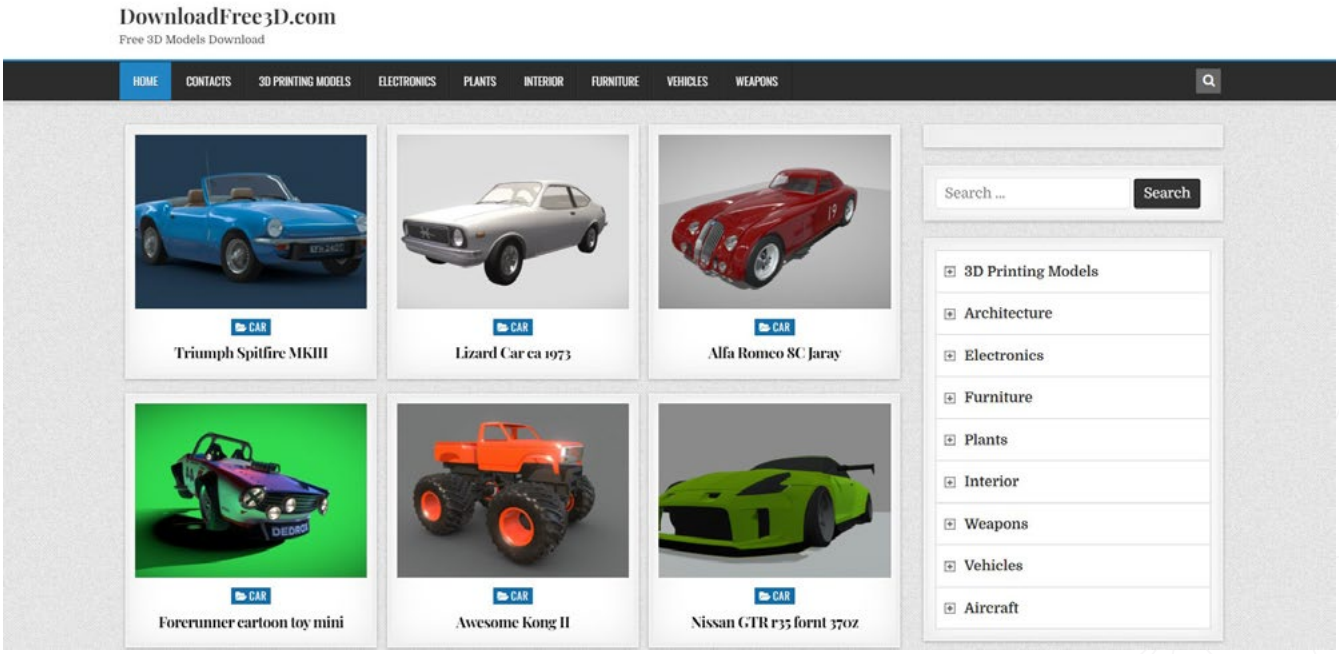
```
<!-- Import the component -->
<script type="module" src="https://unpkg.com/@google/model-viewer/dist/model-viewer.min.js"></script>

<!-- Use it like any other HTML element -->
<model-viewer alt="Neil Armstrong's Spacesuit from the Smithsonian Digitization Programs Office and National Air and Space Museum" src="shared-assets/models/NeilArmstrong.glb" ar environment-image="shared-assets/environments/moon_1k.hdr" poster="shared-assets/models/NeilArmstrong.webp" shadow-intensity="1" camera-controls touch-action="pan-y"></model-viewer>
```



Objetos de recursos 3D.

Para la utilización de las herramientas 3D con base en la realidad aumentada, se deben utilizar diversos recursos, los cuales se pueden conseguir gratis en la siguiente página:

https://downloadfree3d.com/



Permite descargar diversos modelos 3D para su libre utilización, con uso educativo y personal.

5.9. TypeScript

Historia

Fue desarrollado en el 2012 por microsoft como una solución a las diferentes carencias que tenía javascript en ese momento como era la ausencia de clases, módulos y herramientas que optimizaran el flujo de desarrollo. La motivación para crear TypeScript fue mejorar la experiencia de desarrollo de aplicaciones de JavaScript a gran escala mediante la adición de funciones como tipos estáticos, clases y módulos. Estas funciones no están disponibles en JavaScript y pueden ayudar a mejorar la calidad, el mantenimiento y la escalabilidad del código.

TypeScript se usa a menudo en el desarrollo de aplicaciones web basadas en JavaScript, especialmente aquellas que usan frameworks populares como Angular y React. También se utiliza para crear aplicaciones multiplataforma utilizando tecnologías como Electron y Node.js.

Se puede considerar como la solución a muchos de los problemas de JavaScript y que está pensado para el desarrollo de aplicaciones robustas, implementando características en el lenguaje que nos permitan desarrollar herramientas más avanzadas para el desarrollo de aplicaciones.

Línea de tiempo:

- TypeScript tuvo su aparición de manera pública en su versión 0.8 en octubre de 2012, después que Microsoft termino de Desarrollarlo durante 2 años.
- La versión 0.9 de TypeScript fue lanzada en 2013, esta versión incluye soporte para genéricos.
- La siguiente versión 1.0 de TypeScript fue lanzada en el año 2014 durante la conferencia para Desarrolladores llamada Build, creada por Microsoft.
- A mitad del año 2014, el equipo de Desarrolladores de TypeScript anunció un compilador nuevo y veloz, era 5 veces más rápido que el compilador actual.
- En el año 2016 se lanza la versión 2.0 de TypeScript, en esta versión se agregaron varias características notables, una de ellas es la capacidad de evitar la asignación a nulo de las variables.

Definición

Typescript es un superset de JavaScript. Un superset de un lenguaje de programación se define como aquel que, cuando puede ejecutar programas de la tecnología, Typescript en este caso, y del lenguaje del que es el superset, JavaScript en este mismo ejemplo. En resumen, esto significa que los programas de JavaScript son programas válidos de TypeScript, a pesar de que TypeScript sea otro lenguaje de programación. Fue creado como un superset para que las mejoras que salen constantemente de javacsript también estuviesen en él, permitiendo así que typescript este a la vanguardia ante las nuevas versiones de Ecmascript. Esto permite además que typescript se pueda integrar en proyectos ya existentes que utilicen javascript sin necesidad de reescribir todo el código, incluso es muy común encontrar proyectos que utilizan typescript a la par de javascript según las necesidades.

Por si fuera poco, uno de los beneficios adicionales de esta característica del lenguaje, es que pone a disposición el enorme ecosistema de librerías y frameworks que existen para JavaScript como React,Vue, Angular etc.

Sintaxis

TypeScript es un superconjunto de JavaScript, por lo que incluye toda la sintaxis de JavaScript y agrega algunas funciones adicionales.
Aquí hay una breve descripción de algunos de los principales elementos de sintaxis de TypeScript:

Tipos: TypeScript introduce tipos estáticos en JavaScript, lo que le permite especificar el tipo de una variable, función o propiedad de objeto. Por ejemplo, puede declarar una variable como un número, una cadena o un valor booleano utilizando la sintaxis let num: number, let str: string o let bool: boolean.

Interfaces: una interface en TypeScript define un contrato para un objeto o función, especificando la forma y los tipos de sus propiedades y parámetros. Por ejemplo, podría definir una interface para un objeto de usuario con propiedades como id, nombre y correo electrónico, y luego usar esa interface para escribir una variable que represente a un usuario.

Clases: TypeScript admite la programación orientada a objetos basada en clases, con una sintaxis similar a la de otros lenguajes como Java o C#. Puede definir una clase con propiedades y métodos, y usar herencia y polimorfismo para crear una jerarquía de clases relacionadas.

Módulos: TypeScript admite el uso de módulos para organizar y reutilizar código. Un módulo es una unidad de código independiente que se puede importar y usar en otras partes de su aplicación.

Tipos e inferencia de tipos: TypeScript incluye una serie de tipos integrados, como números, cadenas y booleanos, así como soporte para definir tipos personalizados. También cuenta con un sistema de inferencia de tipos que puede inferir automáticamente los tipos de variables y expresiones en función del contexto en el que se utilizan.

Tipos

Los tipos en TypeScript son una forma de asegurar que una variable tenga un cierto tipo de datos. Al especificar el tipo de una variable, TypeScript puede realizar comprobaciones de tipo en tiempo de compilación y ayudar a detectar errores en el código.

En TypeScript, los tipos se dividen en dos categorías: tipos primitivos y tipos de referencia. Los tipos primitivos incluyen números, cadenas, booleanos y algunos otros tipos más simples. Los tipos de referencia incluyen objetos, arreglos y tipos personalizados que se pueden definir en el código.

Además de los tipos predefinidos, también se pueden crear tipos personalizados mediante el uso de interfaces o tipos de alias. Esto puede ser útil para crear tipos más complejos o para dar nombres a tipos existentes.

Primitivos

JavaScript tiene tres tipos primitivos de uso muy común: string, number y boolean. Cada uno tiene un tipo correspondiente en TypeScript. Estos son los mismos tipos/nombres que retorna el operador typeof de JavaScript si se usara sobre diferentes valores:

- string representa valores de cadena como “Hola, mundo”
- number es para números como 42. JavaScript no tiene un valor de tiempo de ejecución especial para números enteros, por lo que no hay equivalente a int o float: todo es simplemente number
- boolean es para los dos valores verdadero y falso.

Es importante resaltar que los nombres de tipo String, Number y Boolean (que comienzan con letras mayúsculas) son legales, pero se refieren a algunos tipos integrados especiales que muy raramente aparecerán en su código. Utilice siempre cadena, número o booleano para los tipos.

Any

TypeScript también tiene un tipo especial, Any, que puede usar siempre que no se desee que un valor en particular cause errores de verificación de tipos. Cuando un valor es del tipo Any, se puede acceder a cualquier propiedad del mismo (que a su vez será del tipo cualquiera), llamarlo como una función, asignarlo a (o desde) un valor de cualquier tipo. A continuación, se presenta un ejemplo de una variable declarada con el tipo any, como puede ver a la variable x se le pueden asignar valores de diferentes tipos, como string, number y boolean0. Esto se debe a que tiene cualquier tipo, lo que le permite tener cualquier valor.:

```
let x: any;

x = 'hello';  // x is a string
console.log(x);

x = 42;  // x is a number
console.log(x);

x = false;  // x is a boolean
console.log(x);
```

El tipo Any se usa a menudo cuando no se conoce el tipo de un valor o cuando se trabaja con valores que provienen de fuentes externas, como datos JSON o entradas de usuario. También puede ser útil cuando desea deshabilitar la verificación de tipos para una parte específica de código. Sin embargo, generalmente se recomienda usar tipos más específicos siempre que sea posible, para obtener los beneficios de la verificación de tipos y evitar posibles errores. El uso de cualquier tipo con demasiada frecuencia puede hacer que su código sea menos seguro y menos predecible.

Null y Undefined

JavaScript tiene dos valores primitivos que se usan para señalar la ausencia de un valor ausente o que no está inicializado: null y undefined. Estos tipos también existen en typescript y su comportamiento depende de la configuración de la opción strictNullChecks.

- Cuando el strictNullChecks desactivado es posible acceder normalmente a los valores null y undefined y los valores null o undefined pueden ser asignados a una propiedad de cualquier tipo, no es recomendado utilizar este modo porque esta falta de verificación de valores tiende a ser una fuente importante de errores.
- Con strictNullChecks activado, cuando un valor es nulo o indefinido se deberán probar esos valores antes de usar métodos o propiedades sobre él. Al igual que al verificar si el valor no está definido antes de usar una propiedad opcional es posible utilizar estrechamiento para verificar que valores podrían ser nulos de la siguiente manera:

```
function doSomething(x: string | null) {
  if (x === null) {
    // do nothing
  } else {
    console.log("Hello, " + x.toUpperCase());
  }
}
```

Arreglos y funciones
Arreglos

En TypeScript, un arreglo es una colección de elementos, de la misma manera que en javascript, con la diferencia que se debe declarar el tipo de elementos que contendrá. Hay dos formas de declarar un arreglo en TypeScript: usando corchetes o el constructor de arreglos. Aquí hay un ejemplo de cómo declarar un arreglo usando corchetes:

```
let numbers: number[] = [1, 2, 3, 4, 5];
```

Esto crea una matriz de números con cinco elementos. El tipo de matriz se infiere como number[], lo que significa que es una matriz de números.
También puede usar el constructor Array para crear una matriz:

```
let names: string[] = new Array<string>('Alice', 'Bob', 'Charlie');
```

Esto crea una matriz de cadenas con tres elementos. El tipo de matriz se especifica explícitamente como string[]. En ambos casos, los elementos del arreglo son de tipo number o string, respectivamente. Puede acceder a los elementos de la matriz utilizando la notación de índice, al igual que en JavaScript. Por ejemplo, numbers [0] sería 1 y names[1] sería 'Bob'.

También puede usar el tipo Array para crear una matriz de cualquier tipo:

```
let values: Array<any> = [1, 'two', true];
```

Esto crea una matriz que puede contener elementos de cualquier tipo. TypeScript también incluye varios métodos de matriz, como push, pop, shift y unshift, que puede usar para manipular los elementos de una matriz.

3.4.2 Funciones

Las funciones son el medio principal para pasar datos en JavaScript. En TypeScript se diferencian porque permiten especificar los tipos de los valores de entrada y salida de las funciones.

Cuando se declara una función, se pueden agregar anotaciones de tipo después de cada parámetro para declarar qué tipos de parámetros acepta la función. Las anotaciones de tipo parámetro van después del nombre del parámetro como se ve a continuación:

```
// Parameter type annotation
function greet(name: string) {
  console.log("Hello, " + name.toUpperCase() + "!!");
}
```

Cuando un parámetro tiene un tipo, se comprobarán los argumentos de esa función:

```
// Would be a runtime error if executed!
greet(42);
Argument of type 'number' is not assignable to parameter of type 'string'.
```

Una función también puede devolver un elemento de algún tipo. Las devoluciones de tipo se escriben luego de la lista de parámetros de la siguiente manera:

```
function getFavoriteNumber(): number {
  return 26;
}
```

Al igual que las anotaciones de tipo de variable, normalmente no se necesita especificar una anotación de tipo en el retorno de la función porque TypeScript deduce el tipo basado en las declaraciones de lo retornado.

Funciones anónimas

Las funciones anónimas son un poco diferentes de las funciones comunes. Cuando aparece una función en un lugar donde TypeScript puede determinar cómo se va a llamar, los parámetros de esa función reciben automáticamente tipos. Por ejemplo:

```
// No type annotations here, but TypeScript can spot the bug
const names = ["Alice", "Bob", "Eve"];

// Contextual typing for function
names.forEach(function (s) {
  console.log(s.toUppercase());
Property 'toUppercase' does not exist on type 'string'. Did you mean 'toUpperCase'?
});

// Contextual typing also applies to arrow functions
names.forEach((s) => {
  console.log(s.toUppercase());
Property 'toUppercase' does not exist on type 'string'. Did you mean 'toUpperCase'?
});
```

Aunque el parámetro s no tenía una anotación de tipo, TypeScript usó los tipos de la función forEach, junto con el tipo inferido del arreglo para determinar el tipo que tendrá.

Objetos y tipos personalizados

Objetos

Estos se refieren a cualquier valor de JavaScript con propiedades. Para definir un objeto en typescript simplemente se enumeran sus propiedades y sus tipos. Por ejemplo:

```
const car: { type: string, model: string, year: number } = {
  type: "Toyota",
  model: "Corolla",
  year: 2009
};
```

Cabe resaltar que al igual que en otros tipos si no especifican los tipos en los elementos de un objeto estos por defecto se tomaran como Any.

En TypeScript también puedes usar la sintaxis de tipos de objeto para especificar qué propiedades y tipos de datos debe tener un objeto. Además se pueden especificar si algunas o todas sus propiedades son opcionales. Para hacer esto, agrega un ? después del nombre de la propiedad, por ejemplo:

```
type User = {
  name: string;
  age: number;
  address?: string; // la propiedad es opcional
};

const user: User = {
  name: "Bob",
  age: 35
};
```

3.5.2 Tipos Personalizados

En TypeScript puedes crear tus propios tipos personalizados para darle un nombre a un conjunto de valores o para darle un nombre a un tipo de datos compuesto. Hay varias formas de crear tipos personalizados en TypeScript estos son:

Interfaces: Las interfaces son una forma de definir un contrato para un objeto o un tipo de datos compuesto. Puedes usar interfaces para especificar qué propiedades y tipos de datos deben tener un objeto o un tipo de datos compuesto.

Tipos de alias: Los tipos de alias son una forma de darle un nombre a un tipo de datos existente, se utiliza normalmente cuando. Por ejemplo:

```
type Age = number;

const age: Age = 30;
```

también es posible hacer que el alias de un tipo que se cree se una unión, por ejemplo:

```
type ID = number | string;
```

En el caso anterior se crea un alias para tipos de datos id o number, esto facilita el uso de múltiples asignaciones de tipo que se necesitan en más de una ocasión dentro de nuestro código.
Enumeraciones: Las enumeraciones son una forma de definir un conjunto de valores constantes. Por ejemplo:

```
enum Color {
  Red,
  Green,
  Blue
}

const color: Color = Color.Red;
```

Tipos de unión: Los tipos de unión son una forma de definir un tipo que puede ser uno de varios valores posibles. Por ejemplo:

```
type Shape = "circle" | "square" | "triangle";

const shape: Shape = "circle";
```

Los tipos de unión tienen un par de particularidades que desglosaremos a continuación:
TypeScript solo permitirá una operación si es válida para todos los miembros de la unión. Por ejemplo, si se tiene una unión con las opciones string | number, no se podrán utilizar métodos que solo están disponibles para un string o number, para visualizarlo un poco mejor se presenta el siguiente ejemplo:

```
function printId(id: number | string) {
 console.log(id.toUpperCase());
 //Property 'toUpperCase' does not exist on type 'string | number'.
  //Property 'toUpperCase' does not exist on type 'number'.
}
```

En el caso que todas elementos de una unión compartan cosas en común es posible realizar o utilizar métodos a dichos elementos sin problema, un ejemplo sencillo es que los arreglos o strings pueden utilizar el método .slice como se presenta a continuación:

```
// Return type is inferred as number[] | string
function getFirstThree(x: number[] | string) {
 return x.slice(0, 3);
}
```

3.6. Clases e interfaces

3.6.1 Clases

TypeScript ofrece soporte completo para las clases introducidas en ES2015. Al igual que con otras características del lenguaje JavaScript, TypeScript agrega anotaciones de tipo y otra sintaxis para permitirle expresar relaciones entre clases y otros tipos. A continuación, un ejemplo de como se ve una clase en typescript:

```
class Point {
  x: number;
  y: number;
}

const pt = new Point();
pt.x = 0;
pt.y = 0;
```

Al igual que con otras características, la anotación de tipo es opcional, pero tomara como default any si no se especifica. Los campos también pueden tener inicializadores y se ejecutarán automáticamente cuando se instancia la clase:

```
punto de clase {
  x = 0;
  y = 0;
}

const pt = nuevo Punto();
```

```
// Imprime 0, 0
consola.log(`${pt.x}, ${pt.y}`);
```

En typescript existe una una configuración llamada strictPropertyInitialization, esta se utiliza para controlar y asegurar que los campos de una clase deben inicializarse en el constructor. Por ejemplo:

```
class BadGreeter {
  name: string;
Property 'name' has no initializer and is not definitely assigned in the constructor.
}
```

Tenga en cuenta que el campo debe inicializarse en el propio constructor. TypeScript no analiza los métodos que invoca desde el constructor para detectar inicializaciones, porque una clase derivada podría anular esos métodos y fallar al inicializar los miembros. La manera correcta de escribirla seria:

```
class GoodGreeter {
  name: string;

  constructor() {
    this.name = "hello";
  }
}
```

Si definitivamente necesitas inicializar el campo de otra manera se puede usar el operador ! como se muestra a continuación:

```
class OKGreeter {
  // Not initialized, but no error
  name!: string;
}
```

Modificador ReadOnly
Los campos pueden tener el modificador de solo lectura(readonly). Esto evita asignaciones al campo fuera del constructor, esto significa que no podrá ser modificado una vez ha sido inicializado:

```
class Point {
  readonly x: number;
  readonly y: number;

  constructor(x: number, y: number) {
    this.x = x;
    this.y = y;
  }
}


let point = new Point(1, 2);
point.x = 3; // Error: Cannot assign to 'x' because it is a read-only property
```

En este ejemplo, la clase Point tiene dos propiedades de solo lectura, x e y, ambas de tipo número. La función constructora inicializa estas propiedades con los valores pasados como argumentos. Si se intenta asignar un nuevo valor a una propiedad de solo lectura, obtendrá un error de tiempo de compilación. Esto se debe a que el modificador de solo lectura garantiza que la propiedad no se pueda modificar después de inicializarla.

El uso del readOnly puede ayudar a mejorar la confiabilidad y la capacidad de mantenimiento del código.

Constructores

Son muy similares a las funciones ya que se pueden agregar parámetros con tipos, valores por defecto y sobrecargas, a continuación, un ejemplo de un constructor en typescript:

```
class Point {
  x: number;
  y: number;

  constructor(x: number, y: number) {
    this.x = x;
    this.y = y;
  }
}
```

Esta clase Point tiene dos propiedades, x e y, ambas de tipo número. La función constructora toma dos argumentos, x e y, y los asigna a las propiedades correspondientes del objeto. Si se quiere crear un nuevo objeto Point puede usar el operador new y pasar los argumentos necesarios de la siguiente manera:

```
let point = new Point(1, 2);
```

Aunque se parecen mucho a las clases existen las siguientes diferencias:

- Los constructores no pueden tener parámetros tipados.
- Los constructores no pueden tener returns tipados: siempre se devuelve el tipo de instancia de clase.

3.6.2 Interfaces

En TypeScript, una interface es una forma de definir un contrato para un objeto o función, especificando la forma y los tipos de sus propiedades y parámetros. Una interface define un conjunto de propiedades y métodos que debe tener un objeto o función, pero no proporciona una implementación para esas propiedades y métodos. A continuación, un ejemplo de una interface:

```
interface User {
  id: number;
  name: string;
  email: string;
  isActive: boolean;
}
```

En esta interface se define un objeto User con cuatro propiedades: id (number), name (string), email (string) y isActive (boolean).

Las interfaces son una herramienta poderosa en TypeScript para definir una estructura, los tipos de objetos y funciones en el código además pueden ser utilizadas reforzar la coherencia y mejorar la capacidad de mantenimiento y la legibilidad del código.

5.10. Introducción a Node.js

Historia y alternativas.

Clase teórica en donde se habla brevemente del desarrollo de NodeJS, de las necesidades por las cuales surgen, sus ventajas y desventajas. También se explica al estudiante cuales son las alternativas de NodeJS que se encuentran actualmente y cuales son las ventajas y desventajas de estas alternativas.

Historia de NodeJS.

NodeJS nace de la mano de Ryan Dahl, un programador que para aquel entonces (2009) trabajaba en una compañía que ofrecía servicio en la nube, Joynet. Node JS surge de la necesidad de poder ejecutar aplicaciones en JavaScript tanto de lado del cliente como del lado del servidor.

Entre los años 2009 y 2014 la compañía en la que trabajaba Dahl apadrinó el proyecto, sin embargo el desarrollo de este no era el esperado por la comunidad, ya que cada versión se demoraba mucho en ser liberada, además de que en ese tiempo nunca se logró liberar una versión estable de Node. Debido al lento avance en NodeJS, y dado que Joynet era muy celoso con quienes podían intervenir en su desarrollo, la comunidad de desarrolladores decide sacar una rama de Node, que fuera compatible con el sistema de paquete NPM y que fuese de código abierto, bajo el nombre de io.js.

Con la existencia de io.js, Joynet decide lanzar su primera versión de Node la 0.12.1, la cual llega con varios errores que luego son corregidos en la versión 0.12.2 lanzada a los pocos días de la primera. Poco tiempo después io.js lanza su primera versión, lo que causa conflicto entre la empresa y la comunidad. Así en mayo de 2015 y con la intervención de la fundación LINUX las dos variantes se unen en lo que se conoce como la NodeJS Foundation, encargada de la reunificación de las dos tecnologías.
La fundación tenía como objetivo unificar las tecnologías de Node y io.js en una sola, donde Joynet se encargaría de gestionar el apartado legal y de mercado de la nueva marca, y la comunidad de estar al pendiente del desarrollo de la tecnología. Dando así

surgimiento a la primera versión estable de NodeJS 4.0.0.
Actualmente NodeJS va en la versión 19.3.0.

¿Por qué surge NodeJS?

En un inicio JavaScript surge como un lenguaje de programación que pudiese ser ejecutado desde el navegador, es decir desde el lado del cliente, por lo tanto cada navegador debía desarrollar su propio motor para JavaScript. En 2009 el gigante tecnológico Google, lanza su propio navegador Google Chrome, el cual integraba el motor JavaScript V8, este es un motor que integraba el intérprete y el compilador de JavaScript en el mismo Software, lo que permitia que el código se ejecutará más rápido en comparación a otros navegadores del momento.

La relación entre el motor V8 de Chrome y NodeJS, nace cuando Dahl buscando una manera en que los servidores web permiten un número enorme de usuarios en simultáneo, encuentra el motor V8, el cual logra después montar en el sistema operativo, que era precisamente lo que Dahl estaba buscando para NodeJS.

Entonces Node JS es un entorno de desarrollo para JavaScript, que puede ser ejecutado desde el lado del cliente, en el navegador, pero que además puede ser ejecutado desde el lado del servidor, que además entre las ventajas que se consigue con esto no solo radica en la rapidez que integra el motor V8, sino que también permite la utilización de un solo hilo de ejecución para la aplicación con lo que se garantiza que no se presenten bloqueos en los procesos.

¿Qué es Node JS?

Node JS es un entorno de ejecución multiplataforma de código abierto que permite a los desarrolladores crear todo tipo de herramientas y aplicaciones del lado del servidor utilizando JavaScript. La ejecución en tiempo real está diseñada para usarse fuera del contexto de los navegadores web, por lo que el marco de trabajo omite las API de JavaScript específicas del navegador web y agrega soporte para las API de sistema operativo más tradicionales, incluidas las bibliotecas de sistemas de archivos y HTTP.
Node JS ofrece una serie de ventajas, entre las que se encuentran:

Un gran rendimiento pues Node JS fue diseñado para optimizar el rendimiento y la escalabilidad en aplicaciones web y es un muy buen complemento para muchos problemas comunes de desarrollo web.

El código está escrito en JavaScript, el cual es un lenguaje de programación relativamente nuevo y se beneficia de los avances en diseño de lenguajes cuando se compara con otros lenguajes de servidor web tradicionales como PHP o Python.
Cuenta con el gestor de paquetes de Node (NPM del inglés: Node Packet Manager) proporcionando acceso a cientos o miles de paquetes reutilizables.

Es portable, con versiones que funcionan en Microsoft Windows, OS X, Linux, Solaris, FreeBSD, OpenBSD, WebOS, y NonStop OS. Además, está bien soportado por muchos de los proveedores de hospedaje web, que proporcionan infraestructura específica y documentación para hospedaje de sitios Node.
Tiene un ecosistema y comunidad de desarrolladores de terceros muy activa.

Un “Hola mundo” en Node JS

En Node JS se puede crear fácilmente un servidor web básico que responda a cualquier solicitud simplemente usando el paquete Node HTTP. Esto creará un servidor y este escuchará las solicitudes en la URL http://127.0.0.1:8000/ (localhost); cuando recibe una solicitud, responde enviando una respuesta escrita: “¡Hola mundo!”.

En el siguiente código se muestra cómo se construye el “hola mundo” en Node JS

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hola Mundo');
});

server.listen(port, hostname, () => {
  console.log(`El servidor se está ejecutando en http://${hostname}:${port}/`);
});
```

¿Qué es NPM?

NPM o “Node Package Manager” es el administrador de paquetes predeterminado para el tiempo de ejecución de JavaScript Node.js. Comenzó como una forma de descargar y administrar dependencias de paquetes .js Node, pero desde entonces se ha convertido en una herramienta utilizada también en JavaScript frontend.
NPM consiste de dos partes principales: una herramienta CLI (interfaz de línea de comandos) para la publicación y descarga de paquetes, y un repositorio en línea que alberga paquetes de JavaScript.

package.json

Cada proyecto de JavaScript se puede considerar como un paquete npm con su propia información de paquete y un archivo package.json que describe el proyecto. package.json se generará cuando ejecute npm init para inicializar un proyecto de JavaScript/

Node.js con los siguientes metadatos principales proporcionados por el desarrollador:
name: el nombre de tu librería/proyecto JavaScript

version: la versión de tu proyecto. A menudo, para el desarrollo de aplicaciones, este campo es a menudo descuidado ya que no hay necesidad aparente de versionar librerías de código abierto. Pero aún así, puede ser útil como fuente de la versión del despliegue.
description: la descripción del proyecto
license: la licencia del proyecto
npm scripts.
package.json también soporta la propiedad scripts que puede definirse para ejecutar herramientas de línea de comandos que se instalan en el contexto local del proyecto.
dependencies vs devDependencies
Estos dos vienen en forma de objetos clave-valor con los nombres de las librerías npm como clave y sus versiones en formato semántico como valor. Estas dependencias se instalan mediante el comando npm install con las banderas --save y --save-dev. Están pensadas para ser usadas en entornos de producción y desarrollo/pruebas respectivamente.

package-lock.json

Este archivo describe las versiones exactas de las dependencias utilizadas en un proyecto de JavaScript npm. package-lock.json es usualmente generado por el comando npm install, y también es leído por nuestra herramienta NPM CLI para asegurar la reproducción de los entornos de construcción para el proyecto con npm ci.

5.11. Introducción a React js

Historia y alternativas.

Clase teórica en donde se habla brevemente del desarrollo de NodeJS, de las necesidades por las cuales surgen, sus ventajas y desventajas. También se explica al estudiante cuales son las alternativas de NodeJS que se encuentran actualmente y cuales son las ventajas y dees se unen en lo que se conoce como la NodeJS Foundation, encargada de la reunificación de las dos tecnologías.
La fundación tenía como objetivo unificar las tecnologías de Node y io.js en una sola, donde Joynet se encargaría de gestionar el apartado legal y de mercado de la nueva marca, y la comunidad de estar al pendiente del desarrollo de la tecnología. Dando así

React

Las bibliotecas de JavaScript son colecciones de código JavaScript escrito previamente que se pueden usar para tareas comunes de JS, lo que le permite evitar el proceso de codificación manual que requiere mucho tiempo. Si hay una función de JavaScript común y corriente que necesita codificar (y que otros desarrolladores han necesitado antes que usted para sus propios proyectos), probablemente haya una biblioteca JS para resolver este problema.

React es una biblioteca de JavaScript que se especializa en ayudar a los desarrolladores a crear interfaces de usuario o UI. En términos de sitios web y aplicaciones web, las IU son la colección de menús en pantalla, barras de búsqueda, botones y cualquier otra cosa con la que alguien interactúa para USAR un sitio web o una aplicación.

Antes de React JS, los desarrolladores estaban atascados construyendo interfaces de usuario con “vanilla JavaScript” o con predecesores de React menos centrados en la interfaz de usuario como jQuery. Eso significó tiempos de desarrollo más largos y muchas oportunidades para errores y fallas. Entonces, en 2011, el ingeniero de Facebook, Jordan Walke, creó React JS específicamente para mejorar el desarrollo de la interfaz de usuario.

Además de proporcionar código de biblioteca React reutilizable (ahorrando tiempo de desarrollo y reduciendo la posibilidad de errores de codificación), React viene con dos características clave que se suman a su atractivo para los desarrolladores de JavaScript:

- JSX
- DOM virtual

Historia

En 2011, los desarrolladores de Facebook comenzaron a enfrentar algunos problemas con el mantenimiento del código. A medida que la aplicación de anuncios de Facebook obtuvo una cantidad cada vez mayor de feature (características), el equipo necesitaba más personas para que funcionara sin problemas. Después de un tiempo, los ingenieros de Facebook no pudieron mantenerse al día con estas actualizaciones en cascada. Su código exigía una actualización urgente para volverse más eficiente. Tenían el modelo correcto, pero necesitaban hacer algo con respecto a la experiencia del usuario. Entonces, Jordan Walke construyó un prototipo que hizo que el proceso fuera más eficiente, y es aquí donde inicia React.

- **2010**
- Facebook introdujo xhp en su pila de php y lo convirtió en código abierto.
- Xhp permitía crear componentes compuestos. Introdujeron esta sintaxis más adelante en React.

- **2011**
- Jordan Walke creó FaxJS, el primer prototipo de React: envió un elemento de búsqueda en Facebook.

- **2012**
- Los anuncios de Facebook se volvieron difíciles de administrar, por lo que Facebook necesitaba encontrar una buena solución. Jordan Walke trabajó en el prototipo y creó React.
- Instagram quería adoptar la nueva tecnología de Facebook. Por esto, Facebook tenía la presión de desvincular React de Facebook y hacerlo de código abierto. La mayor parte de esto fue hecho por Pete Hunt.

- **2013**
- En la JS ConfUS, Jordan Walke presentó React. React se vuelve de código abierto. Dato curioso: el público se mostró escéptico.
- React (by Facebook) se encuentra disponible en JSFiddle
- React y JSX se encuentra disponible Ruby on Rails
- React y JSX disponibles en aplicaciones Python

- **2014**
- Comenzaron las conferencias #reactjsworldtour, para construir una comunidad.
- React Developer Tools se convierte en una extensión de Chrome Developer Tools.
- El lanzamiento de React Hot Loader. React Hot Loader es un complemento que permite que los componentes de React se recarguen en vivo sin pérdida de estado.

- **2015**
- Netflix usa React
- Airbnb usa React
- En React.js Conf 2015 Facebook lanzó la primera versión de React Native durante una charla técnica.
- Redux fue lanzado por Dan Abramov y Andrew Clark.

- **2016**
- Llega Mobx
- La introducción de React Storybook
- Se presenta el sistema de códigos de error de React.
- Introducción de Blueprint: un kit de herramientas de interfaz de usuario de React.

- **2017**
- React 16: error boundaries, portals, fragments y Fiber architecture

- **2020**
- React 17: Después de dos años y medio desde el lanzamiento release major. Esta versión se centró principalmente en simplificar la actualización de React desde versiones anteriores porque no incluía ninguna característica nueva para desarrolladores. Antes de este lanzamiento, la actualización de una versión anterior de React a una nueva requería actualizar toda la aplicación a la vez. Con React 17 vino la opción de actualizar toda su aplicación a la vez o actualizar su aplicación pieza por pieza como mejor le parezca.

- **2022**
- Nuevas características: Concurrencia, automatic batching, nueva característica Suspense, Transitions, client y server rendering APIs, nuevos hooks.

Introducción a Reactjs: Components stateless and stateful, pops, state, JSX, ciclo de vida.

- **JSX**

En el corazón de cualquier sitio web básico se encuentran los documentos HTML. Los navegadores web leen estos documentos y los muestran en su computadora, tableta o teléfono como páginas web. Durante este proceso, los navegadores crean algo llamado Document Object Model (DOM), un árbol de representación de cómo se organiza la página web. Luego, los desarrolladores pueden agregar contenido dinámico a sus proyectos modificando el DOM con lenguajes como JavaScript.

JSX (abreviatura de JavaScript eXtension) es una extensión de React que facilita a los desarrolladores modificar el DOM mediante el uso de un código simple de estilo HTML. Y, dado que la compatibilidad con el navegador React JS se extiende a todos los navegadores web modernos, JSX es compatible con cualquier plataforma de navegador con la que esté trabajando.

El uso de JSX para actualizar un DOM conduce a mejoras significativas en el rendimiento del sitio y la eficiencia del desarrollo. Se trata de la próxima característica de React, el DOM virtual.

```
const element = <h1>Hello, world!</h1>;
```

Ejemplo de elemento JSX. (Reactjs Documentation)

```
const element = (
  <div>
    <h1>Hello!</h1>
    <h2>Good to see you here.</h2>
  </div>
);
```

Ejemplo de elemento JSX con elementos hijos. (Reactjs Documentation)

Una de las ventajas de utilizar JSX es la prevención de ataques de inyección debido a que por defecto React DOM escapa cualquier valor insertado en JSX antes de ser renderizado. De esta manera es posible garantizar que nunca se podrá insertar algo que no se encuentre explícito en la aplicación. Todo es convertido a string antes de ser renderizado.

```
const title = response.potentiallyMaliciousInput;
// Esto es seguro:
const element = <h1>{title}</h1>;
```

Ejemplo de elemento JSX con contenido variable seguro. (Reactjs Documentation)

Una de las ventajas de utilizar JSX es la prevención de ataques de inyección debido a que por defecto React DOM escapa cualquier valor insertado en JSX antes de ser renderizado. De esta manera es posible garantizar que nunca se podrá insertar algo que no se encuentre explícito en la aplicación. Todo es convertido a string antes de ser renderizado.

- **Virtual DOM**

Si un desarrollador usa JSX para manipular y actualizar su DOM, React JS crea algo llamado Virtual DOM. El DOM virtual es una copia del DOM del sitio, y React JS usa esta copia para ver qué partes del DOM real deben cambiar cuando ocurre un evento (como un usuario que hace clic en un botón).

Supongamos que un usuario ingresa un comentario en un formulario de publicación de blog y presiona el botón “Comentario”. Sin usar React JS, todo el DOM tendría que

actualizarse para reflejar este cambio (utilizando el tiempo y la potencia de procesamiento necesarios para realizar esta actualización). React, por otro lado, escanea el DOM virtual para ver qué cambió después de una acción del usuario (en este caso, se agregó un comentario) y actualiza selectivamente sólo esa sección del DOM.

Este tipo de actualización selectiva requiere menos poder de cómputo y menos tiempo de carga, lo que puede no parecer mucho cuando habla de un solo comentario de blog, pero cuando se empieza a pensar en todas las dinámicas y actualizaciones asociadas la suma de esas actualizaciones es mucho.

- **Introducción**

El primer componente en el siguiente ejemplo es App. Este componente es el propietario del Encabezado y el Contenido. Estamos creando Encabezado y Contenido por separado y simplemente agregándole dentro del árbol JSX en nuestro componente de aplicación. Solo es necesario exportar el componente de la aplicación. Se dice que es un componente stateless debido a que simplemente se está visualizando contenido estático y no se está manejando alguna variable dentro del componente.

Cabe resaltar que el siguiente ejemplo ilustra la creación de componentes basados en clases, un enfoque deprecado. Actualmente los componentes están basados en funciones javascript.

```
import React from 'react';
class App extends React.Component {
  render() {
     return (
        <div>
           <Header/>
           <Content/>
        </div>
     );
  }
}
class Header extends React.Component {
  render() {
     return (
        <div>
           <h1>Header</h1>
        </div>
     );
  }
```

```
}
class Content extends React.Component {
  render() {
     return (
        <div>
           <h2>Content</h2>
           <p>The content text!!!</p>
        </div>
     );
  }
}
export default App;
```

Componente basado en clases.

Conceptualmente, los componentes son como funciones de JavaScript. Aceptan entradas arbitrarias (llamadas “props”) y devuelven elementos React que describen lo que debería aparecer en la pantalla.

- **Props en un componente React**

React nos permite pasar información a los componentes usando cosas llamadas props (abreviatura de propiedades). Debido a que React consta de varios componentes, las props permiten compartir los mismos datos entre los componentes que los necesitan. Hace uso del flujo de datos unidireccional (flujo de información desde el componente padre al componente hijo). Sin embargo, con una función callback, es posible devolver props de un componente hijo a uno padre.

Estos datos pueden venir en diferentes formas: números, cadenas, matrices, funciones, objetos, etc. Podemos pasar props a cualquier componente, al igual que podemos declarar atributos en cualquier etiqueta HTML.

Algo para tener en cuenta, las props solo pueden ser leídas más no modificadas.

```
<PostList posts={postsList} />
```

Ejemplo componente que recibe props.

```jsx
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

const root = ReactDOM.createRoot(document.getElementById('root'));
const element = <Welcome name="Sara" />;
root.render(element);
```

Componente Welcome que recibe prop name. (Reactjs Documentation)

Los componentes pueden hacer referencia a otros componentes. Esto permite usar la misma abstracción de componentes para cualquier nivel de detalle. Un botón, un formulario, un cuadro de diálogo, una pantalla: en las aplicaciones React, todo eso se expresa comúnmente como componentes.

```jsx
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

function App() {
  return (
    <div>
      <Welcome name="Sara" />
      <Welcome name="Cahal" />
      <Welcome name="Edite" />
    </div>
  );
}
```

Ejemplo de reutilización de componentes. (Reactjs Documentation)

- **Componentes puros**

Basado en el concepto de pureza en los paradigmas de programación funcional, se dice que una función es pura si cumple las siguientes dos condiciones:

- Su valor de retorno sólo está determinado por sus valores de entrada
- Su valor de retorno es siempre el mismo para los mismos valores de entrada.

Un componente de React se considera puro si genera el mismo resultado para el mismo estado y propiedades.

Es posible establecer un ejemplo de funciones puras con fórmulas matemáticas así: Consideremos esta fórmula matemática: y = 2x.

- Si x = 2 entonces y = 4. Siempre.
- Si x = 3 entonces y = 6. Siempre.
- Si x = 3, a veces y no será 9 o -1 o 2.5 dependiendo de la hora del día o el estado del mercado de valores.
- Si y = 2x y x = 3, y siempre será 6.

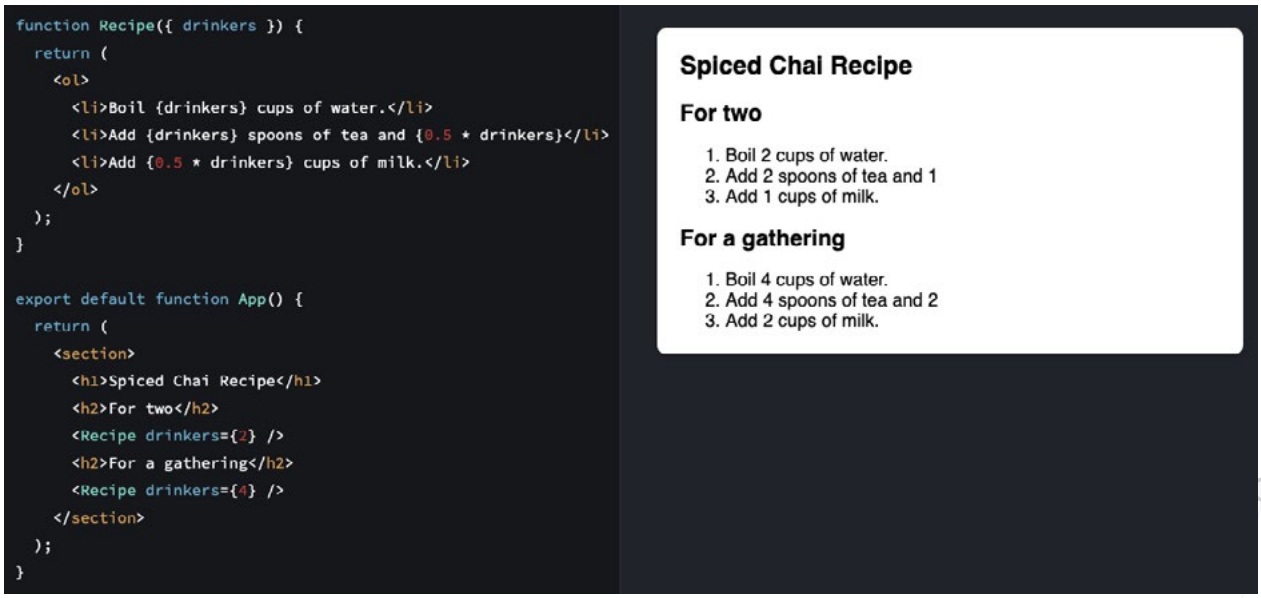
Si esto se convierte en una función de JavaScript, obtenemos lo siguiente:

```
function double(number) {
 return 2 * number;
}
```

Ejemplo función pura

En el ejemplo anterior, doble es una función pura. Si se le pasa 3 como argumento, siempre retornará 6.

React está diseñado en torno a este concepto. React asume que cada componente que escribes es una función pura. Esto significa que los componentes de React que escriba siempre deben devolver el mismo JSX con las mismas entradas:



Ejemplo componente puro. (Reactjs Documentation)

- Estado

Hasta el momento se ha hablado de componentes que reciben props y deben renderizar algún contenido relacionado a estas. Este tipo de componentes son llamados stateless o presentacionales. Ahora, generalmente es necesario crear un componente statefull, es decir con un estado que va a cambiar con el tiempo.

```
import React, {useState} from 'react';

const student = (props) => {
    const [personalState,setPersonalState] = useState(
      {
        name: "Bob",
        age: 18,
        contactNo: "1234567890",
      }
    );
    const [educationState, setEducationState] = useState(
      {
        class: "A1",
        subjects: ["Physics","Chemistry","Biology"],
      }
    );
```

```
    const changePersonalStateHandler = ()=>{
        setPersonalState(
          {
            name: "Alice",
            age: 20,
            contactNo: "956743218",
          }
        );
    }
    const changeEducationStateHandler = ()=>{
        setEducationState({
            class: "A2",
            subjects: ["Physics","Information Technology","Numerical
Mathematics"],
        });
    }

    return (
      <div>
        <h1>Example - UseState hook</h1>
          <button onClick={changePersonalStateHandler}>Change Personal
State</button>
          <button onClick={changeEducationStateHandler}>Change Education
Values</button>
            .............
      </div>
    );
}
export default student;
```

Ejemplo componente con estado. (Reactjs Documentation)

- **Manejo de eventos**

El manejo de eventos con elementos React es muy similar al manejo de eventos en elementos DOM, pero hay algunas diferencias de sintaxis:

- Los eventos en React se nombran usando camelCase, en lugar de minúsculas.
- Con JSX, pasa una función como controlador de eventos, en lugar de una cadena.

```
<form onsubmit="console.log('You clicked submit.'); return false">
  <button type="submit">Submit</button>
</form>
```

Ejemplo manejo de eventos con HTML (Reactjs Documentation)

```
function Form() {
  function handleSubmit(e) {
    e.preventDefault();
    console.log('You clicked submit.');
  }

  return (
    <form onSubmit={handleSubmit}>
      <button type="submit">Submit</button>
    </form>
  );
}
```

Ejemplo manejo de eventos con React (Reactjs Documentation)

- **Renderizado condicional**

En React, es posible crear distintos componentes que encapsulan el comportamiento necesario. Luego, puede representar solo algunos de ellos, según el estado de la aplicación. La representación condicional en React funciona de la misma manera que las condiciones en JavaScript.

Considerando los siguientes componentes:

```
function UserGreeting(props) {
  return <h1>Welcome back!</h1>;
}

function GuestGreeting(props) {
  return <h1>Please sign up.</h1>;
}
```

Componentes stateless (Reactjs Documentation)

Es posible establecer el renderizado condicional de la siguiente manera:

```
function Greeting(props) {
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn) {
    return <UserGreeting />;
  }
  return <GuestGreeting />;
}

const root = ReactDOM.createRoot(document.getElementById('root'));
// Try changing to isLoggedIn={true}:
root.render(<Greeting isLoggedIn={false} />);
```

Renderizado condicional. (Reactjs Documentation)

- Renderizado de listas de elementos

Para renderizar elementos según una lista de componentes se hace uso de la función map(). Aquí se podrá iterar y definir la presentación de cada elemento de la lista.

```
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    <li key={number.toString()}>
      {number}
    </li>
  );
  return (
    <ul>{listItems}</ul>
  );
}
```

Ejemplo de renderización de elementos según una lista. (Reactjs Documentation)

5.12. Introducción a Angular js

¿Qué es Angular JS?

¿Qué es Angular JS?

AngularJS es un framework MVC de JavaScript para el Desarrollo Web Front End que permite crear aplicaciones SPA Single-Page Applications. Entra dentro de la familia de frameworks como BackboneJS o EmberJS.

Con tanta oferta de frameworks se hace difícil elegir cuál usar en nuestras aplicaciones, qué ventajas tienen unos frente a otros, etc.

¿Por qué usar Angular JS?

HTML es excelente para declarar documentos estáticos, pero falla cuando intentamos usarlo para declarar vistas dinámicas en aplicaciones web. AngularJS le permite ampliar el vocabulario HTML para su aplicación. El entorno resultante es extraordinariamente expresivo, legible y rápido de desarrollar.

index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
  </head>
  <body>
    <div>
      <label>Name:</label>
      <input type="text" ng-model="yourName" placeholder="Enter a name here">
      <hr>
      <h1>Hello {{yourName}}!</h1>
    </div>
  </body>
</html>
```

Ventajas de AngularJS

AngularJS es popular entre los desarrolladores web por un par de razones:

- Enlace bidireccional de datos: dado que la arquitectura de AngularJS enlaza JavaScript y HTML, el código de ambos ya está sincronizado. Por lo tanto, el framework ahorra mucho tiempo a los desarrolladores.
- Directivas: el marco amplía la funcionalidad de los archivos HTML con directivas. Para habilitar las directivas, los desarrolladores agregan el prefijo ng- a los atributos HTML.
- Estructura de código: AngularJS brinda plantillas; lo que te permite producir aplicaciones con código limpio. No solo ahorra tiempo, sino que también facilita la modificación o reparación de las aplicaciones.

- Pruebas: el marco admite pruebas unitarias y de integración.
- Futuro brillante: el futuro de Angular es brillante debido a su funcionalidad y popularidad. Su base de usuarios sigue creciendo y tiene una gran cantidad de documentación en profundidad que se actualiza constantemente.
- Compatibilidad móvil y de escritorio: AngularJS puede ejecutarse en la mayoría de los navegadores web. No solo en computadoras de escritorio, sino también en dispositivos móviles.

Otros materiales para profundizar

Recursos de video

CodelyTV - Redescubre la programación (Director). (2021, junio 22). SÓLO 3 LÍNEAS: CSS Grid responsive sin media queries. https://www.youtube.com/watch?v=El0OJ6h_2ZI

codigofacilito (Director). (2015). 1.- Curso AngularJS - Introducción. https://www.youtube.com/watch?v=U4ejmeEals0

Fazt (Director). (2018, julio 22). Nodejs Curso Desde Cero, para principiantes. https://www.youtube.com/watch?v=BhvLIzVL8_o

Fazt (Director). (2019, noviembre 29). Typescript | ¿Qué es Typescript y porqué aprenderlo? https://www.youtube.com/watch?v=wXBl5tZm-OQ

freeCodeCamp Español (Director). (2022, marzo 15). Aprende React Desde Cero—Curso de React Con Proyectos. https://www.youtube.com/watch?v=6Jfk8ic3KVk

GioCode (Director). (2017, marzo 5). ¿Que son los preprocesadores? | Desarrollo web. https://www.youtube.com/watch?v=P7i3-hbNVVg

Referencias bibliográficas de la unidad

Arsaute, A., Zorzán, F. A., Daniele, M., Gonzaléz, A., & Frutos, M. (2018). Generación automática de API REST a partir de API Java, basada en transformación de Modelos (MDD). XX Workshop de Investigadores en Ciencias de la Computación (WICC 2018, Universidad Nacional del Nordeste). http://sedici.unlp.edu.ar/handle/10915/67777

Bryant, J., & Jones, M. (2012). Responsive Web Design. En J. Bryant & M. Jones (Eds.), Pro HTML5 Performance (pp. 37-49). Apress. https://doi.org/10.1007/978-1-4302-4525-4_4

Richards, G., Nardelli, F. Z., & Vitek, J. (2015). Concrete Types for TypeScript. En J. T. Boyland (Ed.), 29th European Conference on Object-Oriented Programming (ECOOP 2015) (Vol. 37, pp. 76-100). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. https://doi.org/10.4230/LIPIcs.ECOOP.2015.76

Tilkov, S., & Vinoski, S. (2010). Node.js: Using JavaScript to Build High-Performance Network Programs. IEEE Internet Computing, 14(6), 80-83. https://doi.org/10.1109/MIC.2010.145

Pérez Ibarra, S. G., Quispe, J. R., Mullicundo, F. F., & Lamas, D. A. (2021). Herramientas y tecnologías para el desarrollo web desde el FrontEnd al BackEnd. XXIII Workshop de Investigadores en Ciencias de la Computación (WICC 2021, Chilecito, La Rioja). http://sedici.unlp.edu.ar/handle/10915/120476

ALCALDÍA MAYOR
DE BOGOTÁ D.C.
SECRETARÍA DE EDUCACIÓN

ATENEA
AGENCIA DISTRITAL PARA LA EDUCACIÓN
SUPERIOR LA CIENCIA Y LA TECNOLOGÍA

BOGOTÁ

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS
Acreditación Institucional de Alta Calidad