**PRIMER PARCIAL APPS NATIVAS**

***Apps Nativas:***son aplicaciones desarrolladas específicamente para una plataforma móvil, como iOS o Android, utilizando los lenguajes de programación y herramientas nativas de esas plataformas.

***Frontend Frameworks:***son conjuntos de librerías que facilitan el desarrollo de interfaces web dinámicas y robustas, utilizando tecnologías como HTML, CSS y JavaScript. Aunque las tecnologías web evolucionan, deben mantener la compatibilidad con sitios antiguos (legacy), lo que impide cambios radicales. Elegir un framework implica trabajar dentro de un sistema que ofrece herramientas específicas, añadiendo complejidad y abstracción al proceso de desarrollo.

Vainilla vs Frameworks → vainilla no requiere compilación y puedo cargar los archivos que yo quiera, sin restricciones, mientras que el Framework tiene una filosofía, hay que seguir reglas. Frameworks está más asociado a SPA y vainilla a MPA.

Algunos de los Frameworks de frontend más populares son:

* [React](https://www.notion.so/React-bb23f49f81e74a228c124b5503194ef8?pvs=21)
* [Angular](https://www.notion.so/Angular-78da57600dff4313ac59611afffab6c6?pvs=21)
* [Vue](https://www.notion.so/Vue-c7c9df5639b64188ba1ff9e2b919e752?pvs=21)
* [Svelte](https://www.notion.so/Svelte-2fae1bf216bf4e24b5063eaa9ac73c09?pvs=21)

***SPA vs. MPA (single page application vs. multiple page application)***

La gran mayoría de los sitios web son MPA “Multiple page applications”. Esto significa que cada página a la cual entramos en un sitio web es una página 100% nueva para nuestro navegador.

Últimamente los requerimientos de los sitios web están aumentando y cada vez más queremos que nuestros sitios web sean aplicaciones weby quebrinden una experiencia similar a aplicaciones de escritorio o móviles, los [Frontend Frameworks](https://gonzalo-bechara.notion.site/Frontend-Frameworks-e724463242a249dc83e17caa57ad72a1?pvs=24) modernos de javascript nos permiten programar de esta manera.

|  | **MPA** | **SPA** |
| --- | --- | --- |
| **Velocidad de carga inicial** | Rápida; cada página se carga de forma independiente | No tan rápida; el contenido puede tardar más en aparecer inicialmente |
| **Velocidad de carga de páginas** | No tan rápida; cada navegación implica cargar toda la página | Casi-instantánea; el contenido se carga de manera dinámica sin recargar |
| **Interactividad** | Peor; más limitada la interacción entre paginas | Mejor; permite una experiencia de usuario más fluida y dinámica |
| **Posicionamiento SEO** | Mejor; cada página tiene su propia URL | Peor |
| **Animaciones** | No se puede entre páginas | Permitido entre páginas |
| **Frameworks** | Laravel, Express, NextJs, Wordpress, Vainilla, etc. | [React](https://gonzalo-bechara.notion.site/React-bb23f49f81e74a228c124b5503194ef8?pvs=24) [Angular](https://gonzalo-bechara.notion.site/Angular-78da57600dff4313ac59611afffab6c6?pvs=24) [Vue](https://gonzalo-bechara.notion.site/Vue-c7c9df5639b64188ba1ff9e2b919e752?pvs=24), etc. |
| **Ejemplos** | [Campus virtual](https://campusvirtual.austral.edu.ar/), [mercadolibre](http://mercadolibre.com.ar/), [rosario3](http://rosario3.com/), [argentina.gob.ar](http://argentina.gob.ar/) | [Instagram](http://instagram.com/), [X](http://x.com/), [YouTub](http://youtube.com/) |

Mientras que las *SPA* son mejores para aplicaciones interactivas y dinámicas donde la experiencia del usuario es una prioridad, pero pueden requerir más esfuerzo para SEO, las *MPA* son más adecuadas para sitios con contenido diverso y estático, donde el SEO y la indexación son importantes.

SEO: optimización para motores de búsqueda, destinado a mejorar la visibilidad y el posicionamiento de un sitio web en los resultados orgánicos de los motores de búsqueda, como Google

En las MPA cada página puede tener su propio estado, y no se mantiene el estado entre las diferentes páginas a menos que se implemente almacenamiento local o cookies.

**LocalStorage/Almacenamiento local:** permite almacenar datos en el navegador del usuario de forma persistente. A diferencia de las cookies, la información almacenada en LocalStorage no tiene una fecha de expiración, lo que significa que se mantiene guardada incluso después de que el usuario cierre el navegador o apague el dispositivo, hasta que se elimine explícitamente. Se accede por método get (localstorage.get(una etiqueta)) y se escribe por método set (localstorage.set(etiqueta valor)).

**Áreas principales del desarrollo web**

Llamamos desarrollo web al proceso de crear páginas web o aplicaciones web, tanto en la parte que se muestra al cliente/usuario del producto como a lo que pasa por detrás y que permite que todo funcione. Nos encontramos con dos áreas principales para el desarrollo web:

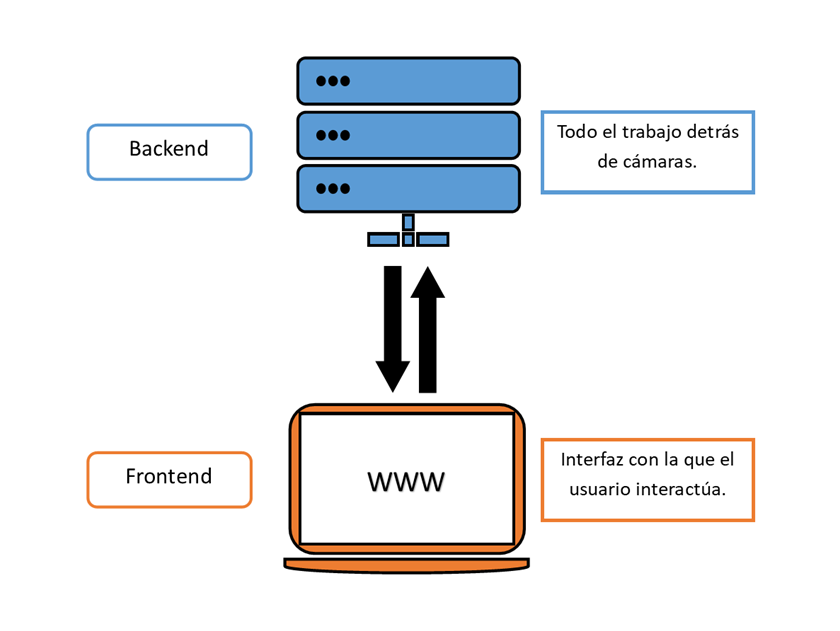
*Frontend*:

El frontend es el “lado del frente” del desarrollo web y se enfoca en el diseño y la implementación de lo que el usuario ve e interactúa en una página web. Esto incluye la creación de elementos web, colores, fuentes, posiciones, y otros estilos, así como la programación de lo que cada elemento hace, como a dónde llevan los botones o qué funcionalidades existen.

Los lenguajes más básicos del frontend son:

* Html
* Css
* JavaScript

*Backend:*

El backend es el “lado de atrás” de una aplicación web y se encarga de la lógica y el procesamiento de datos que no son visibles para el usuario, pero que son esenciales para el funcionamiento de la aplicación. Esto incluye la conexión con bases de datos, autenticación de usuarios, y gestión de transacciones.

A diferencia del frontend, el backend puede desarrollarse en distintos lenguajes de programación como Java, .NET (C#), Python, y JavaScript (Node.js). La elección del lenguaje depende de factores como la velocidad de ejecución, la cantidad de profesionales disponibles, y la experiencia previa con una tecnología.

Ambas áreas del desarrollo web pueden integrarse en una sola aplicación o desarrollarse como aplicaciones separadas. El frontend depende del backend para funciones como la persistencia de datos o la autenticación de usuarios, mientras que el backend, sin un frontend, sería un servicio sin interfaz visual.

*Sin un backend:* Un front no tendría persistencia de datos, no podría tener funcionalidades de cuentas de usuario, de comercio electrónico, no podría tener un algoritmo privado para generar algún tipo de dato, tendría todo su procesamiento generado en tiempo real.

*Sin un frontend:* Un back sería solo un servicio (aplicación que corre “de fondo” sin interfaz visual). En principio no tendría sentido de ser.

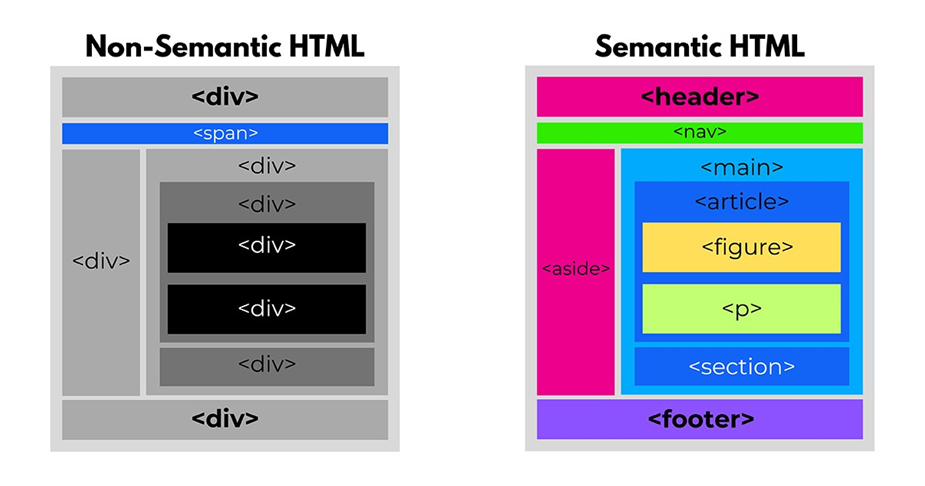
**Métodos:**

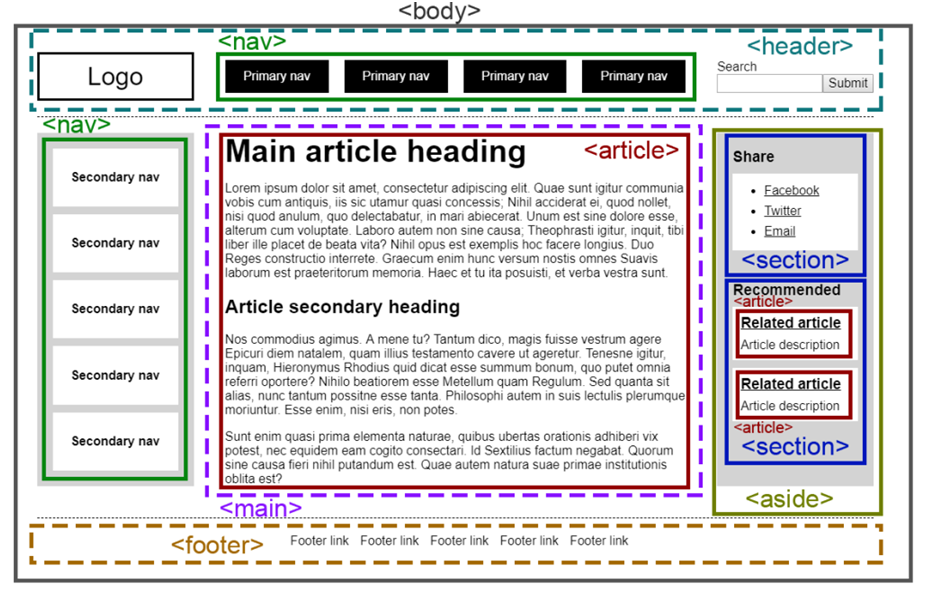
* **Get:** obtener información.
* **Post:** agregar un elemento nuevo. “Se usa cuando ninguno de los otros se puede usar”.
* **Put:** reemplaza un elemento por completo.
* **Patch:** reemplaza parcialmente un elemento.
* **Delete**: borrar un elemento de la base de datos.
* **Fetch**: no traba el sistema. Función asíncrona (tiene que esperar la respuesta del usuario). Hace la solicitud al backend, devuelve una respuesta, después de recibir la respuesta, con esa misma voy a transformarla en res.json. El método fetch se utiliza para hacer solicitudes HTTP desde el frontend a un servidor. Por ejemplo, para obtener o enviar datos. En el then se ejecuta todo lo que va después de la respuesta

El programador puede elegir cualquier método y hacer una función de otro. Es solo una guía.

**HTML**

Define los elementos que tiene una página web. HTML es un lenguaje de marcado basado en etiquetas que describe la estructura de las páginas. Todo el contenido de una página se define en HTML. No se recomienda escribir código CSS o JS dentro del archivo HTML (aunque es posible).

****

****

*Etiquetas Comunes:*

Para crear una se deben utilizar los símbolos “<” y “>”. La gran mayoría de las etiquetas requieren de dos partes, su “apertura” y su “cierre” que se distinguen por el uso de una barra para la parte de cierre. Estas etiquetas permiten poner contenido entre sus dos partes.

También hay etiquetas que tienen una sola parte como , ya que una imagen no puede contener nada adentro que no sea ella misma. Las etiquetas a la vez pueden tener luego del nombre de la etiqueta un atributo con un valor. Esto genera una relación padre-hijo entre las etiquetas.

1. **!Doctype HTML:** Lo usamos en la primera línea de nuestro archivo. Lo hacemos para indicar que estamos queremos que el navegador lea la página en un modo moderno, usando el estándar HTML que está al día.
2. **Body**: Contiene todos los elementos visibles en pantalla, como textos, imágenes, etc.
3. **Head:** Parte del HTML donde se colocan metadatos, enlaces a archivos CSS, scripts, títulos, entre otros elementos que no se renderizan directamente en la pantalla. Es todo lo que no se ve de una página, o lo que no se “renderiza/pinta” como contenido de una página.
4. **Link:** Se utiliza para vincular archivos externos, como hojas de estilo CSS o fuentes.
5. **Etiquetas no semánticas** (no proporcionan información adicional sobre el contenido, pero sirven para agrupar y aplicar estilos):

* **div**: Es un contenedor genérico sin significado semántico, que por defecto se posiciona en bloque.. Se utiliza para agrupar varios elementos y darles un estilo específico.
* **span**: Es un contenedor genérico sin significado semántico, que por defecto se posiciona en línea (inline). Se utiliza para etiquetar fragmentos pequeños de contenido dentro de un texto o párrafo, a los que se les puede asignar una clase o un ID para aplicar estilos.

1. **Etiquetas semánticas** (son elementos HTML que tienen un significado específico y ayudan a describir el contenido de una página web de manera clara y comprensible, tanto para los usuarios como para los motores de búsqueda.):
2. **Header**: Representa la parte superior de la página, donde generalmente se encuentra el logo y el menú de navegación.
3. **Nav**: Se utiliza para definir el área de navegación principal de la página.
4. **Main**: Contiene el contenido principal de la página, excluyendo las secciones de encabezado y pie de página. Dentro de esta sección se pueden incluir (estas etiquetas de contenido):

* **H1 - H6**: Etiquetas que definen títulos de diferentes niveles, donde H1 es el más importante.
* **p**: Representa un párrafo de texto. Sirve para estructurar bloques de contenido legible.
* **a (anchors / anclas):** Crea un enlace o hipervínculo hacia otras páginas o secciones.
* El atributo href especifica la URL a la que redirige, y
* target puede usarse para abrir el enlace en una nueva pestaña (\_blank).
* **Button**: Genera un botón interactivo.
* **Img**: Inserta una imagen en la página. Requiere el atributo src para indicar la ruta de la imagen.

1. **Section:** Define una sección del contenido, normalmente agrupa temas relacionados.
2. **Aside:** Contiene contenido relacionado de manera secundaria, como barras laterales o widgets.
3. **Footer**: Representa el pie de página, ubicado en la parte inferior de la página, donde suelen aparecer información adicional o enlaces.

*IDs y Clase:*

Son *parámetros que podemos agregarle a las etiquetas HTML* para poder identificar nuestros elementos. Los identificamos para luego seleccionarlos en CSS o en JS.

| ID: *Es un identificador que no se repite dentro de nuestra página web.* A diferencia de class, cada valor de id sólo puede ser usado una vez en un documento HTML. Se utiliza para referenciar un elemento específico, tanto en CSS como en JavaScript. | Class: *Es un identificador que se puede repetir.* Esto permite aplicar estilos CSS o asociar el elemento con scripts de JavaScript. Un mismo nombre de clase puede ser utilizado en varios elementos, lo que facilita la aplicación de estilos de forma consistente |
| --- | --- |

Vinculación de archivo CSS al HTML

Para aplicar estilos a un documento HTML desde un archivo CSS externo, se utiliza la etiqueta dentro del documento HTML. El atributo href indica la ubicación del archivo CSS, mientras que el atributo rel especifica que el archivo es de tipo "stylesheet" (hoja de estilos).

*Inline vs Bloque:*

* Que un elemento sea inline significa que ocupa el espacio que ocupa su contenido. Su etiqueta genérica es el **DIV**
* Que un elemento sea block significa que un elemento siempre ocupa el ancho total de la pantalla, y que después de este elemento hay un salto de línea. Su etiqueta genérica es el **SPAN.**

**CSS**

Especifica cómo se ven los elementos HTML. Este lenguaje de estilos permite ajustar el diseño visual de una página, como colores, tamaños, y posiciones de los elementos.

Los selectores CSS son la manera de seleccionar elementos de nuestro DOM. Dentro de los selectores aplicaremos las propiedades CSS

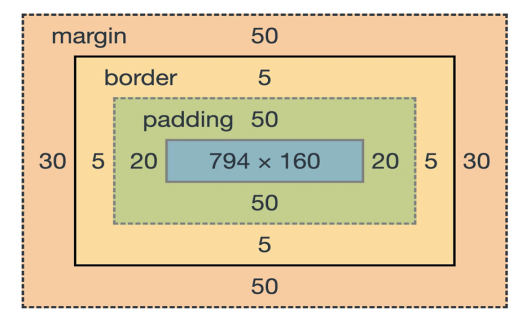
*Selectores CSS:*

* ***Selector de etiqueta*** *(ejemplo: `<h1></h1>`): Se utiliza para seleccionar todos los elementos HTML de un tipo específico.*
* *Ejemplo: `tag {}`*
* ***Selector de clase*** *(ejemplo: `class=" "`): Aplica estilos a todos los elementos que tienen una clase específica. Para usarlo, se coloca un punto (`.`) seguido del nombre de la clase.*
* *Ejemplo: `.class {}`*
* ***Selector de ID*** *(ejemplo: `id=" "`): Selecciona un único elemento con un ID específico. Para usarlo, se coloca un numeral o almohadilla (`#`) seguido del nombre del ID.*
* *Ejemplo: `#id {}`*
* ***Pseudo-selector de estado*** *(`:hover`): Selecciona un elemento cuando está en un estado específico, como cuando el usuario pasa el ratón por encima.*
* *Ejemplo: `tag:hover {}`*
* ***Selector universal:*** *Selecciona todos los elementos de la página. Este selector se representa con un asterisco (`\*`).*
* *Ejemplo: `\* {}`*

*Propiedades comunes:*

Son los aspectos que podemos modificar de un elemento, incluye ítems como tamaño, posicionamiento, visibilidad, bordes, etc. Algunas propiedades dignas de mencionar son:

* **font-family**: Cambia la fuente del texto.
* **font-size**: Ajusta el tamaño del texto.
* **color**: Cambia el color del texto.
* **margin**: Establece el espacio entre un elemento y los elementos cercanos.
* **padding:** Añade espacio entre el contenido de un elemento y su borde.
* **display**: Elegimos si un elemento es un bloque o de línea (o inline-block). También permite ocultar un elemento con display: none
* **border**: “color | estilo | tamaño”, estiliza los bordes de un objeto.
* **border-radius:** Cambia el redondeo de los bordes de un objeto.
* **color**: Establece el color del texto.
* **background-color**: Define el color de fondo de un elemento.
* **font-weight**: Establece el grosor de la fuente (normal, bold, etc.). Cambia el peso del texto
* **text-align**: Alinea el texto dentro de un contenedor (izquierda, derecha, centrado, justificado). Cambiamos el centrado de un texto respecto de su tamaño.
* **line-height**: Define la altura de línea entre el texto, afectando el espaciado vertical.
* **width**: Especifica el ancho de un elemento.
* **height**: Especifica la altura de un elemento.
* **position**: Define cómo se posiciona un elemento en relación con su contenedor (static, relative, absolute, fixed, sticky).
* **top, right, bottom, left**: Especifican la posición de un elemento cuando se usa position diferente a static.
* **flex**: Proporciona capacidades de diseño flexible para los elementos hijos de un contenedor flexible.
* **grid**: Define un diseño de cuadrícula en un contenedor, permitiendo organizar los elementos en filas y columnas.
* **cursor**: Cambia el icono del cursor al pasar sobre un elemento (por ejemplo, pointer, crosshair).
* **text-decoration**: Controla la decoración del texto (subrayado, tachado, etc.).

Modelo de la caja:

Es la manera que tenemos de visualizar el tamaño de un objeto.

El tamaño de un objeto lo componen 4 propiedades:

* El tamaño del contenido (azul).
* El espacio entre el contenido y el borde (padding: verde).
* El grosor del borde (border: amarillo).
* El espacio entre el borde y los elementos circundantes (margen: marrón).

Diseño Responsive Básico con *Media Queries*

El diseño responsive permite que una página web se vea bien en diferentes dispositivos y tamaños de pantalla (teléfonos móviles, tabletas, computadoras). Una de las herramientas clave para lograr esto en CSS son las media queries, que permiten aplicar estilos CSS dependiendo de las características del dispositivo o del tamaño de la pantalla.

*Media Queries*

Las Media queries le permiten aplicar estilos CSS según el tipo general de un dispositivo (como impresión o pantalla) u otras características como la resolución de la pantalla o el ancho del viewport del navegador.Esto ayuda a ajustar el diseño para que se vea bien en pantallas pequeñas (como móviles) y grandes (como desktops).

Las media queries se utilizan para lo siguiente:

* Para aplicar estilos condicionalmente utilizando las reglas de arroba CSS @media e @import.
* Para segmentar medios específicos para <style>, <link>, <source> y otros HTML con el atributo media=.
* Para probar y monitorear los estados de los medios usando los métodos Window.matchMedia() y EventTarget.addEventListener().

¿A qué elementos afecta un background-color? El atributo background-color en CSS aplica un color de fondo al área de contenido de un elemento, incluyendo el padding, pero no afecta los márgenes. Se usa en elementos de bloque (div, section, header), en línea (a, span), tablas (table), y se puede combinar con imágenes de fondo o pseudo-clases (hover). El color no se hereda por defecto y se extiende solo al área dentro de los bordes del elemento

*Librerías CSS:* son colecciones predefinidas de código CSS que facilitan el desarrollo de páginas web, proporcionando estilos, componentes y utilidades listas para usar. Estas librerías permiten a los desarrolladores ahorrar tiempo al no tener que escribir CSS desde cero, ya que incluyen estilos comunes como botones, menús, diseños de grid, formularios, tipografía, entre otros.

* ***Bootstrap:*** Es una de las librerías CSS más populares, desarrollada inicialmente por Twitter. Proporciona un sistema de diseño responsivo y un conjunto de componentes listos para usar, como botones, formularios, tablas, modales, y más.
* ***Tailwind CSS:*** es una librería de utilidad-first que permite a los desarrolladores construir diseños personalizados rápidamente. A diferencia de otras librerías que proporcionan componentes predefinidos, Tailwind ofrece clases de utilidad que permiten un mayor control sobre el diseño y la apariencia.

**SCSS/SASS**

Es una *extensión de CSS* que agrega características avanzadas, como variables, anidación, mixins, herencia, entre otras, para hacer que el desarrollo de hojas de estilo sea más modular, reutilizable y eficiente. Su propósito es mejorar la funcionalidad de CSS permitiendo escribir código más eficiente y mantenible, pero sigue siendo compilado en CSS regular, que es el lenguaje que los navegadores comprenden.

1. **Variables:** permite definir variables para almacenar valores que se pueden reutilizar a lo largo de la hoja de estilo, como colores, tamaños o fuentes, lo que facilita la modificación de esos valores en un solo lugar. Se declaran con el símbolo $ y el nombre de la variable, luego se llaman utilizando la misma sintaxis.
2. **Mixins:** son fragmentos de código reutilizables que pueden incluir estilos complejos. También permiten recibir parámetros para generar estilos dinámicos en función de la entrada. Son equivalentes a las funciones de los lenguajes de programación. Se declaran con la keyword @mixin y se ejecutan con la keyword @include.

**JavaScript**

Define la interacción del usuario con el sitio web, interpreta (en tiempo de ejecución) los navegadores web. Es un lenguaje de programación que permite crear funcionalidades dinámicas en una página.

Dentro de sus características podemos destacar:

* Es un *lenguaje no tipado*, por lo cual las variables tienen tipos que pueden cambiar.
* La capacidad de un IDE para proveer autocompletado es limitado o a veces erróneo, al no poder inferir el tipo de datos de una variable. Esto lleva a que no podamos encontrar errores automáticamente en el desarrollo y tengamos que experimentar los bugs en runtime.
* Es un lenguaje que debe mantener *retrocompatibilidad siempre*, para evitar “romper” la web antigua. Por eso muchas veces hay varias maneras de hacer una misma cosa, la manera “vieja” mantenida para aumentar compatibilidad con sitios legacy y las maneras “actuales”.

*Variables (let,const):*

En JavaScript, las variables se utilizan para almacenar datos que pueden ser reutilizados y manipulados a lo largo de un programa. Existen diferentes formas de declarar variables, siendo las más comunes let y const.

* **let:** Declara variables que pueden ser modificadas. Su alcance es limitado al bloque en el que se declaran. Se utiliza cuando necesitas reasignar el valor de una variable más adelante en el código.
* **const:** Declara constantes cuyo valor no puede cambiar una vez asignado. También tiene un alcance de bloque.Es útil cuando sabes que el valor no se va a modificar.

*Funciones:* Las funciones son bloques de código reutilizables que realizan una tarea específica o calculan un valor. Pueden aceptar parámetros y devolver un resultado. Las funciones pueden ser declaradas o expresadas (asignadas a variables). Se pueden definir con la palabra clave function o como funciones flecha (arrow functions).

*Ejemplos:*

|  |  |
| --- | --- |
| //Función clásica  function sumar(num1, num2){  return num1+num2;  } | //Función de flecha anónima dentro de una variable.  //versión una sola línea con return implícito  const sumarFlecha = (num1, num2) => num1+num2; |

*Event Listeners:* en JavaScript permiten que una función se ejecute cuando ocurre un evento específico en un elemento HTML, como un clic, un movimiento del mouse, o cuando se presiona una tecla.

* *addEventListener:* Esta función se utiliza para asociar un evento a un elemento. Se debe especificar el tipo de evento y la función que se ejecutará cuando ocurra el evento.

Tipos de Datos Primitivos*:*

* **String**: Para cadenas de texto, se pueden escribir de distintas maneras.
* **Number**: Para valores numéricos, incluyendo enteros y decimales. En este lenguaje no hay diferencia de tipo entre números enteros y floating point.
* **Boolean**: Valores de **true o false**. Pueden invertirse sus valores con el uso de ! antes de un boolean.
* **Falsy, truthy**: Son valores que evalúan a true o false sin ser booleanos.
* **Null**: Representa la ausencia intencional de un valor.
* **Array**:Son de base 0 por lo cual arrayVar[0] accede al primer elemento.Se escriben entre corchetes “[ ]” y no tienen porqué compartir tipos de datos en su contenido.
* **Objects**:son el tipo de dato que nos permite guardar todo tipo de información dentro. El objeto comienza y termina con llaves “{ }” y contiene conjuntos de “key : values”, los values pueden ser cualquier tipo de datos, inclusive objetos (formando así objetos anidados).
* **Undefined**: Indica que una variable ha sido declarada pero no inicializada.
* **Symbol**: Representa un valor único e inmutable.
* **BigInt**: Para representar números enteros grandes.

*Objetos en JavaScript:* Un objeto es una colección de pares clave-valor donde las claves son cadenas (o Symbols) y los valores pueden ser cualquier tipo de dato, incluidas funciones. Los objetos permiten almacenar datos estructurados y son fundamentales en la programación orientada a objetos en JavaScript.

*Uso de librerías:* JavaScript permite agregar *librerías*, que son paquetes de funcionalidades adicionales para facilitar el desarrollo. Ejemplos:

* **Bootstrap:** Librería de UI.
* **jQuery**: Manejo del DOM.
* **Parsley**: Validación de formularios.

Selección de elementos: **document.getElement**

En JavaScript, para interactuar con los elementos de una página web (DOM, Document Object Model), puedes usar varios métodos. Entre los más comunes están los métodos **document.getElementById, document.getElementsByClassName, y document.getElementsByTagName,** los cuales permiten seleccionar elementos del DOM para manipularlos.

* **document.getElementById**: Este método selecciona un único elemento del DOM que tenga el ID especificado.
* **document.getElementsByClassName:** Este método selecciona todos los elementos que tienen la misma clase. Retorna una colección de elementos (HTMLCollection), por lo que puedes acceder a ellos por su índice.
* **document.getElementsByTagName**: Este método selecciona todos los elementos que tienen el mismo nombre de etiqueta (por ejemplo, div, p, img). También retorna una colección de elementos.

*Crear Elementos del DOM:*

JavaScript permite crear y manipular elementos del DOM (modelo de objeto de documento) dinámicamente. Esto se realiza mediante métodos que permiten crear nuevos elementos y agregarlos a la estructura de la página.

*Métodos clave:*

* **document.createElement():** Se utiliza para crear un nuevo elemento HTML.
* **element.appendChild():** Agrega un nuevo hijo a un elemento existente, lo que permite insertar elementos creados en el DOM.
* **element.insertBefore():** Inserta un nuevo elemento antes de un elemento existente.

Cambiar Estilos Gráficos Agregando y Borrando Clases: Se pueden modificar los estilos de los elementos en el DOM añadiendo o eliminando clases CSS a través de JavaScript. Esto permite cambiar la apariencia de los elementos sin necesidad de modificar el CSS directamente.

*Métodos comunes:*

* **classList.add():** Agrega una o más clases al elemento.
* **classList.remove():** Elimina una o más clases del elemento.
* **classList.toggle():** Alterna la presencia de una clase; si la clase está presente, se elimina; si no, se añade.

Evento de click y console.log()

El evento de click se activa cuando el usuario hace clic en un elemento de la página web, como un botón, una imagen o un enlace.

Este evento se puede capturar usando los event listeners como addEventListener, y puedes utilizar console.log() para imprimir información en la consola del navegador cuando ocurra el clic.

* **addEventListener para el evento de click**: El método addEventListener te permite agregar una función que se ejecutará cuando ocurra el evento que especifiques, en este caso, el evento click.

*Loops*: Los bucles (loops) en JavaScript permiten repetir un bloque de código múltiples veces. Un tipo común es:

* **for...of:** Este bucle permite iterar sobre los elementos de colecciones iterables como arreglos o cadenas. Es útil para recorrer directamente los elementos sin necesidad de acceder a los índices.
* **for**: El for clásico tiene 3 componentes:
* *forweykord:* necesaria para abrir la sentencia de condición.
* *condición:* Tiene a su vez 3 elementos interiores, la declaración y asignación de una variable, la condición que hace que el for se continúe ejecutado, y la operación que se ejecuta al terminar cada iteración del bucle.
* Código que se ejecuta en cada iteración

**Asincronía:** La **asincronía** se refiere a la capacidad de ejecutar operaciones de manera independiente y no bloqueante, lo que permite que un programa continúe ejecutándose mientras espera la finalización de ciertas tareas.

Hay veces en donde tenemos que hacer procesamientos largos o nos tenemos que comunicar con servidores que no sabemos si nos van a contestar o cuánto van a tardar. En estos casos, muchas veces no tiene sentido que nuestra aplicación se congele hasta que esas actividades se completen.

Hay más de una manera de escribir asincronías en JS, detallo las más utilizadas:

El **.then()** se puede colocar al final del llamado de una promesa para indicar mediante una función de flecha qué se debe ejecutar al recibir la resolución de la promesa.

//Se suele hacer un salto de línea antes del punto del then.

fetch("https://api.com/user/2")

.then((res)=> console.log("Imprimo la respuesta",res));

Para evitar el encadenamiento de muchas promesas con muchos **.then** y generar un código más fácil de leer, se crearon el **async/await.**

* **async**: es una palabra reservada que escribimos antes de la declaración de una función que queremos que sea asíncrona.
* **await**: es una palabra reservada que escribimos antes de una instrucción que queremos esperar que se complete antes de continuar con la ejecución del código.

async function getUser2(){

res = await fetch("https://api.com/user/2");

console.log("Imprimo la respuesta",res));

}

***Fetch*** es una función de JavaScript que permite realizar solicitudes HTTP para obtener o enviar datos a un servidor de manera asíncrona. Se utiliza principalmente para interactuar con APIs y manejar datos en aplicaciones web. 14 El primer parámetro de la función Fetch siempre debe ser la URL. Fetch entonces toma un segundo objeto JSON con opciones como method, headers, request body, etc.

**NodeJs: NODE js**

Node.js es un entorno de tiempo de ejecución de JavaScript de back-end de código abierto y multiplataforma que se ejecuta en el motor Chrome V8, es decir, que podemos ejecutar scripts de javascript sin tener un navegador. Instalar node es como “instalar javascript” en nuestra compu para poder ejecutar los archivos .js sin tenerlos cargados a una página.

Es ***asíncrono y no bloqueante,*** lo que significa que puede manejar múltiples operaciones de entrada y salida (I/O) sin detenerse en cada una de ellas, lo que lo hace altamente eficiente para aplicaciones que manejan muchas solicitudes simultáneamente.

Crear un proyecto con node nos abre las puertas a un desarrollo muchísimo más completo y profesional, incluyendo la posibilidad de usar los [FrontendFrameworks](https://gonzalo-bechara.notion.site/Frontend-Frameworks-e724463242a249dc83e17caa57ad72a1?pvs=24) que cambian completamente la manera de hacer web development.

NPM: permite crear los archivos necesarios para gestionar los desarrollos con Node o JavaScript en general, generando un archivo llamado package.json con metadatos del proyecto. Al instalar Node, también vamos a instalar NPM que es el manejador de paquetes de Node. Esto nos va a permitir agregar librerías a nuestro proyecto de manera rápida y mantenerlas descargadas junto con el proyecto.

*Ejemplo de instalar librerias*:

//Instalación normal

npm i <librería>

npm install <librería>

**TypeScript:**

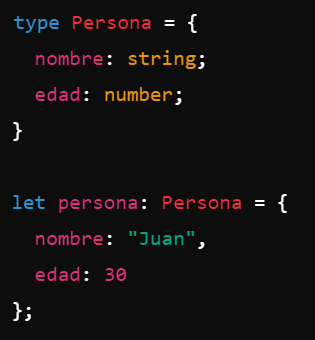
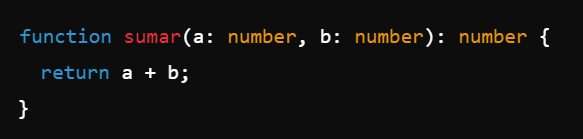
Es un lenguaje de programación basado en Javascript , creado para agregar capacidad de tipado a Javascript. Añade tipado estático, lo que permite detectar errores de forma temprana y mejorar la autocompletación en los IDEs. Aunque los navegadores no interpretan TypeScript directamente, el código debe ser compilado a JavaScript para poder ejecutarse.

*Características:*

* Es un superset de Javascript, por lo cual todas las funciones de JavaScript son válidas en TS.
* Es un lenguaje tipado, por lo cual todas nuestras variables tendrán un tipo de dato predefinido.
* Permite utilizar el tipado *ANY* que admite cualquier tipo de dato (si lo tenés que usar es porque estás haciendo algo mal).
* Es un lenguaje creado para el desarrollo, pero no para la ejecución, ya que los navegadores no interpretan TS, sólo pueden leer JS.
* Es un lenguaje que debe ser compilado a Javascript para que los navegadores lo puedan utilizar, por lo cual requiere de un entorno de ejecución como NodeJs para ejecutar comandos que nos permitan mostrar nuestro trabajo en un navegador web o construir el proyecto.
* Al agregar tipado, permite que los IDE con Intellisense puedan autocompletar código y detectar errores, reduciendo la cantidad de bugs de nuestro desarrollo.

En TypeScript, el tipo se define para especificar los tipos de datos de las variables, que pueden ser tipos de datos integrados, definidos por el usuario u otros tipos de datos definidos antes de declarar cualquier variable que se utilice como entrada en el programa.

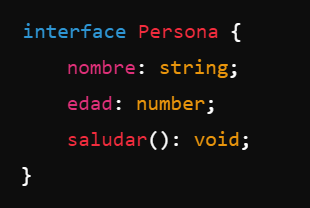
Todos los valores tienen un tipo, aunque no siempre es necesario escribirlo explícitamente, ya que el lenguaje puede deducirlo automáticamente. Es obligatorio anotar el tipo en los argumentos de las funciones, pero cuando se trabaja con objetos, puedes definir el tipo del objeto completo sin necesidad de detallar cada propiedad, ya que TypeScript también puede inferir esos tipos.

 Objetos

Funciones

Las *interfaces* en TypeScript permiten definir la forma que deben tener los objetos, permitiendo describir qué propiedades y métodos deben estar presentes sin especificar su implementación. Las interfaces ayudan a asegurar que los objetos sigan una estructura definida.

En Typescript tenemos la posibilidad de crear nuestros propios tipos de datos gracias a las Interfaces y Types.

* Las interfaces pueden ser extendidas o implementadas por clases, mientras que los tipos no.
* Los tipos son más flexibles y pueden representar uniones o intersecciones de tipos, mientras que las interfaces están más orientadas a describir la forma de los objetos.

Ejemplo:

* Una propiedad “nombre” de tipo string.
* Una propiedad “edad” de tipo number.
* Un método “saludar” que no retorna nada (void).

*Tipos válidos en TypeScript:*

* **string**: Representa cadenas de texto, como palabras o frases.
* **number**: Representa números, tanto enteros como de punto flotante.
* **boolean**: Representa valores lógicos, ya sea true o false.
* **null**: Indica la ausencia intencional de un valor.
* **undefined**: Representa una variable que ha sido declarada pero no inicializada.
* **any**: Permite que una variable contenga cualquier tipo de dato, desactivando las verificaciones de tipo.
* **unknown**: Similar a any, pero obliga a un análisis más detallado antes de su uso.
* **void**: Usado principalmente en funciones para indicar que no devuelven un valor.
* **never**: Indica que una función no regresa ningún valor, normalmente porque lanza un error o entra en un bucle infinito.
* **array**: Representa una lista de elementos, donde todos los elementos son del mismo tipo.
* **tuple**: Un array con un número fijo de elementos de tipos específicos.
* **enum**: Define un conjunto de constantes con nombre.
* **object**: Representa valores no primitivos como arreglos, funciones o instancias de clases.
* **union**: Permite que una variable sea de más de un tipo, por ejemplo, string | number.
* **literal**: Restringe los valores posibles a un conjunto específico.

A tener en cuenta sobre tipos:

Existen tipos inferidos y obligatorios, que son dos enfoques diferentes para manejar los tipos de las variables y funciones:

* Inferidos: TypeScript tiene la capacidad de inferir automáticamente el tipo de una variable basado en el valor que se le asigna, lo que significa que no es necesario declarar explícitamente el tipo.
* Ejemplo: let nombre = "Juan" → TypeScript infiere que "nombre" es de tipo string (no es necesario aclararlo)
* Obligatorios: puedes especificar explícitamente los tipos de las variables, parámetros o funciones, lo que hace que el tipo sea obligatorio. Esto garantiza que las variables o valores deben ser del tipo especificado, y el compilador arrojará un error si no se cumple.
* Ejemplo: let nombre: string = "Juan" → El tipo "string" es obligatorio

Importante: todos los valores en typescript tienen un tipo. Importa saber que las claves son strings pero que los valores son cualquier cosa.

**¿Qué son los eventos?** Un evento es una acción o suceso que ocurre en el navegador, lo que puede ser generado por el usuario (como un clic o una pulsación de tecla) o por el propio sistema (como el final de la carga de una página).

* **Ejemplos de eventos**:
* **Eventos de ratón**: click, hover, dblclick, mouseenter, mouseleave
* **Eventos de teclado**: keydown, keyup, keypress
* **Eventos de formulario**: submit, change, focus, blur
* **Eventos de carga**: load, DOMContentLoaded

**¿Cómo se suscriben los eventos?**

Para manejar los eventos, es necesario **suscribirse** a ellos, lo que significa que se deben registrar funciones que se ejecutarán cuando el evento ocurra.

*Métodos para suscribirse a eventos*

* **addEventListener**: Este es el método más utilizado para agregar un manejador de eventos a un elemento.
  + Sintaxis: element.addEventListener (event, handler, options);
  + Parámetros:
    - event: el tipo de evento a escuchar (por ejemplo, 'click').
    - handler: la función que se ejecutará cuando se dispare el evento.
    - options (opcional): un objeto que puede contener opciones como once, capture, y passive.
* **onclick, onmouseover, etc.:** Se pueden usar propiedades de eventos en el objeto DOM, aunque esta forma es menos recomendada porque solo permite un manejador por evento.
* **removeEventListener**:Si necesitas eliminar un manejador de eventos que previamente has agregado, puedes usar el método.

**Figma**

Figma es una herramienta freemium de diseño de interfaces y experiencias de usuario (UX-UI). Nos permite prototipar nuestra aplicación sin escribir una sóla línea de código.

Algunas ventajas de utilizar una aplicación como Figma o Adobe XD son:

* Diseñar utilizando herramientas de diseño, en vez de diseñar con lenguajes de desarrollo.
* Desacoplar el equipo de diseño con el de desarrollo. Los diseñadores no tienen porqué saber programar su diseño.
* Presentar un prototipo de nuestra aplicación a un cliente (o a nosotros mismos) para verificar que el flujo de la aplicación es la esperada.
* Permite acelerar el proceso de implementación de un diseño, al tener una vista correcta de cómo se debería ver nuestra aplicación.

**Angular**

Angular es un framework de desarrollo de aplicaciones web de código abierto, mantenido por Google. Se utiliza para construir aplicaciones de una sola página (SPA, por sus siglas en inglés) y permite crear interfaces de usuario dinámicas y altamente interactivas**.**

Es un *framework* de front end. Nos permite desarrollar front end de una manera más robusta gracias a las herramientas que nos provee.

Dentro de estas herramientas podemos encontrar:

1. **Componentes:** Son las unidades fundamentales en Angular, responsables de gestionar una parte específica de la interfaz de usuario. Los componentes encapsulan el código, HTML, CSS y la lógica asociada para cada parte de la aplicación. Los componentes tienen un selector. Cada componente tiene su propia lógica, vista y estilo.
2. **Servicios:** Se encargan de la lógica de negocio y la comunicación entre diferentes componentes. Se suelen utilizar para realizar llamadas a APIs, manejar datos y compartir información entre componentes. No es más que una clase, son un poco más simples de entender porque no tienen nada. Provee algunos métodos y maneja algo internamente. Es un solo archivo a comparación de las otras tres; en este caso solo se crea un ts. Los servicios se inyectan.
3. **Módulos**: Los módulos en Angular agrupan componentes, directivas y servicios relacionados en una unidad. Los módulos permiten una mejor organización del código y la reutilización de funcionalidades.
4. **Router:** El sistema de navegación en Angular que enlaza los distintos componentes y módulos. Define qué ruta (path) carga qué componente, permitiendo la creación de aplicaciones de una sola página (SPA).
5. **Signals:** Una característica avanzada de Angular utilizada para manejar el estado y las interacciones en tiempo real dentro de la aplicación.
6. **Specs:** Son los archivos de prueba (tests) que Angular genera automáticamente cuando creamos un componente, servicio o módulo. Estos se utilizan para realizar pruebas unitarias (unittesting).
7. **Inputs:** Se utilizan para crear campos interactivos en un formulario, permitiendo a los usuarios ingresar y enviar datos al servidor, y son fundamentales para la interacción del usuario con las aplicaciones web:

* Permiten a los usuarios ingresar datos, que pueden ser de diversos tipos, como texto, números, correos electrónicos, contraseñas, fechas, etc.
* Interactividad: Facilitan la interacción del usuario con la aplicación. Los usuarios pueden responder a preguntas, hacer selecciones, o completar formularios.
* Accesibilidad: Ayudan a hacer las aplicaciones más accesibles al permitir que los usuarios con diversas capacidades ingresen datos mediante teclados, lectores de pantalla y otros dispositivos de asistencia.

Los elementos <input> pueden ser de varios tipos, cada uno con características específicas:

* **text:** Para texto libre.
* **password**: Para contraseñas, oculta la entrada del usuario.
* **email:**Para correos electrónicos, incluye validación de formato.
* **number:** Para ingresar números, permite establecer límites.
* **date:**Para seleccionar fechas con un selector de fecha.
* **checkbox:**Para seleccionar opciones binarias.
* **radio:**Para seleccionar una opción entre varias.
* **file:**Para cargar archivos.

*Estructura de un Proyecto en Angular:*

Mantener una estructura de proyectos nos permite que nos sea fácil encontrar donde se encuentra cada recurso de nuestra app cuando trabajamos en un proyecto grande. También baja la barrera de entrada de cualquier dev a nuestro proyecto, ya que el mismo se va a poder de manera rápida en un ambiente estructurado.

Un proyecto Angular está compuesto por varios archivos y carpetas que organizan los componentes, servicios, módulos, y otros recursos necesarios para la aplicación. Los servicios se utilizan para manejar lógica de negocio y se inyectan en los componentes. Los componentes tienen un selector que los identifica y permite su uso dentro de otras plantillas.

*Archivos principales en un proyecto Angular:*

1. **src/app/app.module.ts**: Define todos los módulos de la aplicación, incluyendo `FormsModule`, `HttpClientModule`, y registra los servicios globales.
2. **component.ts**: es el archivo typescript que contiene la lógica del componente, es decir, el código que define su comportamiento. (se definen métodos y propiedades del componente)
3. **component.html**: Plantilla HTML del componente raíz que define la estructura visual en el navegador.
4. **component.css o component.scss**: Contiene los estilos CSS personalizados para el componente.
5. **src/app/app-routing.module.ts**: Configura las rutas de navegación entre los distintos componentes de la aplicación.
6. **header.component**: Componente Header: es una parte integral de la interfaz de usuario de una aplicación web. Su función principal es:

* Navegación: Incluye enlaces a diferentes secciones de la aplicación.
* Identidad de marca: Suele contener el logo de la empresa o la aplicación.
* Acceso rápido: Puede incluir elementos como botones de inicio de sesión, perfil de usuario y notificaciones.

*Otros archivos importantes:*

* **angular.json:** Archivo de configuración del proyecto. Define cómo construir la aplicación y qué estilos se deben incluir.
* **package.json:** Define las dependencias y scripts del proyecto.
* **tsconfig.json:** Configuración del compilador TypeScript.
* **styles.css:** Archivo global de estilos que se aplica a toda la aplicación.

*Angular CLI:* es una herramienta de línea de comandos que facilita la creación, configuración y mantenimiento de aplicaciones Angular. Proporciona comandos para automatizar tareas comunes de desarrollo.

Comandos importantes de Angular CLI:

* **Crear un nuevo proyecto Angular**: ng new <nombre-proyecto>
* **Servir la aplicación en modo de desarrollo**: ng serve
* **Generar un componente**: ng generate component <nombre-componente>
* **Generar un servicio**: ng generate service <nombre-servicio>
* **Compilar la aplicación**: ng build --prod

*Inyección de servicios*: Los servicios se inyectan en los componentes o en otros servicios mediante inyección de dependencias. El decorador @Injectable() se utiliza para marcar una clase como un servicio que puede ser inyectado.

*Servicios principales en un proyecto Angular:*

* **service.ts:** Es el archivo principal donde se define el servicio. Aquí se implementan métodos que proporcionan lógica reutilizable y se inyectan en componentes o en otros servicios.
* **service.spec.ts**: Es el archivo para las pruebas unitarias del servicio. Se usa para validar la lógica del servicio, similar a cómo se prueban los componentes.

*Características principales de Angular:*

* Utiliza Typescript para su desarrollo, lo cual lo posiciona mejor para proyectos de gran escala.
* Es más completo que otros frameworks, permitiendo que se pueda programar toda una web app sin instalar otras dependencias más que las que incluye por defecto.
* Tiene una comunidad muy grande y activa, aumentando el tamaño de recursos que podemos encontrar online.
* En su configuración por defecto sirve para desarrollar **SPA**s o Single Page Applications, que son web apps que no se recargan completamente al navegar entre distintas páginas, sino que pueden actualizar el contenido de una página sin recargar los elementos que ya se encuentran cargados. Esto lo convierte en un creador de aplicaciones con **CSR** (client side rendering) en donde el software de lógica de la vista se realiza en el dispositivo cliente; sin embargo también puede crear sitios con SSR.
* Nos provee un entorno que facilita el testing unitario y testing e2e.
* Incluye una herramienta CLI, que nos permite desarrollar de manera más rápida.

*Conceptos adicionales en Angular:*

* **Fetch**: Utilizado para realizar solicitudes HTTP, pasando una URL y encabezados (headers). Los encabezados se utilizan para pasar información adicional en la solicitud, como el tipo de contenido (Content-Type) o tokens de autenticación (Authorization).
* **ngOnInit**: Método que se ejecuta cuando un componente se inicializa. Es ideal para hacer llamadas a servicios.
* **Local Storage**: Utilizado para almacenar datos persistentes en el navegador del usuario, permitiendo que la aplicación conserve información entre sesiones.
* **Templates y \*ngFor**: Permiten crear la estructura visual en Angular, iterando sobre datos con \*ngFor para mostrar listas de forma dinámica.
* **ngModel**: Permite la vinculación bidireccional de datos (two-way data binding) entre un elemento de formulario y una propiedad del componente en Angular. Esto significa que cuando se actualiza una variable, su valor se refleja automáticamente en el formulario, y cualquier cambio en el formulario se refleja en la variable. Es útil para manejar formularios, ya que permite verificar la validez de los datos y hacer validaciones fácilmente.
* **NgIf:** Directiva estructural que muestra u oculta un elemento basado en una condición booleana.
* **NgClass:** Es una directiva de Angular que permite agregar o quitar clases CSS dinámicamente en un elemento. Funciona mediante un objeto en el que las claves son los nombres de las clases y los valores son booleanos o expresiones que se evalúan como `true` o `false`. Esto permite aplicar o eliminar clases según condiciones específicas del programa. Es necesario usarlo con una condición, ya que cuando la condición es `true`, la clase se aplica; de lo contrario, se omite.
* **NgStyle:** Directiva atributiva que permite aplicar estilos dinámicamente a un elemento.
* **ActivatedRoute:** permite acceder a la información de la ruta activa, como los parámetros o el estado de la URL. (route)
* **RoutingModule:** es donde se definen las rutas de la aplicación. Se crea un módulo separado para configurar las rutas. (route)
* **RouterLink:** es una directiva que se utiliza para enlazar rutas dentro de la aplicación. Genera automáticamente enlaces a las rutas configuradas en el Router. Se usa para navegar entre páginas dentro de una SPA. Mediante el uso de router Link y rutas definidas en routers, se pueden manejar rutas y ubicaciones. Cuando tenemos un botón que te lleva a otra página, usamos el router link y el path con corchetes, dentro de ellos se escribe alguna ubicación lógica dentro de la página (rutas definidas en el routes).
* **@if:** Permite verificar si el usuario tiene privilegios de administrador. Si la condición es verdadera, se muestra un menú con opciones adicionales que solo los administradores deberían ver, como configuraciones avanzadas o herramientas de gestión. Esto ayuda a personalizar la experiencia del usuario y a mantener una interfaz más limpia y relevante según el rol del usuario en la aplicación.

*Auth (Autenticación)*

La autenticación es un proceso crucial en aplicaciones web que protege los recursos y asegura que solo los usuarios válidos tengan acceso a funcionalidades específicas. Generalmente implica:

* Registro e inicio de sesión: Los usuarios se registran y crean credenciales que son verificadas en cada solicitud.
* Gestión de sesiones: Puede incluir el uso de tokens (JWT) o cookies para mantener la sesión activa.
* Control de acceso: Dependiendo del rol del usuario (administrador, usuario regular), se restringen o permiten ciertas acciones.

Autenticación con Fetch y Bearer Token: Para realizar solicitudes fetch, se usa el método de autenticación Bearer, que es de tipo portador. Cada fetch necesita información como:

* La composición de la URL (esquema de la URL).
* El método de acceso.
* El cuerpo de la solicitud.
* Encabezados necesarios, siendo el encabezado más común el de Authorization.

**Git y GitHub**

Git: es una herramienta que permite mantener un control exacto del versionamiento de un proyecto. (Sistema de control de versiones - CVS)

Gracias a git podemos:

* Tener un registro de todas las versiones de nuestro proyecto a través del tiempo.
* Ver las líneas de código que cambiaron versión a versión.
* Manejar el código en distintas ramas representando distintos ambientes o grupos de trabajo.
* Trabajar en equipo en el mismo proyecto sin mayores problemas.
* Trackear las contribuciones de distintas personas al proyecto.

GitHub: Es una plataforma para alojar proyectos Git en la nube. Proporciona almacenamiento gratuito, herramientas colaborativas, y facilita el despliegue de aplicaciones web mediante GitHub Pages.

***SWEETALERT 2***

Es una popular librería para crear modales atractivos y altamente personalizables en aplicaciones web. Permite presentar alertas y formularios de manera más visual y amigable que los diálogos nativos del navegador. Algunas de sus características incluyen:

* *Estilos personalizables:* Puedes modificar colores, tamaños y comportamientos.
* *Soporte para promesas*: Permite manejar respuestas de usuario (confirmar, cancelar) fácilmente.
* *Contenido enriquecido:* Puedes incluir texto, imágenes, formularios y botones personalizados.

Para mostrar un modal, se puede invocar Swal.fire() con diferentes opciones como título, texto, íconos y botones. Esto facilita la interacción del usuario y mejora la experiencia de usuario.

**PREGUNTAS que dijo el profesor:**

* ¿Cuáles son las diferencias entre Single y Multiple page applications?
* ¿Herramientas, eventos, semántica html?
* ¿A qué nos referimos con asincronía?
* ¿Tipos en Typescript (inferidos y obligatorios)? ¿Interfaces?
* ¿Para qué se usa la etiqueta input?
* ¿A qué elementos afecta un background color?
* ¿Para qué se usa ng model?