No es el fin, es el comienzo

Hemos llegado a la última clase de nuestro curso y tras haber aprendido mucho sobre Javascript podemos afirmar que ésto recién comienza.

En nuestro recorrido hemos atendido conceptos como:

* Variables y tipos de datos
* Control de flujos e iteradores
* Funciones, parámetros, métodos.
* Objetos, clases y Arrays.
* DOM, acceso y manipulación, Storage.
* Manejo de Eventos.
* Operadores avanzados de JS.
* Librerías.
* Asincronía y Promesas.
* Peticiones HTTP, APIs y Fetch.

Sin embargo queda mucho por recorrer aún y ésta clase trata de los caminos que podemos tomar a partir de este punto. En el mercado laboral actual los perfiles más demandados con JS son: backend con NodeJS, o frontend con dominio de algún framework/librería como ReactJS, Angular, o VueJS.

Veremos algunas herramientas y conceptos generales para que puedas orientarte mejor en tu siguiente decisión!

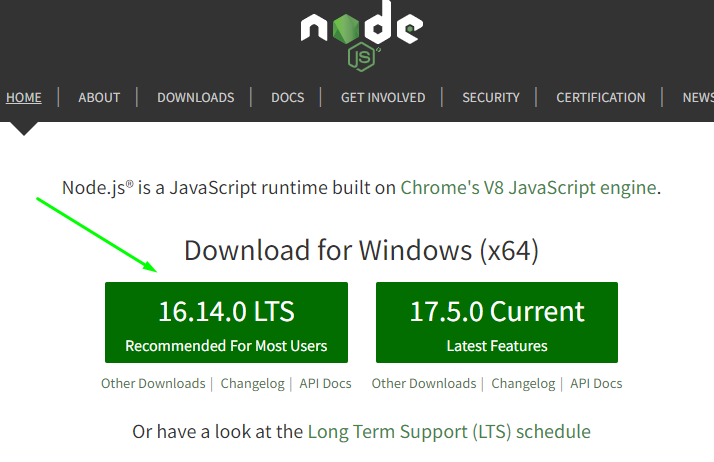
NodeJS

Recordemos que Javascript es un lenguaje *interpretado* que se ejecuta a través de un programa adicional o entorno de ejecución. Hasta el momento todo lo que hemos desarrollado se ejecuta en *un mismo entorno, el navegador*.

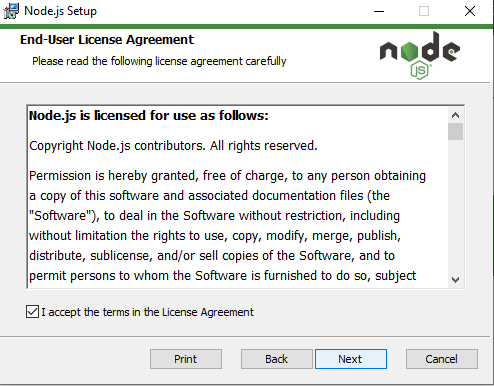
Hay un entorno adicional de ejecución para Javascript llamado **NodeJS**. Éste es un programa que permite ejecutar código JS por fuera del navegador. Con él podemos construir desde aplicaciones de línea de comandos (consola) hasta servidores HTTP para potenciar aplicaciones web. Está construido usando el motor V8 de Chrome.

**Instalación**

El proceso es sencillo y accesible. Ingresa a la página de [Node](https://nodejs.org/en/) y descarga el instalador de la versión LTS:



Ahora sólo sigue los típicos pasos de instalación:



Una vez finalizado puedes verificar si la instalación fue exitosa ingresando a la terminal de comandos (PowerShell, GitBash, o la que desees), e ingresar el comando **node -v.**

 Debería retornar la versión de Node instalada. Hecho esto, ya estás listo para usar Node.

**El comando *node***

Cuando instalamos Node en nuestro sistema, éste nos provee el comando *node* que sirve para ejecutar archivos JavaScript. Supongamos que tenemos un archivo denominado *coder.js* con lo siguiente:

// coder.js

let mensaje = "Hola Mundo Coder!"

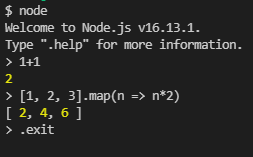
console.log(mensaje)

Luego podemos ejecutar este archivo usando el comando *node* desde cualquier terminal de comandos:



El método *console.log* de Node es similar al que utilizamos en el navegador, con la diferencia que en Node el mensaje se mostrará por la terminal en la cual usamos el comando, en vez de la consola del navegador.

Si utilizamos *node* sin pasar un archivo, nos habilita un prompt para escribir código JS y ver resultados inmediatos:



**Node != Browser**



Node es un entorno diferente al navegador, por lo que no posee los mismos recursos. Las variables globales como *Object, Array,* o *JSON* están disponibles; pero no podemos acceder a recursos provistos por el navegador como el *alert(), prompt()* o el DOM:

En cambio, Node trae otras grandes herramientas que el navegador no posee, como el *File System (sistema de archivos)* o el *módulo HTTP* para crear servidores web.

**Backend con NodeJS**

Node JS ha cobrado mucha popularidad para desarrollar aplicaciones que funcionan del lado del servidor. Es posible escribir con Node programas que escuchen y respondan peticiones HTTP (como las que vimos la clase anterior!), creen y manipulen archivos, interactúen con bases de datos, y demás.

Al trabajar en el back no disponemos de la UI del navegador para ir testeando los resultados, por lo que la principal herramienta para ésto es la *consola* *cli* (command line interface). Con ella podemos ejecutar nuestro programa en el servidor; ver los *logs* que generemos; visualizar errores y mensajes informativos explícitamente generados por Javascript en tiempo de ejecución dentro del contexto del programa del servidor.

**NPM**

NPM (Node Package Manager) es un repositorio de módulos de Node. ¿Recuerdan nuestra clase de librerías? Muchos de esos programas que incorporamos a nuestros proyectos están escritos como *paquetes,* o módulos, *de Node*. El repositorio de NPM es donde se listan y alojan éstos módulos, pudiendo descargarlos e integrarlos a nuestras apps de forma sencilla.

Con la instalación de Node se incluye el comando *npm* que nos permite acceder a este repositorio y administrar estos paquetes.

El comando *npm install* nos permite instalar cualquier módulo disponible en este repositorio:



Éste crea una carpeta *node\_modules* en la cual se alojarán los paquetes instalados. Éstos se pueden utilizar en nuestras aplicaciones a través del sistema de módulos de JS.

***¿Por qué vemos esto?***

Aparte del backend, los principales frameworks de Frontend tienen ambientes de desarrollo que trabajan con Node y con NPM para instalar y manipular módulos y el proyecto en general.

***INIT y package.json***

NPM nos permite administrar los paquetes instalados y guardar información sobre nuestro proyecto. El comando ***npm init*** crea un archivo llamado *package.json* el cual contiene información sobre nuestra app, como el nombre, autor, versión, dependencias, y demás.

Simplemente abrimos nuestro proyecto y corremos el comando *npm init.* Aceptamos todas las opciones y tendríamos un archivo como el siguiente:

{

"name": "coder",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"author": "",

"license": "ISC"

}

En este archivo también se listan todas las dependencias (paquetes) agregadas a nuestro proyecto, las cuales podemos instalar corriendo el comando *npm install*. Esto es útil cuando clonamos otros proyectos de algún repositorio y necesitamos reinstalar todos los paquetes necesarios para que funcione.

En el apartado *scripts* se pueden listar comandos para la CLI a correr a través de npm.

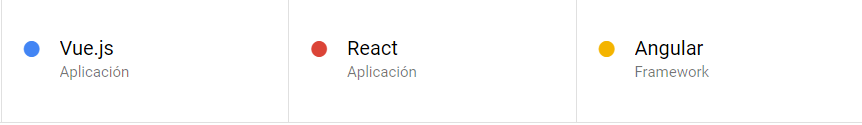
NPM, Node, y el package es algo con lo que trabajarán habitualmente al perfeccionarse en backend o front-end.

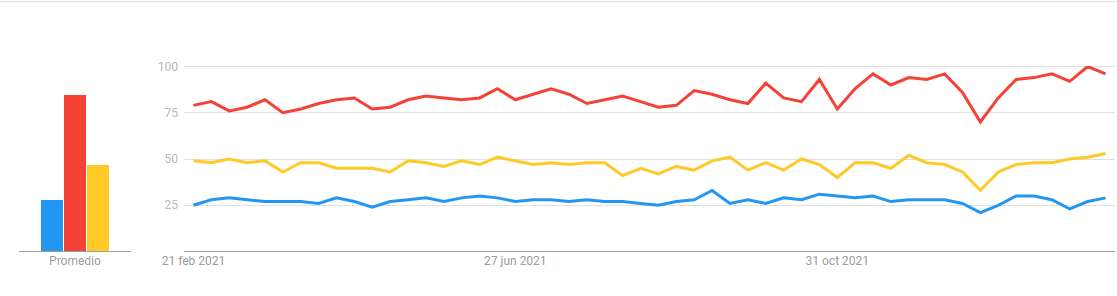
Los caminos del Front

En el mercado laboral actual, resulta indispensable dominar alguna de las librerías/frameworks populares de JS para desarrollar aplicaciones de Front-end. Estas son:

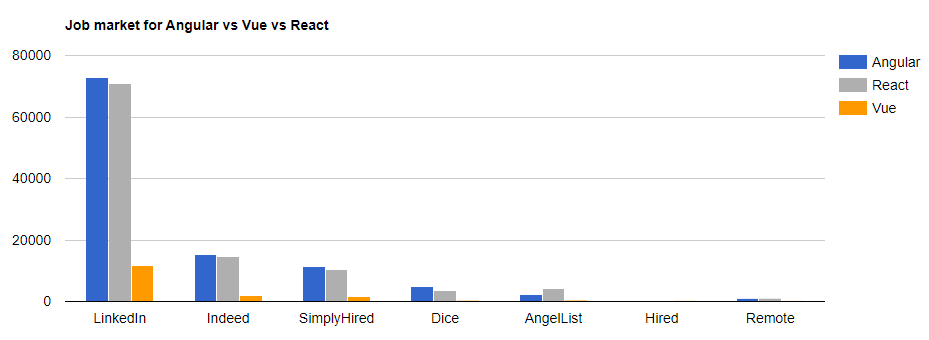
* ReactJS
* Angular
* VueJS

Todos están basados en JavaScript y tienen sus semejanzas y diferencias, aunque la popularidad de cada uno está marcada:





Y ésto se refleja en el mercado laboral, midiendo la cantidad de puestos demandados para cada frame:



ReactJS es el que domina el mercado siendo el más popular de todos y el más demandado, seguido por Angular y VueJS. Todos están basados en JS y sirven para construir aplicaciones web de forma avanzada. Antes de avanzar con los detalles de cada uno, veamos algunos conceptos adicionales

SPA

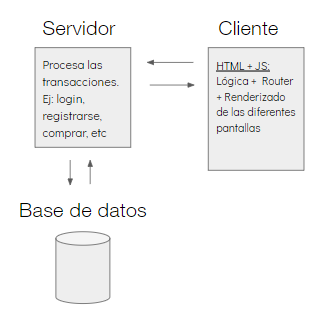
En el desarrollo tradicional de aplicaciones web se suelen tener varios archivos html vinculados a un sitio, los cuales representan las distintas vistas o secciones del mismo.

Una SPA (Single Page Application), o *aplicación de una sola página*, es una web app que consiste de un único archivo html que se carga al ingresar al sitio, y el resto de la navegación y cambios de página se da mediante modificación dinámica del DOM con Javascript.

Tiene el propósito de brindar una experiencia más fluida al usuario, mejorando la performance y la velocidad.

En una SPA todos los códigos de [HTML](https://es.wikipedia.org/wiki/HTML), [JavaScript](https://es.wikipedia.org/wiki/JavaScript), y [CSS](https://es.wikipedia.org/wiki/Hojas_de_estilo_en_cascada) se cargan una sola vez​ o los recursos necesarios se cargan dinámicamente cuando lo requiera la página, normalmente como respuesta a las acciones del usuario. La página no tiene que cargarse de nuevo en ningún punto del proceso.

En resumen, una SPA es más rápida, liviana y flexible que una app tradicional, brindando mejor experiencia de usuario.

**Arquitectura**

La mayor parte de la funcionalidad de la aplicación queda en el cliente.

Se accede al servidor usando AJAX para obtener los datos necesarios para su funcionamiento, generalmente una API REST. Toda la información necesaria para que la aplicación funcione se cargará en el cliente durante la petición inicial y el acceso a diferentes páginas de la aplicación las maneja el **router**.

**Router**

El router, o ruteador, dentro de una aplicación SPA cumple la función de controlar e interpretar cada solicitud que el usuario haga en la página. En una SPA los cambios de url *no significan cargar nuevos archivos html*, sino que son captados por un *router* que entiende qué modificar del DOM para representar una nueva vista.

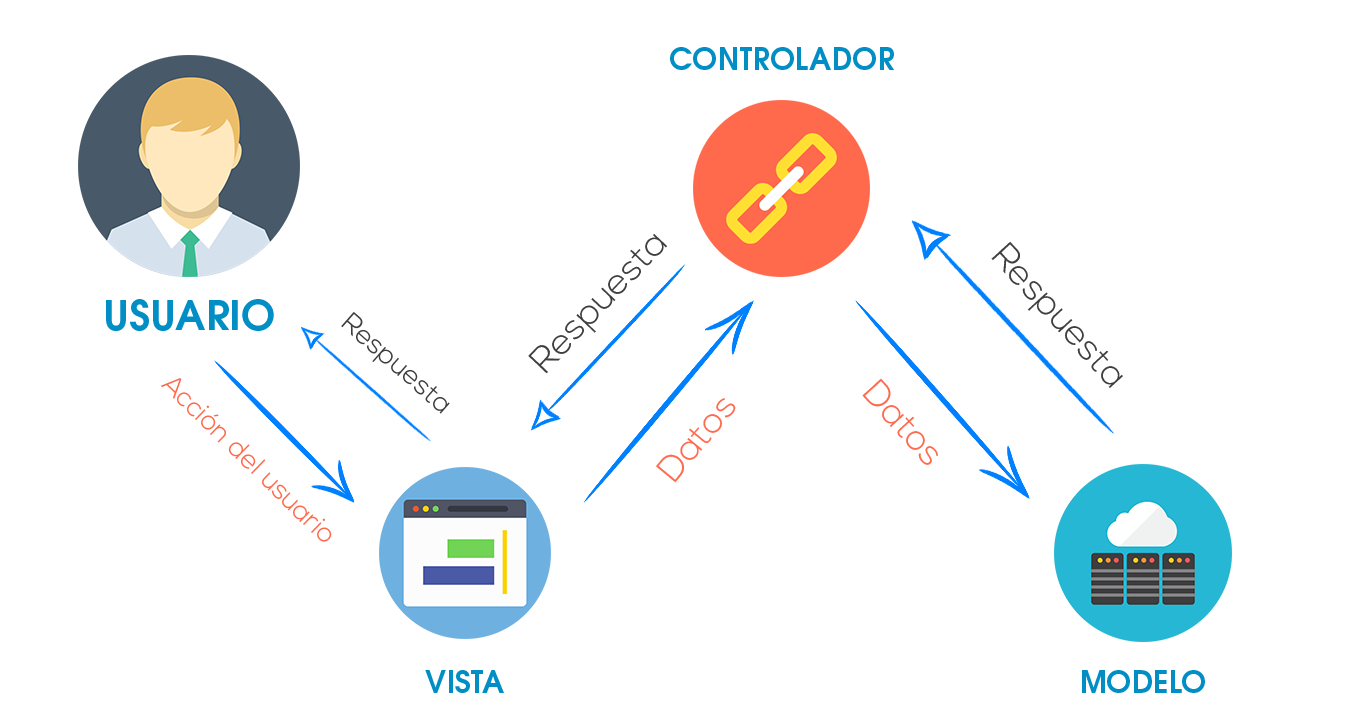
Ese cambio de URL puede implicar realizar al servidor una solicitud de los datos necesarios para luego renderizar mediante JS la nueva página que se mostrará.  
Durante este proceso, el usuario no verá recargar la pestaña del navegador, sino que todo el proceso ocurre de manera asincrónica.

**Modelo-Vista-Controlador (MVC)**

El MVC es un patrón de arquitectura de software que separa los datos de una aplicación de su representación, y el módulo encargado de gestionar los eventos y la comunicación.

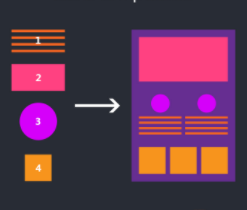
Este patrón define por un lado componentes para la representación de la información, y por otro lado para la interacción del usuario.

* El **modelo**: es la representación de la información con la cual el sistema opera y se encarga de gestionar su acceso. Envía a la 'vista' aquella parte de la información que se le solicita. Las peticiones de acceso o manipulación de información llegan al 'modelo' a través del 'controlador'.
* El **controlador:** responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información. También envía comandos a la ‘vista’. Se puede decir que el 'controlador' hace de intermediario entre la 'vista' y el 'modelo'.
* La **vista**: presenta el 'modelo' en un formato adecuado para interactuar (UI), por tanto requiere de dicho 'modelo' la información que debe representar como salida.



**Orientación a componentes**

Angular, React y Vue trabajan con componentes. Un componente es un conjunto de elementos que conforman una unidad funcional (como un header, un footer, un carousel, etc.). Pensar en componentes permite dividir nuestra app en diferentes piezas y trabajarlas por separado, integrándolas luego en una estructura mayor para formar la app.



Este enfoque nos permite separar la interfaz de usuario en piezas independientes y reutilizables. Además nos permite desarrollar la lógica de comportamiento de cada componente de forma modular, sin interferir con la lógica de otros.

Significa que un componente se compone tanto por los elementos que lo representan como por la lógica que le da funcionamiento.

ANGULAR - VUE - REACT

**ANGULAR**

Desarrollado por Google, se lanzó primero en 2010, siendo el más viejo de los tres. Es un framework de Javascript basado en Typescript. En 2016 se lanza Angular 2 con cambios sustanciales.

La última versión es Angular 13 publicada en noviembre de 2021.

Angular utiliza un sistema de inyección de dependencias jerárquico impulsando su rendimiento.

También implementa la detección de cambios basados en árboles unidireccionales.

Angular posee una curva de aprendizaje empinada. Como framework, ofrece una solución integral y completa para desarrollar aplicaciones, por lo que saber Angular permite construir aplicaciones complejas en corto tiempo. Dominar Angular implica conocer y aprender conceptos como MVC y Typescript. Lleva tiempo aprenderlo, pero vale la pena ya que abarca casi todas las aristas del desarrollo front-end.

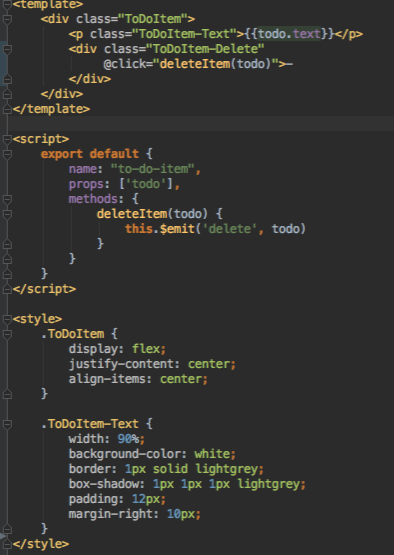
Angular es una gran opción para equipos grandes y desarrolladores que manejen Typescript.

*“Angular nos ofrece desarrollo de principio a fin sin salir del ecosistema”*

**VUE**

Vue, conocido también como Vue.js, es el más joven de los tres, desarrollado por Evan You en 2014. En los últimos años ha dado un gran salto en popularidad, a pesar de no tener el apoyo de grandes compañías como Facebook o Google. La última versión estable es la 3.0, lanzada en 2020.

Vue JS es un framework javascript que incorpora un conjunto de herramientas y funciones para un desarrollo dinámico de páginas y aplicaciones web. Su concepto nació para cubrir la necesidad de no tener que escribir tanto código JS, garantizando así acotar los tiempos del programador.

Vue permite crear aplicaciones SPA altamente flexibles y dinámicas. En Vue, la vista (UI) y el comportamiento (lógica) forman parte de los componentes. Te permite combinar la estructura de un componente, con lógica y estilos en un mismo script.

Es altamente customizable lo que lo vuelve más fácil de aprender que React o Angular. Vue es una gran opción si te gusta la simplicidad y flexibilidad, aunque pueden jugar en contra ya que permite escribir código torpe o desprolijo.

*Framework progresivo*

Los frameworks progresivos como Vue JS, a diferencia de los comunes, contienen todas sus funcionalidades principales en una librería muy pequeña y permiten ir añadiendo otras librerías a medida que se necesiten. De esta forma, podemos ir incorporando a nuestro proyecto con Vue JS solo aquellas librerías que prestan funciones que necesitemos, lo cual permitirá construir aplicaciones web realmente livianas que cuenten con un rendimiento verdaderamente óptimo.

**REACT**

React, desarrollado por Facebook, se lanzó originalmente en 2013, y es usado por facebook ampliamente en sus productos (Fb, IG, WP). La versión estable actual es la 17.x. 

React combina la lógica y la vista de los componentes, y su integración entre ellos. Es decir que en el mismo bloque de código define la estructura del elemento y su lógica.

La documentación de React es exhaustiva y completa, ofreciendo guías prácticas que facilitan su aprendizaje.

React no es un framework completo, y funcionalidades avanzadas requieren la instalación de librerías adicionales de terceros. Sin embargo esto lo vuelve customizable, permitiendo crear aplicaciones de todo tipo y complejidad, sumado a su flexibilidad y performance lo vuelven el más popular del grupo.

React es una buena opción para introducirse a los frameworks de Javascript y para aquellos que gusten de la flexibilidad.

Fullstack con Javascript

**MERN, MEAN y MEVN stack**

La sigla MERN viene de *MongoDB, Express, React, Node.* Puedes reemplazar React por Angular o Vue, y eso da MEAN y MEVN.

Estos stacks se componen de tecnologías desarrolladas enteramente en Javascript:

* **MongoDB:** es un sistema de base de datos NoSQL orientado a documentos y de código abierto.
* **Express:** es un framework para Node.js que sirve para crear aplicaciones de backend en menos tiempo.
* **React/Vue/Angular:** frameworks de frontend para crear aplicaciones web.
* **NodeJS**: entorno en tiempo de ejecución multiplataforma para la capa del servidor basado en JavaScript.

Al estar todo desarrollado en Javascript, nos permite profundizar en un solo lenguaje de programación, logrando así enfocar y reforzar nuestros conocimientos y con ello ser más productivos, pudiendo desarrollar perfiles de front y back en menos tiempo.

