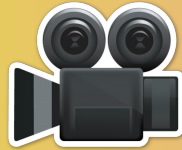




***¡LES DAMOS LA
BIENVENIDA!***

¿Están listos?

RECORDÁ PONER A GRABAR LA CLASE

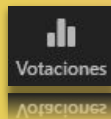


PRESENTACIÓN DE ESTUDIANTES



Por encuestas de Zoom:

1. País
2. Conocimientos previos en la disciplina
3. ¿Por qué elegiste el curso?



Se responde en encuestas separadas creadas por el docente.



¿DUDAS DEL ON-BOARDING?

MIRALO AQUI

ACUERDOS Y COMPROMISOS

Convivencia:



- ✓ Chequea aquí nuestro [código de conducta](#) y ayúdanos a generar un ambiente de clases súper ameno.
- ✓ Al momento de interactuar (en las clases u otros espacios), ten en cuenta las [normas del buen hablante y del buen oyente](#), que nunca están de más.
- ✓ Durante las clases, emplea los medios de comunicación oficiales para canalizar tus dudas, consultas y/o comentarios: **chat Zoom público y privado y Slack.**
- ✓ Verifica el estado de la **cámara y/o el micrófono** (on/off) de manera que esto no afecte la dinámica de la clase.

ACUERDOS Y COMPROMISOS

Distractores:



- ✓ Encuentra tu espacio y crea el momento oportuno para **disfrutar de aprender**.
- ✓ Evita dispositivos y aplicaciones que puedan **robar tu atención**.
- ✓ Mantén la **mente abierta y flexible**; los prejuicios y paradigmas no están invitados.

ACUERDOS Y COMPROMISOS

Herramientas:



- ✓ Mantén a tu alcance **agua-mate-café**.
- ✓ Conéctate desde algún equipo (laptop, tablet) que te permita **realizar las actividades** sin complicaciones.
- ✓ Si lo necesitas ubica lápiz y papel para que no se escapen las ideas, sin embargo recuerda que en el **Drive** tienes **archivos** que te ayudarán a repasar, incluidas las **presentaciones**.
- ✓ Todas las clases quedarán grabadas y serán compartidas tanto en la **plataforma de Coderhouse** como por **Google Drive**.

ACUERDOS Y COMPROMISOS

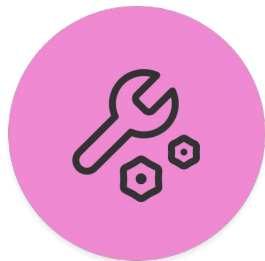
Equipo:



- ✓ **¡Participa de los Afters!** Son un gran espacio para atender dudas y mostrar avances.
- ✓ Intercambia ideas y consultas por el chat de Slack: **entre todos nos ayudamos a mejorar.**
- ✓ Siempre interactúa con otros/as **respetuosamente.**
- ✓ No te olvides de **valorar tu experiencia educativa** y de contarnos cómo te va.

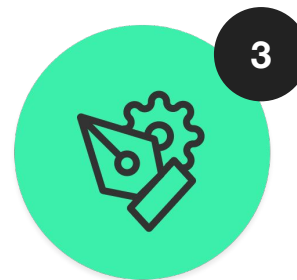
DESAFÍOS Y ENTREGABLES

Son actividades o ejercicios que se realizan durante la cursada, para enfocarse en la práctica.



Desafíos genéricos

Ayudan a poner en práctica los conceptos y la teoría vista en clase. No deben ser subidos a la plataforma.



Desafíos entregables

Relacionados completamente con el **Proyecto Final**. Deben ser subidos obligatoriamente a la plataforma hasta 7 días luego de la clase para que sean corregidos.

DESAFÍOS Y ENTREGABLES

Son actividades o ejercicios que se realizan durante la cursada, para enfocarse en la práctica.



Desafíos complementarios

Desafíos que complementan a los entregables. Son optativos y, de ser subidos a la plataforma a tiempo y aprobados, suman puntos para el top 10.



Entregas del Proyecto Final

Entregas con el estado de avance de tu **proyecto final** que deberás subir a la plataforma a lo largo del curso y hasta 7 días luego de la clase, para ser corregidas por tu docente o tutor/a.



PROYECTO FINAL

El Proyecto Final se construye a partir de los **desafíos** que se realizan clase a clase. Se va creando a medida que el estudiante sube los desafíos entregables a nuestra plataforma.

El objetivo es que cada estudiante pueda utilizar su Proyecto Final como parte de su portfolio personal.

El **proyecto final** se debe subir a la plataforma la ante-última o última clase del curso. *En caso de no hacerlo tendrás 20 días a partir de la finalización del curso para cargarlo en la plataforma. Pasados esos días el botón de entrega se inhabilitará.*

¿CUÁL ES NUESTRO PROYECTO FINAL?



PLATAFORMA DE PEDIDOS ONLINE DE COMIDA

El proyecto final consiste en la creación de un proyecto frontend basado en VueJS para hacer pedidos online de comida.

Deberá contener el **perfil de administrador**, el cual podrá listar, realizar altas y bajas de los productos gastronómicos y modificar sus características. También podrá ver el pedido detallado de todos los clientes.

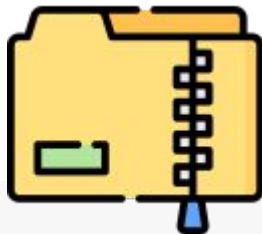


PLATAFORMA DE PEDIDOS ONLINE DE COMIDA

El **perfil cliente** podrá ver el listado de todas las comidas y elegir una de ellas para enviarla a una vista de compra (carrito), y luego de establecer la cantidad que va a encargarse del producto elegido, el sistema calculará el precio a pagar por comida y el total, para la posterior validación y envío del pedido por parte del cliente.

MODELO DE PROYECTO FINAL

Proyecto
Final



- 📁 .git
- 📁 public
- 📁 src
- 📄 .gitignore
- 📄 babel.config.js
- 📄 package.json
- 📄 package-lock.json
- 📄 README.md

[pedidocomidaApp-MOCK](#)

[API.zip](#)



ENTREGA	REQUISITO	FECHA
1° entrega	Crear el proyecto con Vue CLI 2, componentes y la lógica y estructura de datos	Clase N° 7
2° entrega	Definir formularios avanzados, registro de usuarios y servicios http	Clase N° 10
3° entrega	Incorporar vuex para mantener el estado e importar los datos al backend	Clase N° 13
Proyecto final	Crear el proyecto de producción en Vue CLI 2 con Vuex 3 y subir la solución a la nube. Convertir el proyecto realizado a Vue CLI 3 utilizando Vuex 4	Clase N° 16

¡IMPORTANTE!

Los desafíos y entregas de proyecto se deben cargar hasta siete días después de finalizada la clase. Te sugerimos llevarlos al día.

LUNES 16/03 20:30HS

10. Estrategia de contenido para Twitter y LinkedIn

DESAFÍO - EXPIRA EL 23/03/2020

Crear publicaciones para Twitter

👍

● HOY 20:30

📁

Tenes tiempo hasta el 23/03/2020

↑

ENTREGAR



Clase 01. Vue JS

INTRODUCCIÓN Y CONFIGURACIÓN DE HERRAMIENTAS



OBJETIVOS DE LA CLASE

- Conocer el framework.
- Ejecutar las instalaciones de herramientas.
- Incorporar conceptos avanzados de programación en ECMAScript

CRONOGRAMA DEL CURSO

Clase 1



Introducción y configuración de herramientas



REPOSITORIO CON GIT



ECMASCRIPT

Clase 2



Proyecto CDN y primeras directivas



PROYECTO CDN CON HTML, CSS Y JS



USO DE DATA BINDING



CONTADOR

¿QUÉ ES VUEJS?

¿Qué es Vue JS?



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.

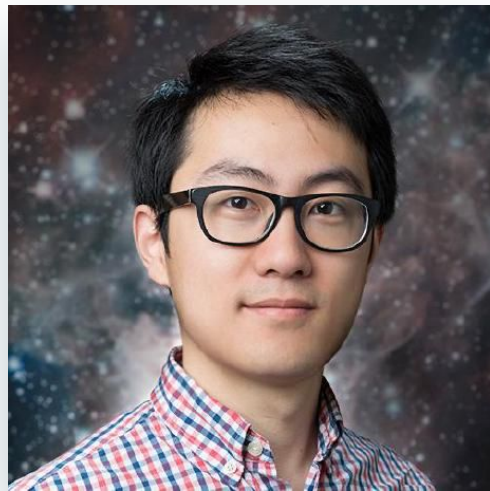
Historia

Fue creado en 2014 por *Evan You*, ex-trabajador de Google.

Su golpe de suerte fue cuando *Taylor Otwell*, creador del framework PHP [Laravel](#), tuiteó que se estaba aprendiendo

Vue porque React le pareció muy confuso.

Desde entonces, Vue JS, escaló rotundamente en popularidad gracias a su sencillez y a todo lo que ofrece.



Te dejamos por aquí el link al [Documental de Vue JS](#). 🧐

CODER HOUSE

USOS Y BENEFICIOS DE VUE

¿Para qué sirve Vue JS?



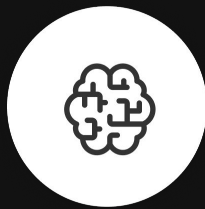
Vue JS te permite crear vistas de tus páginas web, hacerlas dinámicas, conectarlas a un servidor backend para tener datos dinámicos, crear sitios SPA mediante ruteo, entre un sinnúmero de posibilidades más.

Todo esto, simplificando el código JS.

¿Para qué sirve Vue JS?

Todas estas ventajas hacen de Vue JS un **framework completo** **pensado para los programadores web**, con buena curva de aprendizaje y que puede usarse en todo tipo de proyectos web.





¡PARA PENSAR!

*¿Qué otros framework JS conoces?
¿En qué piensas que se distingue de otros framework
conocidos?*

CONTESTA EN EL CHAT DE ZOOM



Pros Vue

- Muy liviano respecto a otros frameworks JS, lo cual le aporta un gran rendimiento.
- Tiene una curva de aprendizaje muy rápida, respecto a Angular o React.
- Su documentación es clara y contiene muy buenos ejemplos prácticos.
- Se integra fácilmente a cualquier proyecto, permitiendo utilizarlo casi de forma transparente.

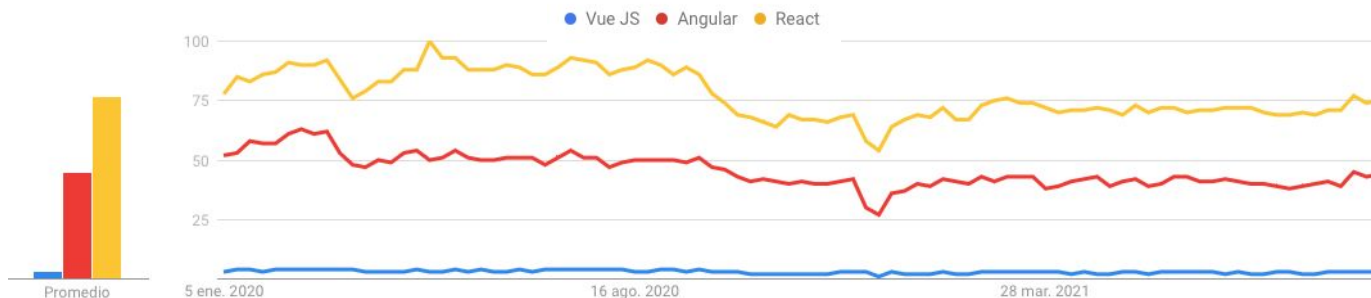
COMUNIDAD

Si bien su comunidad aún es pequeña, en comparación a React o Angular, Vue JS ha crecido significativamente en popularidad, en un período de tiempo muy corto.

Esto le permitió sobrepasar a Frameworks y Librerías JS, que contaban con comunidades afianzadas muchísimos años antes de que Vue aparezca en el Mercado.

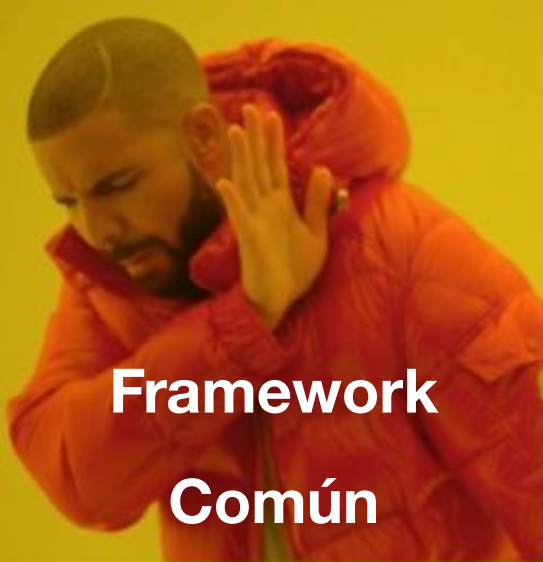
Vue versus Angular & React

Acorde a Google Trends, React y Angular dominan el mercado seguidos, en tercer lugar, por Vue JS. Más allá de esta estadística, no podemos determinar cuál es el mejor de los tres, sin antes tener conocimiento de todas estas tecnologías en el terreno del desarrollo de aplicaciones web.



Fuente: **Google Trends** ([enero 2020 a octubre 2021](#))

FRAMEWORK PROGRESIVO



**Framework
Común**



**Framework
Progresivo**

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.

FRAMEWORK PROGRESIVO

De esta forma, podemos ir incorporando a nuestro proyecto con Vue JS solo aquellas librerías que prestan funciones que necesitemos (*API Rest, Frameworks CSS, etcétera*), lo cual permitirá construir aplicaciones web realmente livianas que cuenten con un rendimiento verdaderamente óptimo.



GIT EN VISUAL STUDIO CODE

GIT en VISUAL STUDIO CODE



Además de contar con una excelente herramienta para la línea de comandos, **GIT se integra muy fácilmente con Visual Studio Code.**

Veamos cómo sacar provecho de este fantástico software de control de versionado sin tener que salir de nuestro editor de código favorito.

GIT en VISUAL STUDIO CODE



Realicemos a continuación un ejercicio para entender las particularidades de Git y cómo está integrado a Visual Studio Code.

Asegúrate de tener [VS Code ya instalado](#), como así también Git. Si no tienes este último, [descárgalo desde su web oficial](#), según tu sistema operativo.



EJEMPLO EN VIVO

Veamos las particularidades de Git y cómo se integra a Visual Studio Code.



Creación del proyecto

Con las aplicaciones y herramientas
instaladas, realiza lo siguiente 🖱️

1

Crea una carpeta en
algún lugar tu
computadora

2

Abrela con VS Code mediante
el menú **File / Open Folder** o
abriendo VS Code primero, y
arrastrando la misma hasta el
IDE

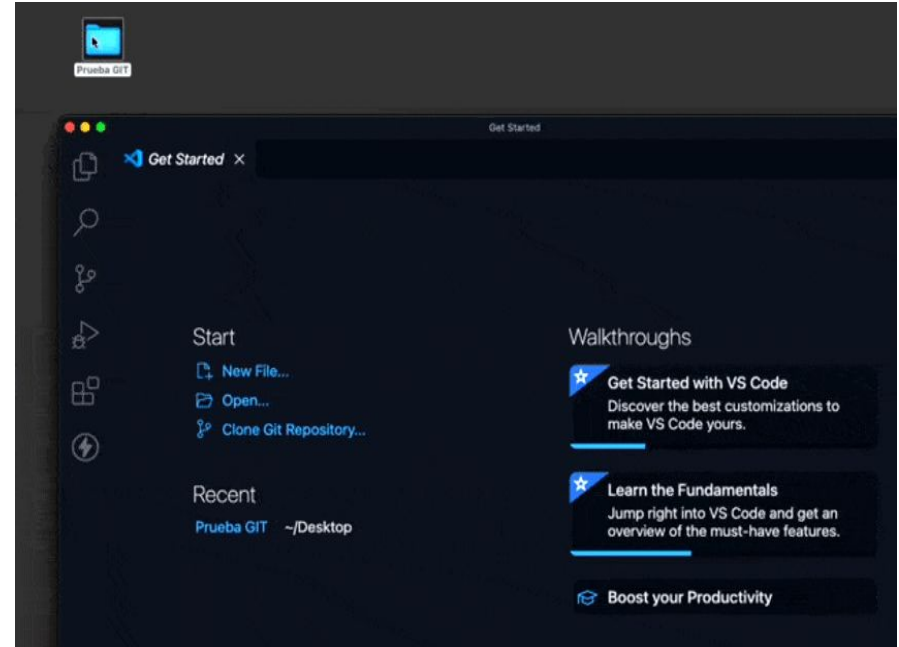
3

Ubica la barra lateral de
acciones de VS Code y
pulsas el ícono **Git Source
Control**



Inicializar el repositorio

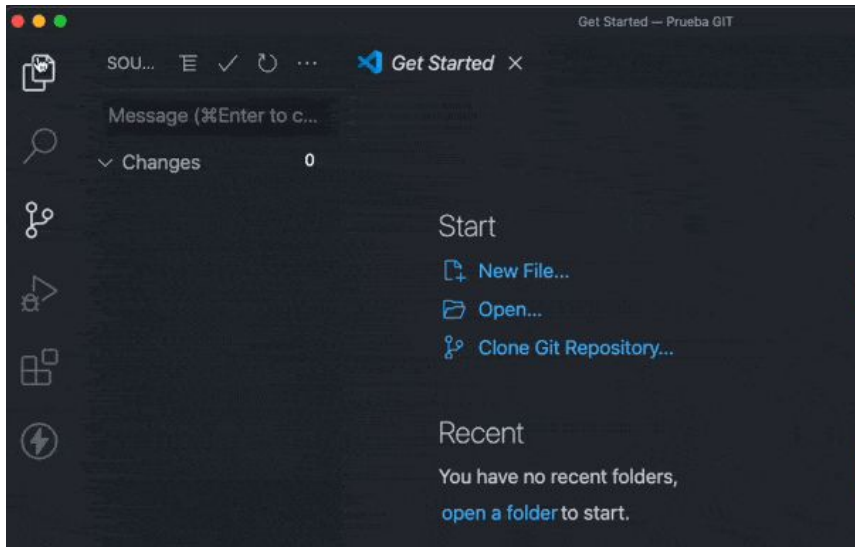
Posicionate en el botón **Source Control**, y haz clic en **Iniciar repositorio / Initialize Repository**, así tendrás activas las opciones de Git dentro de tu proyecto.





Crear un nuevo archivo

Crea, a continuación, un archivo dentro del proyecto para que éste tenga un contenido válido.



Paso a paso:

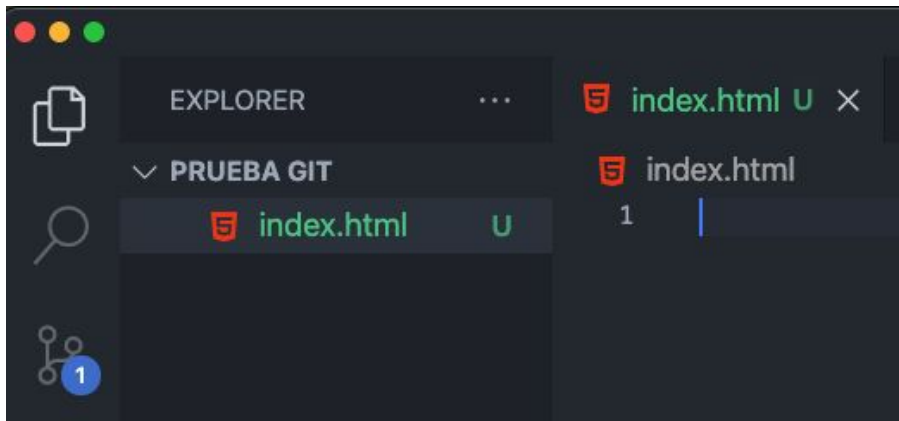
- En VS Code, dirígete al botón **Explorer**.
- Pulsa el botón **New File**.
- **Ingresa un nombre** para el archivo.
- **Pulsa Enter** para confirmarlo.

Indicadores de GIT

Ejemplo
en vivo



Veamos los indicadores de GIT dentro de nuestro proyecto:



1. Junto al nombre del archivo aparece la letra **U** , en color verde
2. Junto al botón **Source Control** figura un badge azul con el nro. 1

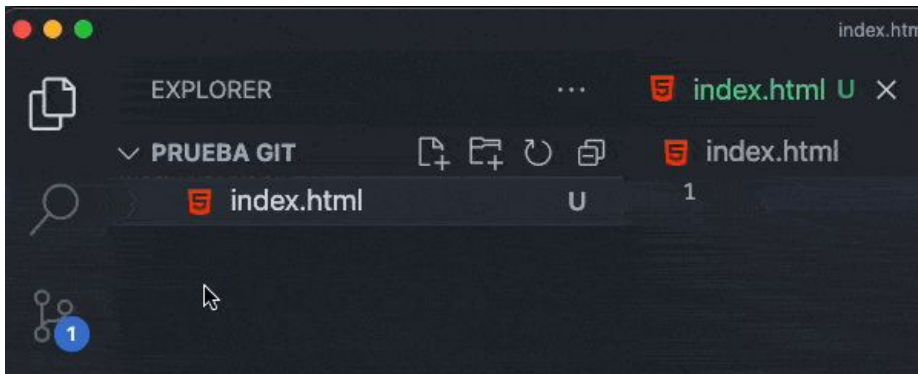
Estos parámetros definen que: existe un archivo en estado **'Untracked' (U)**, y que hay un cambio pendiente (**1**).

REALIZAR UN COMMIT



Staging Area

Previo a realizar **commit**, debes poner el archivo en estado **Tracked**, o sea, bajo el **rastreo de Git**.

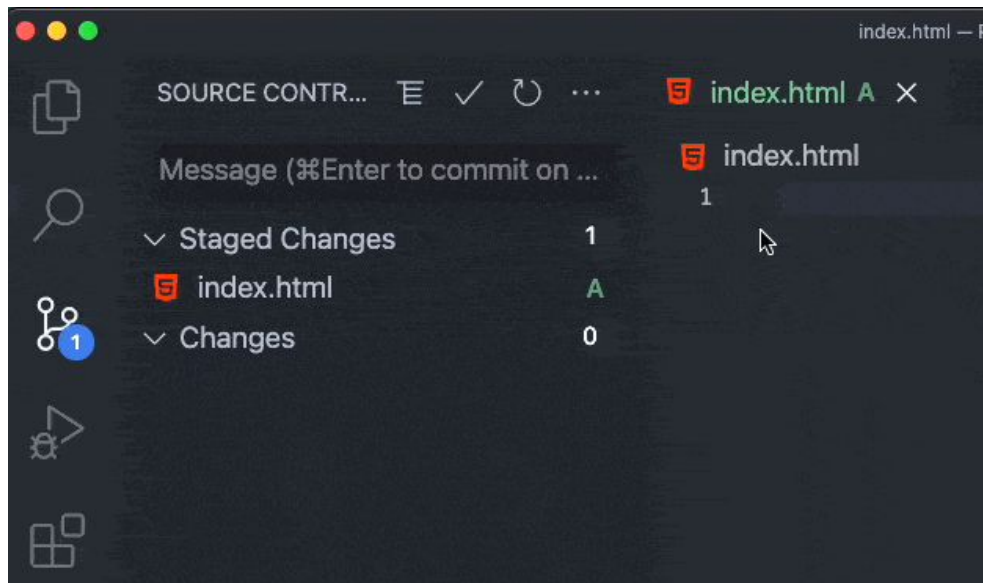


- Pulsa el botón **Source Control**
- Haz clic en el ícono **+** junto al nombre
- Con esto, has añadido el archivo al **área de staging**.


El archivo pasa al apartado **Staged Changes**, así ya **puedes realizar commit**.

Realizar un commit

Ejemplo
en vivo



Para realizar **commit**, debes ubicarte primero en la opción **Source Control**.

Luego pulsa el botón  en la barra de herramientas superior e ingresa una descripción para dicho commit seguido de la tecla **Enter**.

Realizar un commit

Ejemplo
en vivo

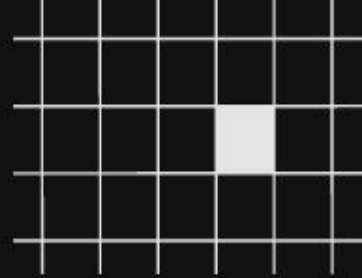
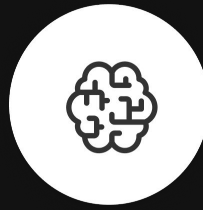


```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1">
7   <title>Document</title>
8 </head>
9 <body>
10
11 </body>
12 </html>
```

Realizado el commit, la interfaz de Visual Studio Code vuelve a mostrarse tal como estaba antes de añadir el archivo, lo cual indica que no hay cambios pendientes.



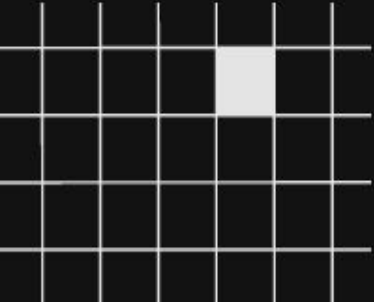
CREAR NUEVAS RAMAS



¡PARA PENSAR!

¿Qué diferencias reconoces con respecto a otros framework conocidos?

CONTESTA EN EL CHAT DE ZOOM



CREAR NUEVAS RAMAS

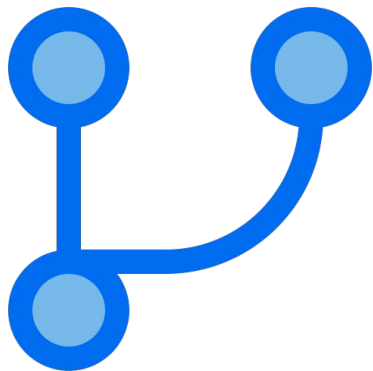
Git tiene como rama principal de cada proyecto, a la Rama **Master**, la cual se crea con la primera confirmación de cambios que realizamos de ese proyecto.

Y **también nos permite ir creando diferentes ramas** del mismo proyecto.

Veamos a continuación cómo hacerlo.



Crear una nueva rama

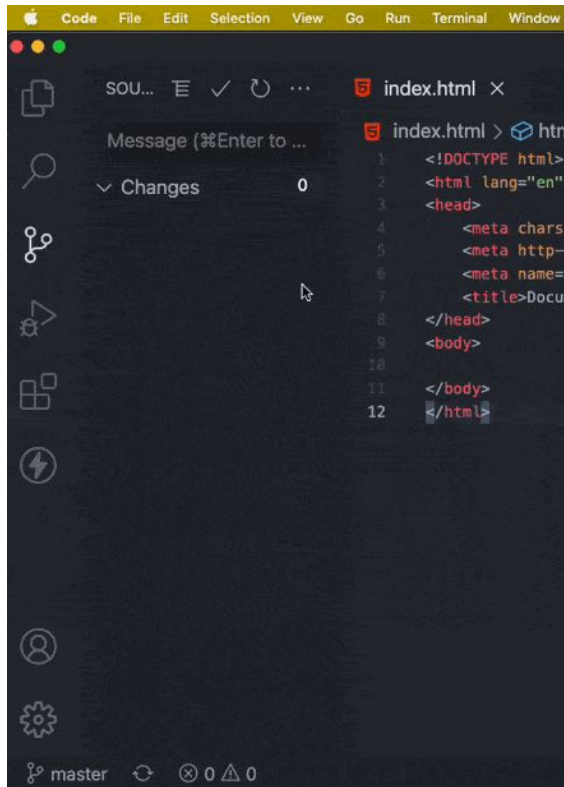


Para crear una nueva rama, utilizamos en Visual Studio Code la **Paleta de Comandos**, o **Command Palette**, una de las funcionalidades más importantes de Visual Studio Code.

Veamos cómo hacerlo 👁️👁️



Crear una nueva rama



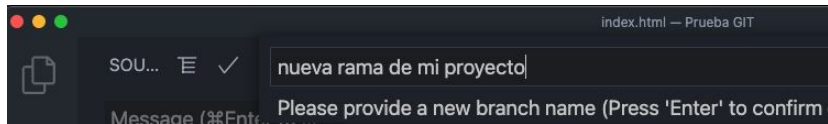
Abrimos la Paleta de Comandos mediante alguna de estas opciones: 

1. **View > Command Palette** (Ver > Paleta de comandos)
2. O pulsando las teclas **Ctrl+Shift+P**

Luego, escribimos `git branch`, para filtrar y visualizar todas las opciones de las ramas git disponibles.

Crear una nueva rama

Ejemplo
en vivo

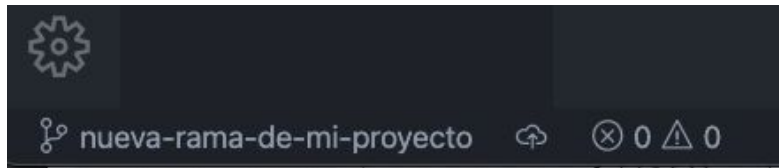


Selecciona **Git Create Branch**.

A continuación, escribe en la caja de texto el nombre de la nueva rama y, finalmente, pulsa **Enter**.

La nueva rama es creada con la descripción ingresada. Puedes visualizarla al pie del IDE.

A partir de ahora, todo lo que hagas se registrará en dicha rama del Proyecto.





REALIZAR UN MERGE

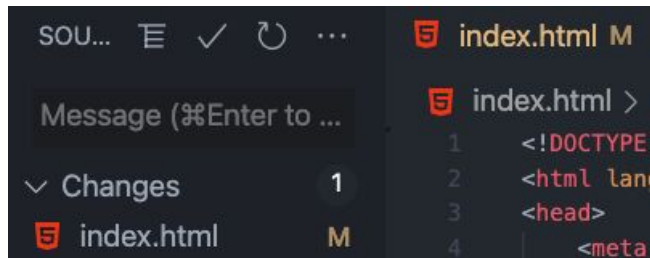


Realizar un Merge

Realicemos a continuación un Merge, que vaya desde nuestra nueva rama hacia la rama Master. Para ello, realiza un cambio mínimo en el archivo.



```
<body>
  <h1>Hola Coders!</h1>
  <p>Escribo esto para generar cambios en la nueva rama.</p>
</body>
</html>
```

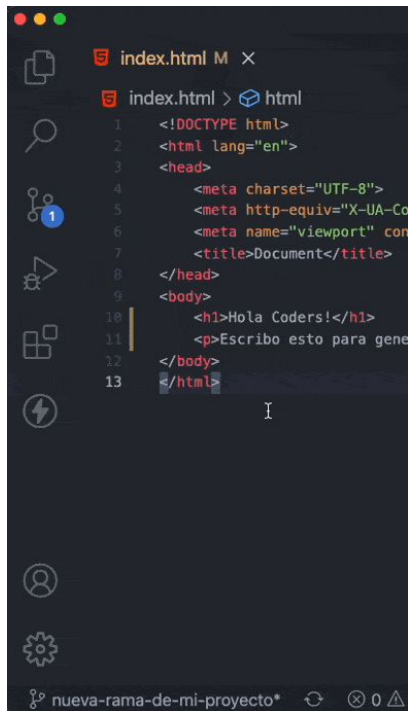


Verás que, junto al nombre del archivo, aparece una letra **M**, la cual indica que hubo modificaciones sobre el mismo.

Esto ya nos habilita a realizar el Merge.



Seleccionar rama Master



```
index.html M X
index.html > html
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Co
6   <meta name="viewport" con
7   <title>Document</title>
8 </head>
9 <body>
10   <h1>Hola Coders!</h1>
11   <p>Escribo esto para gene
12 </body>
13 </html>
```

👉 Vamos a posicionarnos en la rama Master:

- Haz clic sobre el nombre de la rama actual, en la **barra de tareas del IDE**.
- Se desplegará la Paleta de Comandos donde visualizas el **listado de ramas disponibles**.
- Allí, selecciona nuevamente la rama Master.



El comando git merge

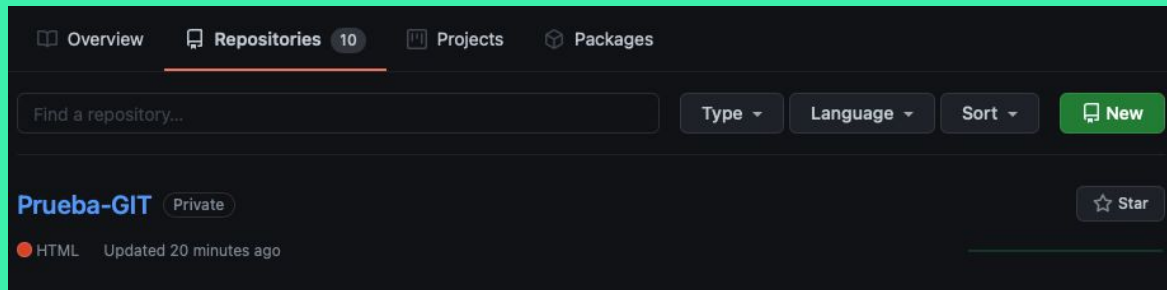
```
✕  
html  
html>  
g="en">  
  
charset="UTF-8">  
http-equiv="X-UA-Compatible" content="IE=edge">  
name="viewport" content="width=device-width, initial-scale=1.0">
```

1. Despliega nuevamente la **Paleta de Comandos**.
2. Escribe, a continuación, `git merge`.
3. Selecciona de la lista, la opción `git merge branch...`
4. Finalmente selecciona la rama que creamos como origen de este Merge.



GIT PUSH

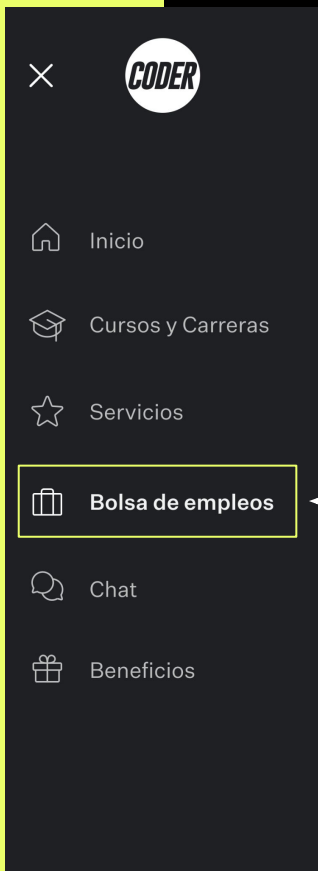
Cuando hagas **Push**, podrás visualizar tu proyecto publicado en github.com.





BREAK

¡5/10 MINUTOS Y VOLVEMOS!



Nuevo

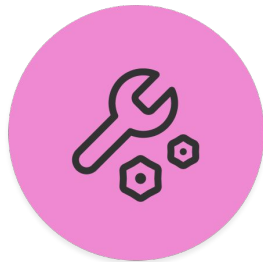
¡Lanzamos la Bolsa de Empleos!

Un espacio para seguir **potenciando tu carrera** y que tengas más **oportunidades de inserción laboral**.

Podrás encontrar la **Bolsa de Empleos** en el menú izquierdo de la plataforma.

Te invitamos a conocerla y ¡postularte a tu futuro trabajo!

Conócela



REPOSITORIO CON GIT

Crear un repositorio local de Git

Tiempo estimado: 10 minutos

CREA UN REPOSITORIO CON GIT

Desafío
generico



Luego de incorporar el proceso de uso de git a través de Visual Studio Code, llega tu turno de poner a trabajar tu espacio.

- Crea un repositorio local de GIT y luego sincronízalo con Github.
- Incluye en tu proyecto 2 o más archivos, y ten presente agregarlos al área de Stage.
- Has uso de Visual Studio Code para realizar los procesos de **add**, **commit** y **push** hacia el repositorio remoto.

Cuentas con **10 minutos** para realizar el ejercicio.

UN REPASO POR LAS NOVEDADES DE ES6 Y SUPERIORES



ECMAScript



ES6

ECMAScript es el estándar que regula la estructura y sintaxis del lenguaje JavaScript. Desde su versión 6, lanzada en 2015, JS comenzó a incorporar una serie de novedades importantes que potenciaron fuertemente este lenguaje.

Hagamos, a continuación, un repaso por las más importantes.



ARROW FUNCTIONS

Arrow functions, o funciones de flecha, llegan al lenguaje para simplificar la creación de funciones convencionales. Las mismas se construyen a partir de una constante, dejando de lado el uso de las palabras reservadas **function** y **return**.

Resumen su estructura de una forma notable pudiendo, en algunos casos, simplificar la misma a una única línea de código.

```
const miFuncion = ()=> {  
  console.log('arrow function  
convencional')  
}
```

```
const miFuncionParams = (par1, par2)=> {  
  console.log('Función con parámetros')  
}
```

```
const miFuncionRet = ()=>  
console.log('Función con retorno')
```



CLASS

Las clases en EcmaScript son similares al patrón orientado a objetos basado en prototipos. Son más fáciles de usar porque **su forma declarativa y única permite fomentar la interoperabilidad.**

```
class Persona {  
  constructor(nom, ape) {  
    this.nombre = nom  
    this.apellido = ape  
  }  
  nombreCompleto = () => `${this.nombre} ${this.apellido}`  
}
```

```
//instancia y uso  
const salesMan = new Persona('Joe', 'MacMillan')  
salesMan.nombreCompleto()
```




PROMISES

Una promesa representa los procesos que ya están sucediendo. Puede estar en uno de los siguientes pasos: **pendiente**, **resuelta**, **rechazada**.

Cuando comienza la ejecución de la función, asume el estado Pendiente. Al completarse la ejecución del código, la promesa se Resuelve o se Rechaza.

```
const prom2 = Promise.resolve(7)
prom2
  .then(y => y * 3)
  .then(y => y * 10)
  .then(x => Promise.reject('Error: se rompió nuestra aplicación :('))
  .then(y => console.warn("Esto no se debería ver."))
  .catch(e => console.error(`Aquí llegamos porque, apareció un método reject()`))
```



IMPORT MODULE

Otra fabulosa característica de ES es la posibilidad de **importar módulos**.

```
import * as myModule from '/modules/my-module.js'
```

Inicialmente pensada para cargar módulos JS completos o parciales, en sus últimas versiones podemos también importar HTML y CSS, además de archivos JS.

```
import '/myStylesheets/file.css!'
```



TEMPLATE STRINGS

Nos permiten construir cadenas sintácticamente, combinando HTML, comillas simples y dobles, además de integrar variables o constantes para obtener su valor. Template Strings puede funcionar combinándose con **backticks** y **Template Literals**.

```
const miVariable = 'Gordon Clark'

const plantillaHTML = `

Mi nombre es: ${miVariable}</p>`

divPresentacion.innerHTML = plantillaHTML


```



DESTRUCTURING

Se trata de una expresión que permite utilizar valores de matrices o propiedades de objetos en distintas variables. A su vez, se puede combinar con los operadores **Rest** o **Spread**, para actualizar una matriz de objetos usando la propiedad de desestructuración.

```
const company = {name: "Cardiff Electric",  
                  location: "Silicon Praire, TX",  
                  founded: 1983,  
                  trade: "Computers"}  
const {name, trade} = company
```



USO DE *let* Y *const*

```
const nombre = 'Cameron'
```

```
let apellido = 'Howe'
```

```
function consultarNombre() {
```

```
  let nombre = 'Donna'
```

```
  let apellido = 'Clark'
```

```
    console.log(`${nombre} ${apellido}`)
```

```
}
```

La palabra reservada **let** es similar a **var**, excepto que **let** tiene solo un alcance de bloque, mientras que **var** consigue un alcance global.

Las declaraciones utilizando **const** generan una sola asignación y su alcance es de bloque, por lo tanto **let** y **const** no se incluyen en el ámbito global.



OPERADOR DE EXPONENCIACIÓN

```
const resultado = (9 ** 2)
    console.log(resultado)
//devuelve 81
```

Además de admitir las operaciones aritméticas básicas, **ES7** introdujo el operador de exponenciación, ******. Este hace el mismo trabajo que la función **Math.pow()**, devolviendo el valor del primer argumento elevado a la potencia del segundo argumento.



Array.includes()

Este método comprueba el valor pasado como argumento. Si la matriz analizada contiene el valor, devolverá como resultado **true**; de lo contrario, devolverá **false**.

Es ideal para buscar datos en un array de objetos JSON. 🙌

```
const paises = ['Argentina', 'Argelia', 'Andorra', 'Angola', 'Antigua y Barbuda', 'Arabia Saudita']  
const res = paises.includes('Bélgica')  
//devuelve false
```



OBJECT RESTING

Nos permite destruir un objeto, recopilando sus elementos restantes en un nuevo objeto.

```
const noPassword = ({ passwd, ...rest }) => rest
const empleado = {
  id: 5,
  nombre: 'John Bosworth',
  passwd: 'IhaveNoPassw0r!')
}

noPassword(empleado) //resultado { id: 5, nombre: 'John Bosworth' }
```




array.flat()

Aplana de forma recursiva una matriz hasta el nivel que se le especifica.

Puede aplanar la matriz tantas veces como se le mencione.

```
let arrayNumerico = [1, 2, [3, 4, [5, 6]]]  
arrayNumerico.flat(2)
```

```
// El resultado >>> [1, 2, 3, 4, 5, 6]
```



array.flatMap()

Permite mapear una matriz de elementos, y luego aplanarla de una sola vez consiguiendo así una matriz unidimensional aplanada como resultado de su uso.

```
let arrayNumerico = [1, 2, 3, 4]
arrayNumerico.flatMap(x => [x * 2])

// El resultado >>> [2, 4, 6, 8]
```



trimStart() y trimEnd()

Recorta los espacios en blanco al inicio; `trimStart()`, o final; `trimEnd()`, de una cadena de caracteres. Llega como una actualización/reemplazo futuro de `trimLeft()` y `trimRight()`.

```
let nombreCompleto = "  HALT AND CATCH FIRE      "  
  
nombreCompleto.trimStart() //Devuelve "HALT AND CATCH FIRE      "  
nombreCompleto.trimEnd()   //Devuelve "  HALT AND CATCH FIRE"
```



EL OBJETO GlobalThis

```
globalThis.setTimeout  
globalThis.XMLHttpRequest  
globalThis.location.href    //pertenece a window  
globalThis.caches           //pertenece al objeto self
```

Para unificar el uso de un objeto global en todas las plataformas que ejecutan JS, llega **globalThis**. Este nos permite instanciar un objeto puntual, anteponiendo globalThis a dicho objeto, lo cual nos permite evitar tener que saber si este se encuentra en el objeto **self**, **window** o **frames**.

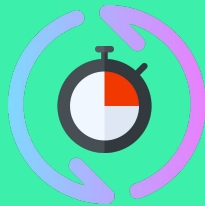


string.replaceAll()

```
const res = cadena.replaceAll('ipsum')
```

Este método llega para reemplazar todas las apariciones indicadas dentro de una cadena. Reemplaza al método `replace()`, el cual se utilizaba anteriormente iterando a lo largo del string, dado que solo reemplazaba el primer valor que aparecía.

ITERACIÓN ASINCRÓNICA



ES 2018 introdujo iteradores e iterables asincrónicos, los cuales, a través de una nueva **Iteration Statement** conocida como `for - await - of`, agregan sintaxis para crear funciones de generador asíncrono.



OPERADOR COALESCENTE NULO

```
const name = '' || 'Joe MacMillan'  
// Resultado: Joe MacMillan  
const name = '' ?? 'Joe MacMillan'  
// Resultado: ''
```

El **operador coalescente nulo** funciona de forma similar a `&&` o `||` devolviendo el operando derecho cuando el operando izquierdo es nulo o indefinido. De lo contrario, devuelve su operador izquierdo.



OPERADOR DE ENCADENAMIENTO OPCIONAL

```
let arrayItem = arr?.[42]

let prop = potentiallyNullObj?.[x++]

let duration = vacations.trip?.getTime?.()
```

Este operador permite leer el valor de una propiedad dentro de una cadena de objetos conectados, sin la necesidad de validar que cada referencia en la cadena sea válida.



SEPARADOR NUMÉRICO _

```
const valorNro = 2_103_197_521_172
```

El caracter underscore, o guión bajo, se integró a los valores numéricos en JS para poder leer de forma más cómoda números con muchas cifras. Este caracter simula ser el separador de miles, sin alterar los valores numéricos almacenados en variables.



Promise.finally()

Este método permite realizar dentro de una Promesa, la ejecución de un bloque de código determinado, más allá del resultado de dicha promesa (`resolve` o `reject`).

Se comporta de forma similar al bloque `finally` incluido en `try-catch`.

```
const prom2 = Promise.resolve(21)
prom2
  .then(y => y * 3)
  ...
  .catch(e => console.error(`Aquí llegamos porque apareció un método reject()`))
  .finally(e => console.info(`Este mensaje se verá más allá de cómo termina la Promesa.`))
```

Promise.any() y Promise.all()

Ejemplo
en vivo



```
var p1 = Promise.resolve(3)
var p2 = 1337
var p3 = new Promise((resolve, reject) => {
    setTimeout(resolve, 100, "foo");
})

Promise.all([p1, p2, p3]).then(values => {
    console.log(values); // [3, 1337, "foo"]
})

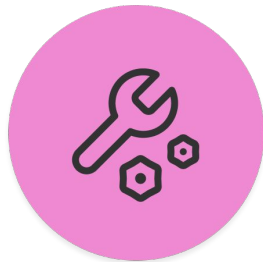
Promise.any([p1, p2, p3])
    .then(response => console.log(response))
    .catch(error => console.log(error));
```

El método `promise.all()` es ejecutado luego de que son resueltas todas las promesas mencionadas al definirlo, mientras que `promise.any()` es ejecutado y se resuelve cuando alguna de las promesas que recibió como parámetro haya sido resuelta.

PROMISES Y GENERATORS

Ambas características se pueden combinar para realizar operaciones asíncronas con la ayuda de un código de aspecto sincrónico. Internamente, las funciones asíncronas funcionan de manera similar a los generadores, sin embargo, no se traducen a funciones de éste último.





ECMAScript

Realizar un programa en ECMAScript

Tiempo estimado: 10 minutos

ECMAScript

Tiempo estimado: 10 minutos

Desafío
generico



Te proponemos realizar un algoritmo en ECMAScript, utilizando una o más características de E.S. de las que estuvimos repasando recién.

Si utilizas la extensión Live Server, diseña este algoritmo pensando en que pueda ser ejecutado en la Consola JS. Y si eres del Team Node JS, piénsalo para que sea ejecutado en la Terminal o Línea de Comandos.

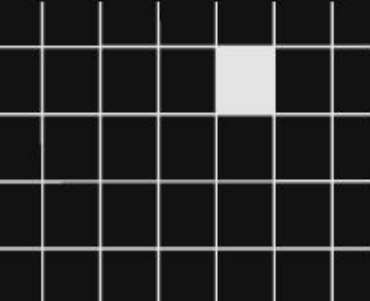
¿PREGUNTAS?





¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- Vue: Historia, beneficios y estadísticas
 - Framework: común versus progresivo
 - Uso de Git desde Visual Studio Code
 - Novedades de ECMAScript
- 



OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDO LA EDUCACIÓN

CODER HOUSE