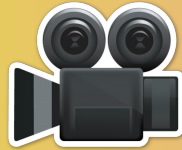




Clase 9. Vue JS

Router y Life Cycle Hooks

RECORDÁ PONER A GRABAR LA CLASE





OBJETIVOS DE LA CLASE

- Configurar el módulo de router en VueJS para trabajar manejado desde HTML ó desde el código.
- Comprender las funciones de ciclo de vida de los componentes Vue y su uso en distintos escenarios.

CRONOGRAMA DEL CURSO

Clase 8



Formularios



FORMULARIO CON VUE



APLICANDO VUE-FORM



FORM CON VUE CLI Y SUS VALIDACIONES

Clase 9



Router y Life Cycle Hooks



SINGLE PAGE APPLICATION



EJEMPLO EN VIVO



LIFECYCLE HOOKS

Clase 10



API Rest



PETICIÓN REMOTA VÍA
FETCH



PETICIONES REMOTAS VÍA
AXIOS API



SEGUNDA ENTREGA
PROYECTO FINAL

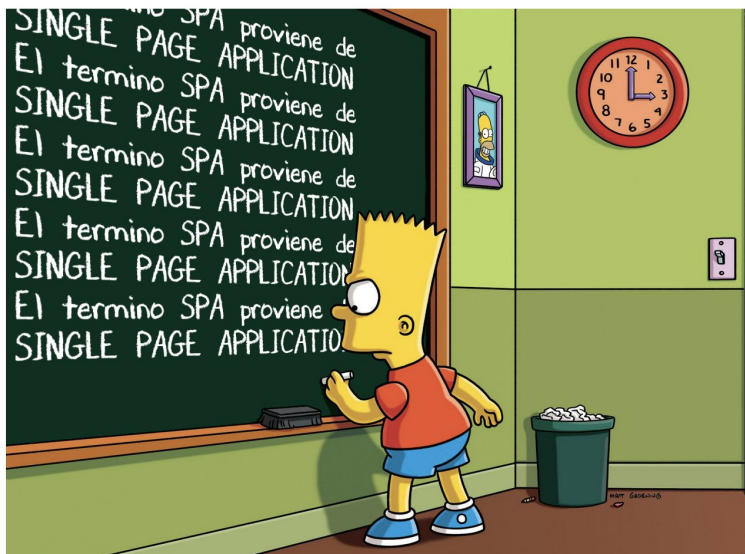
ROUTER EN VUE JS

ROUTING Y SPA

Desde hace más de una década el desarrollo de aplicaciones web incluyó, casi de forma mandatoria, la creación de aplicaciones SPA, o aplicaciones de una sola página. Y el ruteo -o routing- es clave para el correcto desarrollo de este paradigma.

Veamos a continuación cómo Vue maneja estos paradigmas.

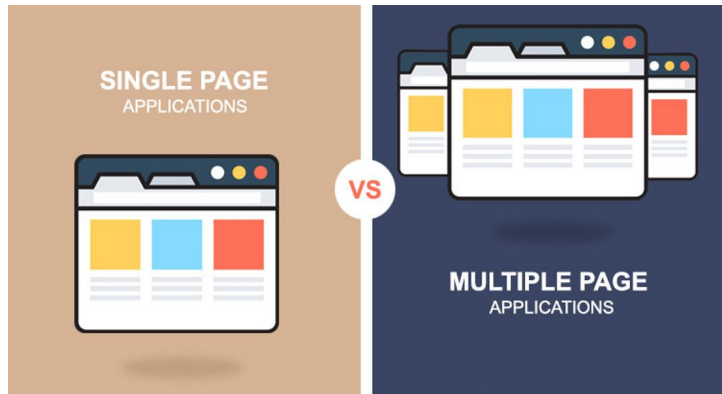
SPA



Tal como nos anticipa Bart, el término SPA proviene de **Single Page Application**.

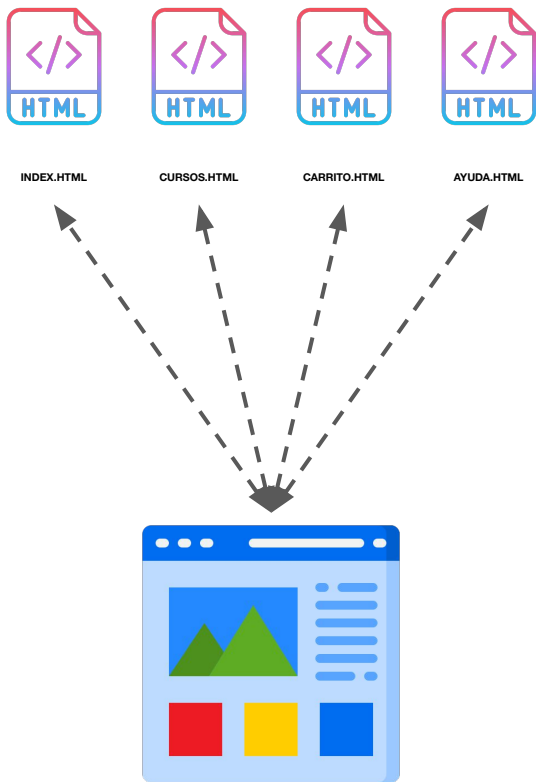
Esta tecnología, creada en 2003, popularizada en 2005 y de adoptada masivamente hacia el año 2010, cambió la forma de traer contenido desde un sitio web hacia la computadora o dispositivo de los usuarios.

SPA



La forma de transmisión de páginas desde un servidor web al usuario se hacía bajo el paradigma **Multiple Page Applications**, lo que implica que cada sección de un sitio web está construida sobre un documento HTML individual.

SERVIDOR WEB: www.elsitioquenavego.com

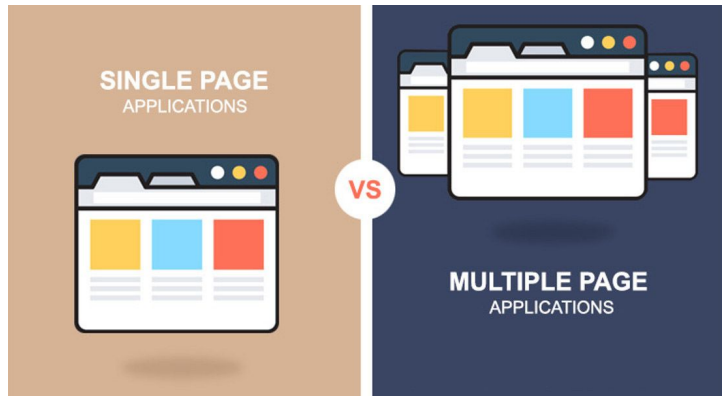


NAVEGADOR WEB

SPA

De esta forma, por cada una de las secciones que recorre el usuario de un sitio web, hace una nueva petición al servidor, trayendo un nuevo documento HTML, con sus encabezados estilos, lógica JS y contenido multimedia.

El paradigma MPA genera así lentitud en la respuesta del contenido que el usuario desea ver.



SPA

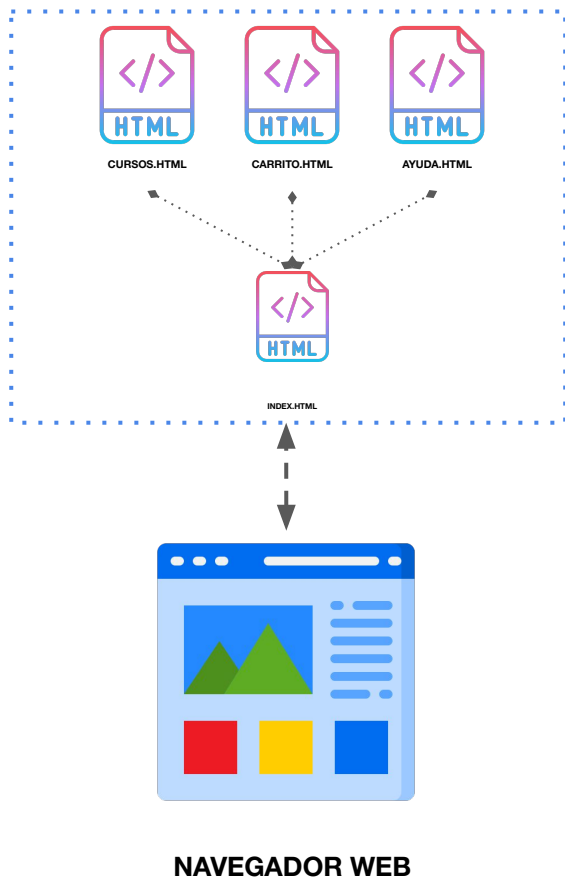
Bajo la propuesta del paradigma SPA el sitio web se diseña en una sola página y cada sección (o `<body>`) de las otras páginas web que lo conforman está contenido (de forma oculta) en la página principal.

Cuando el usuario peticiona otra sección del sitio web, se escribe en el documento HTML el bloque de código HTML de esa sección



TIEMPOS DE ADOPCIÓN

En términos de adopción de esta tecnología, se tuvo que conjugar algunas correcciones sobre la forma de mostrar el contenido dado que se debía contar también con la generación del historial de navegación, además de poder disponer de un ancho de banda generoso que permitiese bajar casi toda la aplicación o sitio web en la primera petición, en un tiempo prudencial que no ahuyente al usuario por la posible espera generada.



SPA

Cuando el usuario peticiona al servidor, la página principal trae consigo el HTML de cada una de las secciones que componen este sitio o aplicación web.

Su carga en el navegador web será algo más lenta la primera vez (porque baja más contenido). Pero luego, al navegar por sus secciones, la información de cada una de ellas aparece instantáneamente.

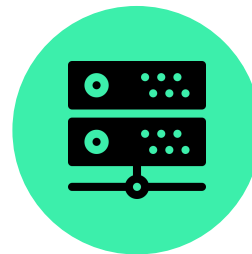
BENEFICIOS DE CREAR UNA SPA



La posible lentitud de carga inicial del sitio web se verá altamente compensada con la rapidez general de su navegación.



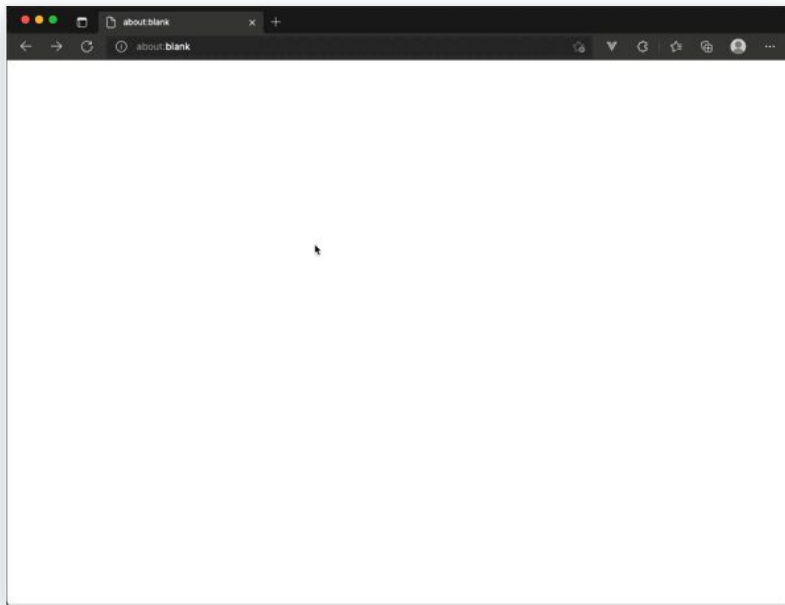
Nos ayuda a diseñar muy bien su lógica y estilo ya que toda esta información viajará también en la carga inicial del sitio.



Disminuye de forma considerable las peticiones de datos al servidor ya que la lógica se ejecutará del lado del cliente.



EJEMPLO DE UNA SPA

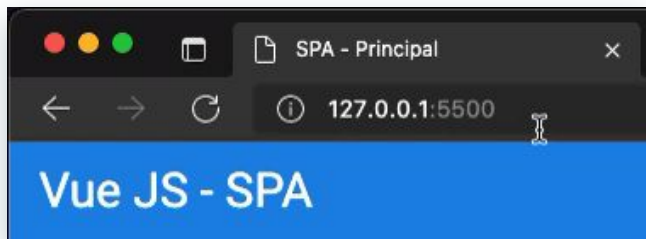


Este es un ejemplo real de cuán rápido responde una SPA. Su ventaja es notoria ante cualquier otra aplicación de tipo MPA.

Incluso, podemos pensar un web component dedicado a controlar aquellas URL que puedan ser ingresadas a mano por el usuario, y que no existan como ruta.



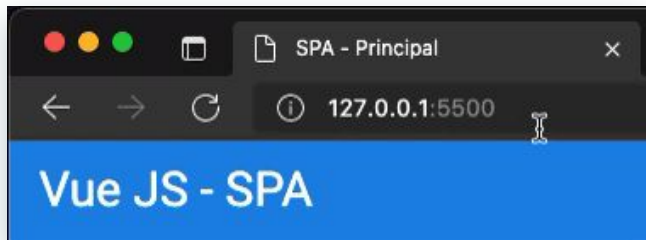
RUTEO o ROUTING



El Ruteo, o Routing, establece un *path* que se visualiza en la barra de URL para que el usuario interprete que está navegando por diferentes secciones de un sitio o aplicación web.



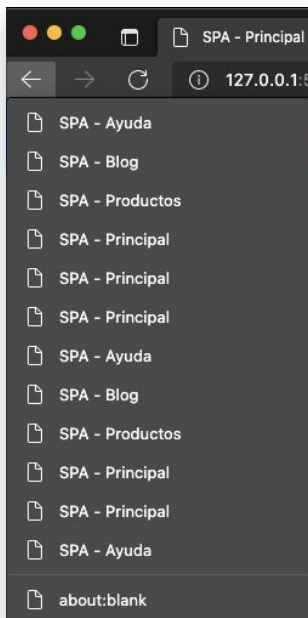
RUTEO o ROUTING





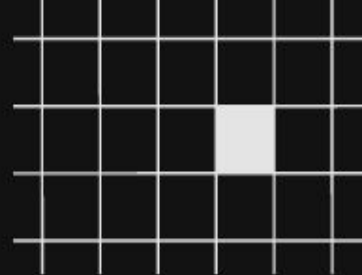
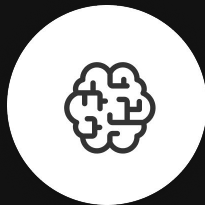
Para nosotros, el ruteo nos indica qué porción de código/web component debemos mostrar en pantalla de acuerdo a la petición realizada por el usuario. En algunos casos, la ruta puede contener un carácter del tipo hash.



RUTEO o ROUTING



Además, permite generar un historial de navegación por las distintas secciones para que, aquellos usuarios que recurran a la navegación mediante las teclas  /  o su combinación por teclado, puedan desplazarse sin problemas a través de todas las secciones que contiene una SPA.

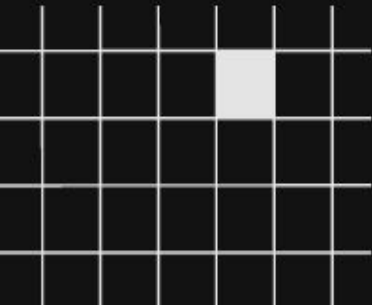


¡PARA PENSAR!

¿Desarrollaste alguna vez una aplicación del tipo SPA con Vanilla JavaScript u otra librería o framework JS?

¿Cómo te fue con la experiencia?

CONTESTA A TRAVÉS DEL CHAT DE ZOOM



VUE-ROUTER

VUE-ROUTER



Vue Router es el complemento oficial de Vue para poder establecer los mecanismos de ruteo de una SPA, en un desarrollo basado en Vue/Cli.

Si bien existen otras librerías para establecer ruteo en nuestras aplicaciones Vue, utilizaremos la oficial para poder sacar el máximo provecho y compatibilidad.

VUE-ROUTER



Durante la instalación de Vue/Cli, contamos con la opción dentro del proceso que nos permite seleccionar si deseamos instalar, o no, Vue Router.

Para validar la existencia del mismo, podemos abrir cualquier proyecto Vue/Cli anterior, y verificar en el archivo `package.json`, apartado “`dependencias`” si existe o no la referencia a Vue Router.

INSTALAR VUE-ROUTER VÍA NPM



INSTALAR VUE-ROUTER VÍA NPM

→ Abre el archivo `package.json`, y utiliza el buscador de tu IDE para saber si `vue-router` está o no disponible para utilizarse.

```
...  
"vue-router": "^3.0.3",  
...
```

Puede que tengas instalada la referencia con un número de versión diferente al mostrado aquí. Si no lo encuentras, lo instalaremos a continuación.



INSTALAR VUE-ROUTER VÍA NPM

→ Si, por ejemplo, tu proyecto Vue/Cli se llama `spa-vue`, ingresa a su carpeta o ábrelo con Visual Studio Code y luego **ejecuta en la Terminal:**

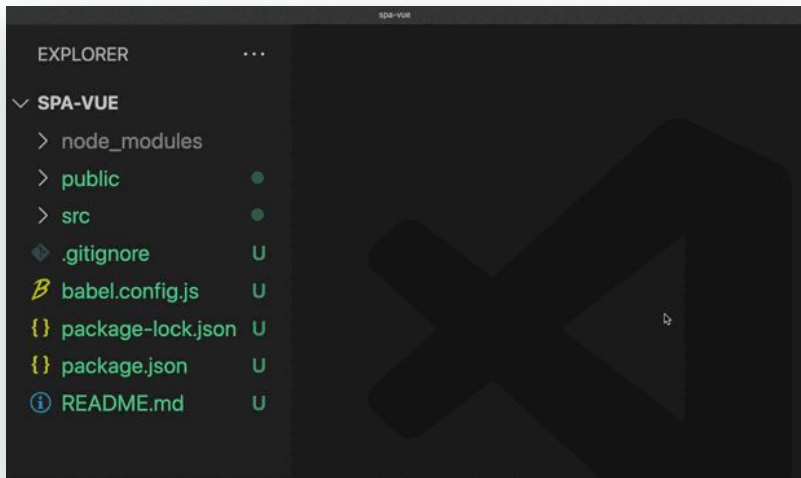
```
_> npm install vue-router --save
```

De esta forma te aseguras que la referencia a `vue-router` se guarde en el archivo `package.json` de tu proyecto.





INSTALAR VUE-ROUTER VÍA NPM



→ Finalizado el proceso de instalación, abre el archivo `package.json` para corroborar que `vue-router` fue agregado exitosamente en tu proyecto. 😎



VUE ROUTER

Vue Router es el complemento ideal para facilitar el proceso de creación de rutas y así lograr desarrollar, en base a nuestros proyectos Vue/Cli, un sistema de ruteo fácil para aplicaciones de una sola página, partiendo de los web components que conformen nuestra aplicación web.

¿CÓMO FUNCIONA VUE-ROUTER?



CÓMO FUNCIONA VUE ROUTER

```
import Vue from 'vue'
import VueRouter from 'vue-router'

Vue.use(VueRouter)
```

→ El primer paso para integrar `vue-router` es importar la dependencia dentro de nuestro proyecto Vue/Cli.

Dentro de un archivo llamado `router.js` en el mismo nivel que `main.js` se debe agregar la importación de `vue-router`, `vue`, y el método `Vue.use()` referenciando a `VueRouter`.

CÓMO FUNCIONA VUE ROUTER

→ Con `vue-router` agregado e importado dentro de nuestro proyecto, el paso siguiente es definir los componentes web de nuestra aplicación.

→ Luego, con los componentes web definidos, creamos en el archivo `router.js` donde venimos trabajando un objeto JSON que dispondrá de las rutas y el componente web que debe mostrar.



CÓMO FUNCIONA VUE ROUTER

```
import Vue from 'vue'
import VueRouter from 'vue-router'
import Home from './views/Home.vue'
import Contacto from './views/Contacto.vue'

Vue.use(VueRouter)
```

→ Debajo de la declaración de **VueRouter** iremos agregando cada uno de los archivos que compondrá una vista de nuestra aplicación SPA.

Estos archivos, debemos tenerlos previamente creados para que nuestra App no arroje error.



CÓMO FUNCIONA VUE ROUTER

```
...  
Vue.use(VueRouter)  
  
export default new router({  
  mode: 'history',  
  base: process.env.BASE_URL,  
  routes: [...]  
})
```

→ A continuación, debemos declarar la sentencia **export** para exportar un módulo con una instancia del objeto **Router**, la cual contendrá un array de objetos con los lineamientos que le darán vida a las rutas dentro de nuestra aplicación Vue.



CÓMO FUNCIONA VUE ROUTER

```
...  
Vue.use(VueRouter)  
  
export default new router({  
  mode: 'history',  
  base: process.env.BASE_URL,  
  routes: [...]  
})
```

→ Al instanciar a **router**, le indicamos mediante los parámetros del objeto, la forma en la cual debe actuar el ruteo.

La propiedad **mode**, indica que el ruteo debe generar historial de navegación. La propiedad **base** indica que la base de la navegación será la URL base del entorno donde la aplicación se encuentre y, **routes**, el array que contendrá las vistas de ruteo que usaremos.

LA CARPETA VIEWS





VIEWS

```
import Home from './views/Home.vue'  
import Contacto from './views/Contacto.vue'
```

Cada Vista de nuestro proyecto será nada más y nada menos que un archivo `.vue` con la estructura específica del apartado `<template>` que conocimos en los archivos de componentes vue. Y, para un mayor orden de las Vistas, se recomienda organizarlas dentro de una carpeta llamada `/views/` en el raíz de nuestro proyecto.

Al momento de importarlas, debemos referenciar el path referente a dicha carpeta.



VIEWS

```
<template>
```

```
<div>
```

```
<h1>Bienvenidos</h1>
```

```
<p>
```

```
  Aquí detallamos el contenido de bienvenida dedicado al  
  usuario que accede a este sitio web o aplicación.
```

```
</p>
```

```
</div>
```

```
</template>
```

👉 Esta es la estructura de ejemplo de la Vista del componente `Home.vue`.

Como podemos apreciar, solo está conformada por el apartado

`<template>`, similar al utilizado en cada Componente Vue.



VIEWS

Si bien la estructura de las Vistas es prácticamente similar a la de cada Componente Vue, tengamos presente que los componentes pueden conformar una pequeña sección de lo que se verá en un documento HTML y, una Vista, contendrá uno o más componentes web que terminarán conformando el documento contra el cual el usuario interactuará.

RELACIÓN RUTA - VISTAS



RELACIÓN VISTA - RUTAS

Volvamos al archivo `router.js` a trabajar con nuestro array `routes[]`. Dentro de este debemos establecer un elemento JSON por cada una de las Vistas que contendrá nuestra App Vue.

Cada elemento deberá tener una propiedad `path`, una propiedad `name` y la propiedad `component` donde detallamos: la ruta base, el nombre de dicha ruta y el componente asociado a la misma, el cual proviene de la vista relacionada.

```
routes: [  
  {  
    path: '/',  
    name: 'home',  
    component: 'Home'  
  },  
  {  
    path: '/contacto',  
    name: 'contacto',  
    component: 'Contacto'  
  },  
  ...  
]
```



RELACIÓN VISTA - RUTAS


```
...
routes: [
  {
    path: '/',
    name: 'home',
    component: () => import('./views/Ayuda.vue')
  }...
]
```

Otra alternativa para referenciar dentro del array `routes[]` a las Vistas de la aplicación es creando una función como ser `component:` `()=>` y dentro de ella importar la Vista, especificando su ruta completa.

Así obviamos el uso de `import` al inicio del archivo.



RELACIÓN VISTA - RUTAS

 127.0.0.1:5500/contacto



De esta forma, cuando se ejecute tu aplicación Vue/Cli, la propiedad **path** del objeto routes definirá la barra de separación del dominio de tu aplicación y la propiedad **name** será la cual refleje el nombre establecido para dicha ruta.

RELACIÓN VISTA - RUTAS



Ya tenemos a **Vue-Router** importado en nuestro proyecto Vue/Cli, las rutas de nuestra aplicación definidas, las Vistas creadas e importadas de forma directa o mediante una función dentro de **routes** y la instancia del objeto **Router** para que se establezca la generación del historial de navegación en el web browser.

CAMBIOS EN APP.VUE



<router-link>

```
<template>
  <div id="app">
    <div id="menu">
      <router-link to="/">Home</router-link>
      <router-link to="/contacto">Contacto</router-link>
      <router-link to="/ayuda">Ayuda</router-link>
    </div>
    ...
  </div>
</template>
```

Vue-Router cuenta con dos etiquetas o tags HTML que debemos agregar en el archivo **app.vue**. La primera de ellas es **<router-link>**, la cual posee un atributo denominado **to**, donde definimos la ruta base. Entre el tag de apertura y cierre de esta etiqueta, debemos agregar el nombre de la Vista a visualizar mediante dicho hipervínculo.



<router-view>

```
...  
</div>  
  
<HelloWorld msg="Bienvenido a tu SPA"/>  
  <router-view/>  
</div>  
</template>
```


Por otro lado, está la etiqueta o tag HTML `<router-view>`.

La misma se agrega simplemente en el apartado de `app.vue` para que, al hacer clic sobre un hipervínculo determinado, la Vista que representa a tal vínculo se cargue donde está definida esta etiqueta mencionada.



APP.VUE

```
<template>
  <div id="app">
    <div id="menu">
      <router-link to="/">Home</router-link>
      <router-link to="/contacto">Contacto</router-link>
      <router-link to="/ayuda">Ayuda</router-link>
    </div>
    
    <HelloWorld msg="Bienvenido a tu SPA"/>
    <router-view/>
  </div>
</template>
```

La Vista completa de `app.vue` deberá quedar establecida de forma similar al bloque de código  aquí representado.

IMPORTAR COMPONENTES A UNA VISTA



IMPORTAR COMPONENTES A UNA VISTA

```
//Ayuda.vue  
...  
</template>  
  
<script>  
import HelloWorld from  
  'src/components/HelloWorld.vue'  
...  

```

Si necesitas armar una Vista con uno o más componentes web ya desarrollados debes importarlos dentro de éstas, tal como importamos dentro de un archivo de componente vue u otros componentes.



IMPORTAR COMPONENTES A UNA VISTA

```
//Ayuda.vue
<template>
  <div>
    <h1>Ayuda</h1>
    <p>
      Bienvenidos a nuestra Ayuda en línea.
    </p>
    <hello-world msg="Importé un componente Vue a
una Vista">
  </div>
</template>
```

Una vez importado, ya puedes utilizarlo dentro del apartado Template de la Vista, configurándolo con sus atributos específicos tal como vimos en la clase de creación de Componentes Web con Vue/Cli.



INSTANCIAR ROUTER Y VUE

Como paso final, nos queda incluir en la instancia del objeto Vue la referencia de `router`, para que nuestra primera SPA funcione correctamente.

```
new Vue({  
  router,  
  render: h => h(App),  
}).$mount('#app')
```



SPA FUNCIONAL

Solo nos queda probar la SPA 100% funcional.
Si tienes conocimiento previo del tema, verás cómo
Vue resuelve de forma automática el historial de
navegación y también que no es necesario apoyarse
en construir una Single Page Application utilizando
hash en la barra de direcciones.

ROUTING DINÁMICO



ROUTING DINÁMICO

Para establecer un ruteo dinámico en una SPA, debemos definir en el path, la ruta base seguida de la barra, **dos puntos y un identificador**. Este oficiará como variable, tomando el valor que se pasa a la ruta definida a través de la última barra.

```
routes: [  
  ...  
  {  
    path: '/contacto',  
    name: 'contacto',  
    component: 'Contacto'  
  },  
  {  
    path: '/usuarios/:id',  
    name: 'usuarios',  
    component: 'Usuarios'  
  },  
]
```



ROUTING DINÁMICO

```
const usuario = {  
  template: '<div> Usuario: {{ this.$route.params.id }} </div>'  
}
```

El valor del segmento dinámico recibido a través de la URL estará accesible mediante `this.$route.params` en cada uno de los componentes Vue.

Como en este 🖱️ ejemplo de código, puedes armar un template para luego reflejarlo en la Vista del componente.

Ejemplo
en vivo



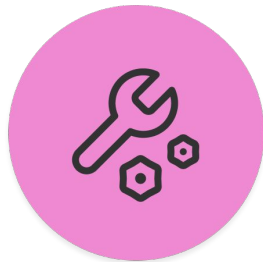
¡VAMOS A PRACTICAR LO VISTO!

CODER HOUSE



BREAK

¡5/10 MINUTOS Y VOLVEMOS!



CREA UNA SINGLE PAGE APPLICATION

Genera una aplicación SPA con **vue ui**.

Tiempo estimado: 15 minutos.

CODER HOUSE



CREA UNA SINGLE PAGE APPLICATION

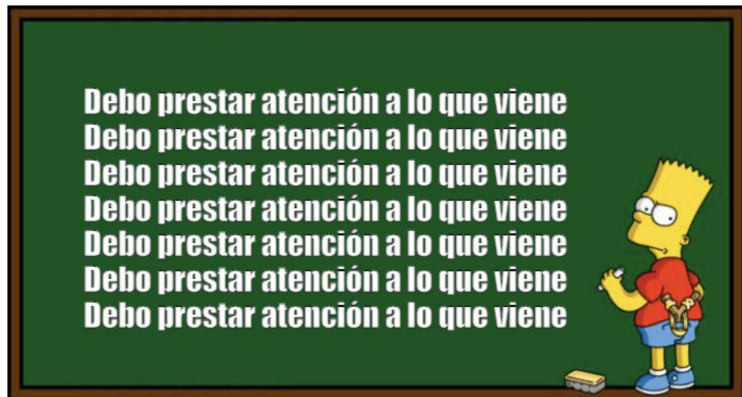
Utiliza la Terminal para ejecutar `vue ui` y generar el proyecto base de una aplicación Vue SPA.

Cuando tengas creado el proyecto, ábrelo con Visual Studio Code y agrega al menos dos Vistas en este. Referencia las vistas dentro del array `routes[]` y genera los `router-links` correspondientes para visualizar las mismas.

Tiempo estimado: 15 minutos.

LIFE CYCLE HOOKS

LIFE CYCLE - HOOKS

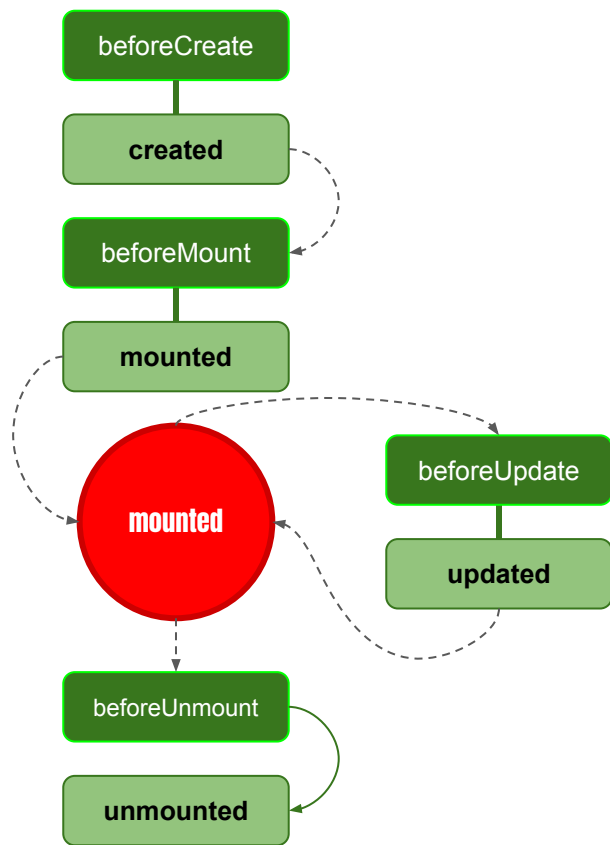


Cuando trabajamos con Vue Components o con instancias de Vue, los componentes transitan por una serie de fases llamadas **Ciclo de Vida**, en inglés: (Life Cycle).

Si bien puede parecer complejo, con el entendimiento de algunos conceptos clave, todo será mucho más fácil.

¡Let's dive in!

CODER HOUSE



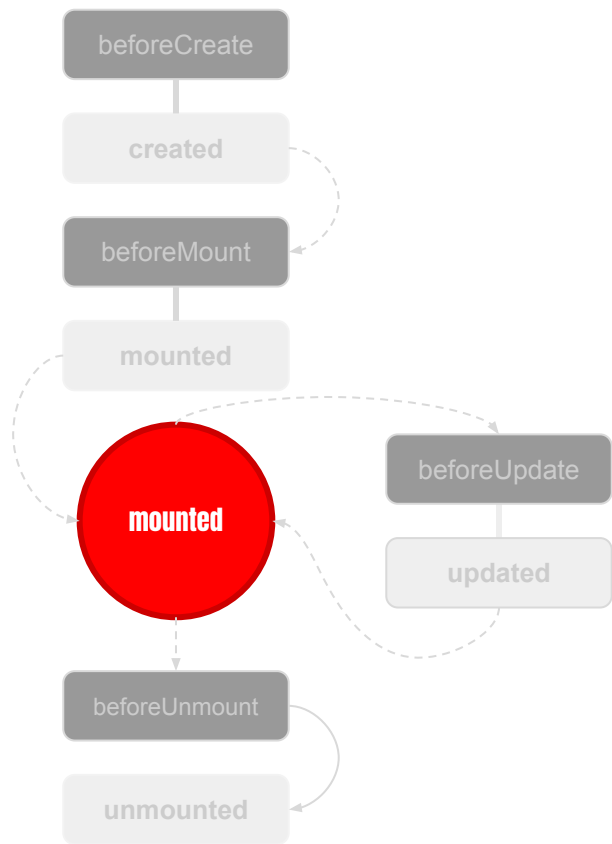
FASES DEL CICLO DE VIDA

👉 Este es un diagrama con las fase del ciclo de vida en Vue.

beforeCreate, la primera de las fases, es donde se inicializa y comienza a crearse el componente.

Esta fase se identifica en color verde oscuro. Tanto esta como el resto de las fases en color oscuro (*precedidas por la palabra **before***) dan inicio a un proceso.

FASES DEL CICLO DE VIDA



👉 El círculo denominado **mounted** identifica el momento en el cual un componente se ha cargado y se encuentra disponible en la Vista.

Veamos una descripción general de cada una de las fases y, luego, un detalle de éstas para saber identificar en qué momento debemos programar y hacer interactuar nuestras funcionalidades, con los componentes de la aplicación.

DESCRIPCIÓN DE LAS FASES

Hook	Momento en el cual se invoca	\$el	\$data
beforeCreate	Se invoca al inicializar y justo antes de procesar opciones.	✗	✗
created	Después de crearse. Los datos se encuentran ya disponibles.	✗	✓
beforeMount	Se invoca inmediatamente antes de la fase de montaje en el DOM.	✗	✓
mounted	Se invoca cuando el componente se muestra en la página (no incluye hijos).	✓	✓
beforeUpdate	Se invoca cuando cambian los datos (antes de modificar el DOM).	✓	✓
updated	Se invoca cuando cambian los datos y el DOM ya fue modificado.	✓	✓
beforeUnmount *	Se invoca justo antes del desmontaje, aunque el componente sigue siendo funcional.	✓	✓
unmounted *	Se invoca en el momento en que un componente (y sus hijos), fueron desmontados.	✓	✓

✗ :: (aún no podemos realizar acciones sobre el elemento y/o los datos)

✓ :: (ya podemos ejecutar acciones sobre el elemento y/o los datos)

* :: (ver aclaración de nomenclaturas en la siguiente diapositiva)



NOMENCLATURA DE FASES

En el detalle de las fases de ciclo de vida tengamos presente que, dentro de Vue/Cli 2, la fase **beforeUnmount** se llama **beforeDestroy** y, la fase **unmounted** se llama **destroyed**.

Los nombres que detallamos aquí corresponden a Vue 3. Este cambio se dió para poder tener un hilo más coherente de los nombres de todo el ciclo de vida de los componentes Vue.

HOOKS

HOOKS

Cada uno de los ciclos de vida de un componente Vue cuenta con su `hook`, a través del cual podremos **definir una función con código específico**, que se ejecutará en el momento preciso que ese ciclo de vida se está ejecutando.

👁️ Veamos algunos ejemplos a continuación:





beforeCreate

Estamos en el momento temporal denominado **beforeCreate**. Disponemos de una propiedad en data llamada **nombreAplicacion** que usamos como título en la Vista.

Si intentara aplicar dicha propiedad en este momento, ocurrirá un error dado que **data:** aún no existe.

ERROR

```
<script>
export default {
  data: {
    nombreAplicacion: 'Mi App Vue/Cli'
  },
  beforeCreate: ()=> {
    console.log('En este momento del
ciclo, data: no existe aún.')
  }
}
</script>
```



created

Estamos en el momento temporal **created**, **data:** y su contenido ya se encuentra accesible.

Podremos acceder la o las propiedades que data contiene como también a los métodos pero, por el momento, seguimos sin poder trasladar nada la Vista en cuestión.

```
...  
  created: ()=> {  
    console.log('En este momento del  
ciclo, ya puedo acceder a data: '  
+ $this.data.nombreAplicacion)  
  }  
}  
</script>
```



beforeMount

Seguimos con acceso a **data:** pero por el momento no podemos interactuar de cara hacia la Vista del cliente.



```
...  
beforeMount: ()=> {  
  console.log( 'data: disponible, pero  
no aún la opción de usar su contenido en  
pantalla')  
}  
}  
</script>
```



mounted

Tenemos vía libre para utilizar el contenido de **data:**
en el componente, aprovechando su reactividad.



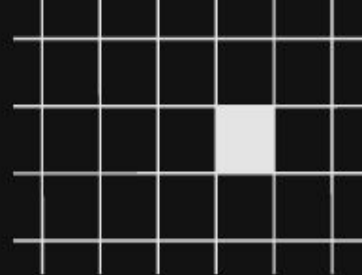
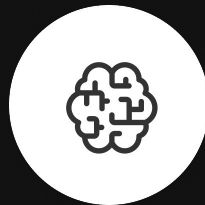
```
...  
  mounted: ()=> {
```

```
    console.log( 'Tenemos vía libre para  
utilizar todo el contenido de data en el  
componente, aprovechando su  
reactividad.')
```

```
  }
```

```
}
```

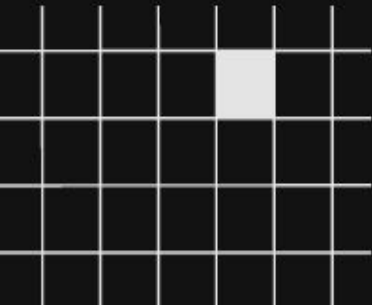
```
</script>
```



¡PARA PENSAR!

¿Recuerdan qué pasaba más allá de haber alcanzado el ciclo `mounted` con los posibles hijos de los componentes?

CONTESTA A TRAVÉS DEL CHAT DE ZOOM





mounted

¡Exacto! Los componentes hijos pueden no haber sido creados todavía.

Si por algún motivo estamos renderizando elementos HTML en este momento del ciclo de vida y queremos acceder a uno de los elementos sin que ocurra un error, podemos recurrir a la función `$nextTick`, contenida en `this`.

```
...  
mounted: ()=> {  
  this.$nextTick()=> {  
    console.log(this.$el.querySelector('h1'))  
  }  
}  
}  
</script>
```

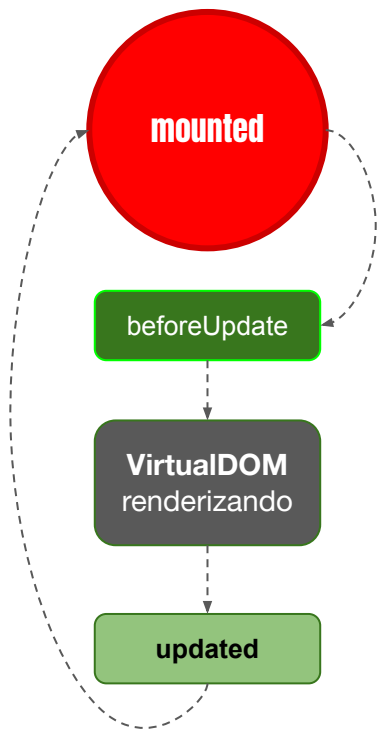
CICLO DE VIDA DE MOUNTED

CICLO DE VIDA DE MOUNTED

En el gráfico principal de Life cycle, pasamos por alto la mención de dos estados: `beforeUpdate` y `updated`. Los mismos solo se establecen cuando el ciclo de vida principal alcanzó el estado `mounted`.

Veamos a continuación qué ocurre en estos ciclos independientes del resto de los mencionados.

CICLO DE VIDA DE MOUNTED



beforeUpdate es el ciclo que comienza a correr en el momento en el cual **mounted** está en modo reactivo y alguna acción se está llevando a cabo en la Vista de la aplicación.

updated, en cambio, entra en vigencia cuando la acción anterior fue llevada a cabo y el **virtualDOM** finalizó el renderizado con la actualización de Vista pertinente.



CICLO DE VIDA DE MOUNTED

```
...  
beforeUpdate: ()=> {  
  console.log('Antes de actualizar...')  
},  
updated: ()=> {  
  console.log('Actualización OK!')  
}  
}  
</script>
```

Por supuesto que, tanto `beforeUpdate` y `updated` pueden ser controlados también, a través del código.



UPDATED

Si bien estos ciclos de vida contenidos en **mounted** nos permiten invocar funciones o métodos para actualizar y renderizar los elementos de la Vista, se recomienda no estar alterando de forma constante la información en **data**, recurriendo en su lugar al uso de **propiedades computadas** o **watchers**.



beforeUnmount / beforeDestroy

Finalmente, en el instante en el cual se descarta un componente web, se ejecuta **beforeUnmount** (Vue3) o **beforeDestroy** (Vue2).

Aquí es donde tenemos la posibilidad de ejecutar, por ejemplo, código que guarde algún estado o enviar datos vía API Restful, previo a eliminar dicho componente.

```
...  
  beforeUnmount: ()=> {  
    console.log( 'Guarda estados o  
cambios en tu aplicación, ¡Ahora!')  
  }  
}  
</script>
```



unmounted / destroyed

`unmounted` (Vue3) o `destroyed` (Vue2) se ocupa de las acciones luego de que el componente ha sido destruido.

Este proceso ocurre cuando cambiamos de Vista o eliminamos un componente del DOM, habiéndolo ocultado mediante, la directiva `v-if`.

```
...  
  unmounted: ()=> {  
    console.log( 'El componente ha sido  
destruido. Hasta la vista, baby! 🕶 ')  
  }  
}  
</script>
```

HOOKS, MIXIN y CANCELACIÓN DE EVENTOS

CICLO DE VIDA Y HOOKS

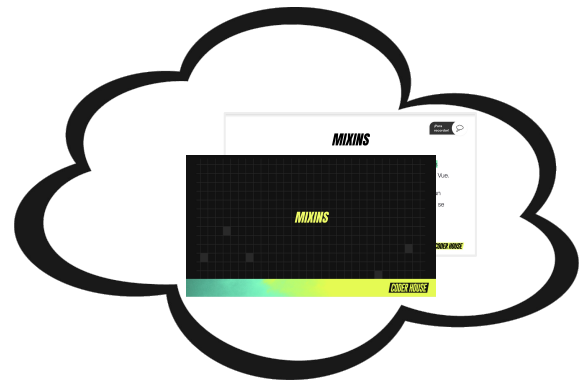
Veamos algunos otros métodos globales que disponemos para ejecutar durante el life cycling de un componente.

<i>Hook</i>	<i>Momento en el cual se invoca</i>
bind	Se invoca la primera vez que se instancia en un elemento.
inserted	Se llama cuando el elemento que tiene una directiva, se insertó en el HTML.
update	Se llama si cambia algún nodo interno del componente.
componentUpdated	Luego de que el nodo padre o alguno de los nodos hijos se actualizaron, se invoca este método.
unbind	Se llama una sola vez, en el momento en que la directiva es quitada del elemento donde ésta se encuentra.

CICLO DE VIDA Y MIXINS

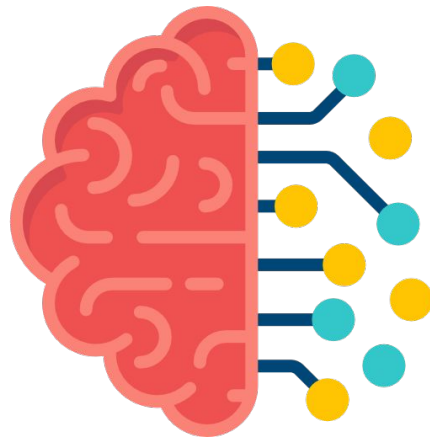
¿Recuerdas que algunas clases atrás, hablamos de los `mixins` en Vue/Cli?

Los mismos **permiten distribuir funcionalidades reutilizables en Vue**, específicamente para componentes. Dijimos que para utilizarlos debíamos definir un objeto con su propia funcionalidad, el cual luego haría uso del mixin, instanciándose previamente.



CICLO DE VIDA Y MIXINS

👉 Con la incorporación de las diferentes etapas de Life Cycling, ten presente que para utilizar un mixin en un componente Vue, cualquier método computada o watcher que haga uso de dicho mixin, deberá ser invocado recién cuando el ciclo `mounted()` se haya completado exitosamente.



CANCELACIÓN DE EVENTOS

Ejemplo
en vivo



Finalmente, si necesitas cancelar algún evento de un componente

Vue que aún sigue visualizado en pantalla, puedes recurrir al
método `$destroy()`.

El mismo se ocupará de invocar a las instancias

`beforeDestroy()` y `destroy()` del componente en cuestión,
cancelando los eventos que estas pueden tener internamente.



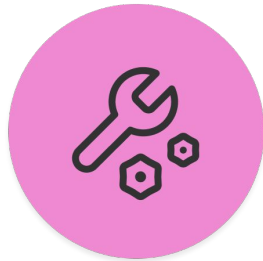
```
...  
methods: {  
  cancelarEventos: ()=> {  
    this.$destroy()  
  }  
}
```

Ejemplo
en vivo



¡VAMOS A PRACTICAR LO VISTO!

CODER HOUSE



LIFE CYCLE HOOKS

Aplicar Life Cycle Hooks para registrar y desregistrar un evento.

Tiempo estimado: 15 minutos.



LIFE CYCLING HOOKS

Agrega, en algún proyecto Vue/Cli que tengas creado, una etiqueta del tipo paragraph o similar donde mostrar un contador numérico.

Establece un método Vue (*el que creas más conveniente*) que aumente el contador cada 1 segundo sobre dicho paragraph usando, por ejemplo: `JS setInterval()`.

Por el lado del Life Cycling, utiliza alguno de los métodos vistos el cual, durante la etapa de carga del componente, comience a ejecutar el contador numérico cuando la etiqueta paragraph ya está visible.

Luego, utiliza otro método cuando se descarga el componente para cancelar `setInterval()` previo a la descarga total de dicho componente.

Tiempo estimado: 15 minutos.

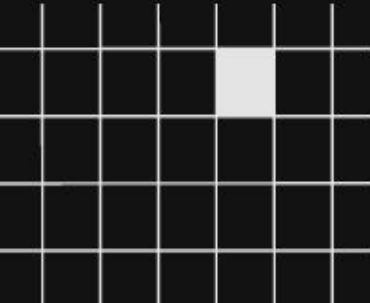
¿PREGUNTAS?





¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- 
- Vue-router
 - Ruteo en HTML y el código
 - Ruteo dinámico
 - Life Cycle hooks
 - Cancelación de eventos



OPINA Y VALORA ESTA CLASE