Consumo de datos desde una API REST

Ciclo de vida de un componente

{desafío} latam_

Implementar un aplicativo web que consume datos desde una API REST utilizando la librería Axios para dar solución a una problemática

- Unidad 1: Introducción a Componentes Web y Vue Js
- Unidad 2: Binding de formularios
- Unidad 3: Templates y rendering en Vue
- Unidad 4: Manejo de eventos y reutilización de componentes
- Unidad 5: Consumo de datos desde una API REST

Te encuentras aquí

{desafío} latam_

¿Qué aprenderás en esta sesión?

- *Utiliza hooks del ciclo de vida de un componente para resolver un problema planteado.*

{desafío} latam_

¿Qué es la asincronía?

¿Cómo la asincronía afecta el comportamiento de una aplicación?

{desafío} latam_

¿Qué son las promesas y cómo se manipulan en JavaScript?

{desafío} latam_

¿Qué son las funciones async/await?

{desafío} latam_

¿Qué es el try / catch ?

{desafío}
latam_

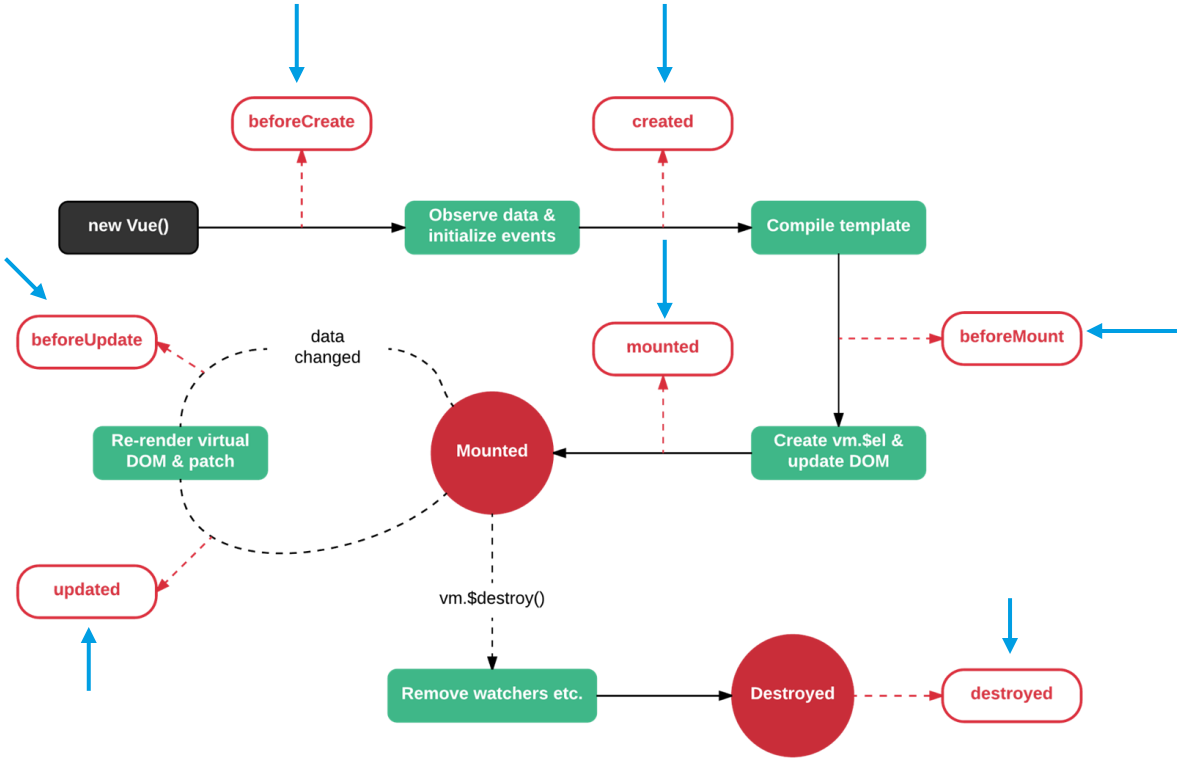
/* Ciclo de vida de un componente */

Ciclo de vida de un componente

ciclos

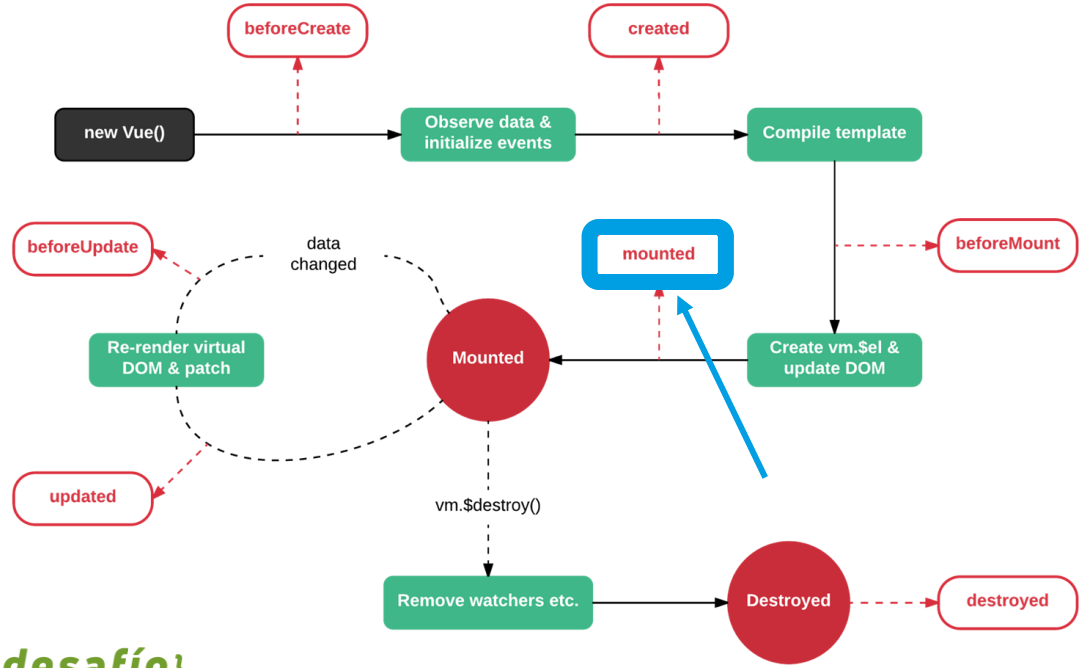
Cada componente Vue Js pasa por un ciclo de vida también conocido como Lifecycle Hooks y está dividido en **7 etapas**.

Podemos ingresar en cada una de estas etapas para ejecutar lo que necesitemos.



Ciclo de vida de un componente

mounted



Todos los hooks se abordan de la misma manera declarándose en la exportación del componente.

Así que usemos el **mounted** como ejemplo para ejecutar un código en nuestro componente justo luego de montarse en el DOM.

De esta manera no dependeremos de una acción del usuario para realizar alguna tarea inicial

{desafío} latam_

Ejercicio guiado 1

{desafío} latam_

Ciclo de vida de un componente

mounted

1. Declara el método mounted directamente el la exportación del componente

1. Dentro del **mounted** agrega un console.log() que muestre por consola “Likecycle hooks”

1. Abre el navegador y observa en la consola el mensaje

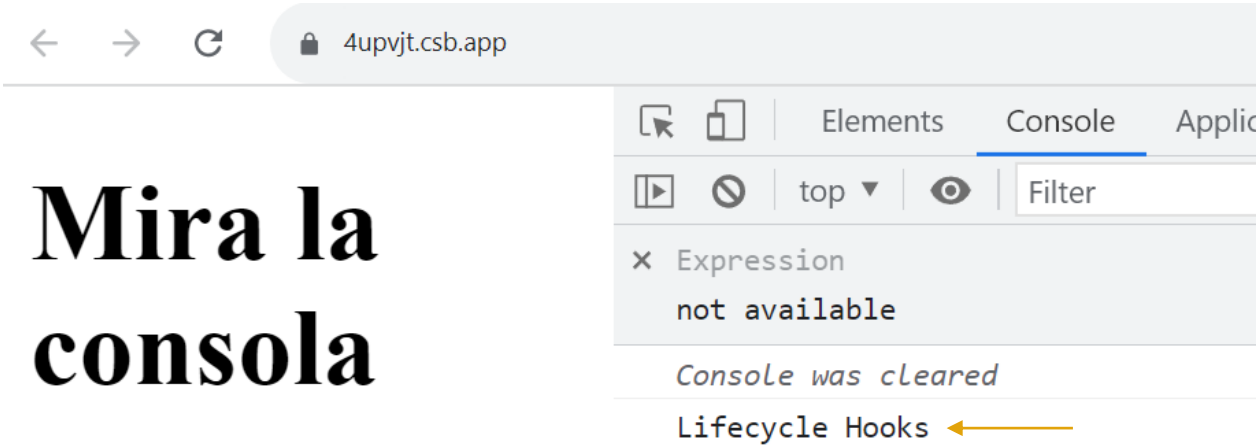
```
<template>
 <div>
   <h1>Mira la consola</h1>
 </div>
</template>

<script>
export default {
 mounted() {
   console.log("Lifecycle Hooks");
 },
};
</script>
```

Ciclo de vida de un componente

mounted

De esta manera podemos ejecutar algún código justo al iniciar nuestra aplicación.



¿Qué se te ocurre que sea útil ejecutar dentro del `mounted`?

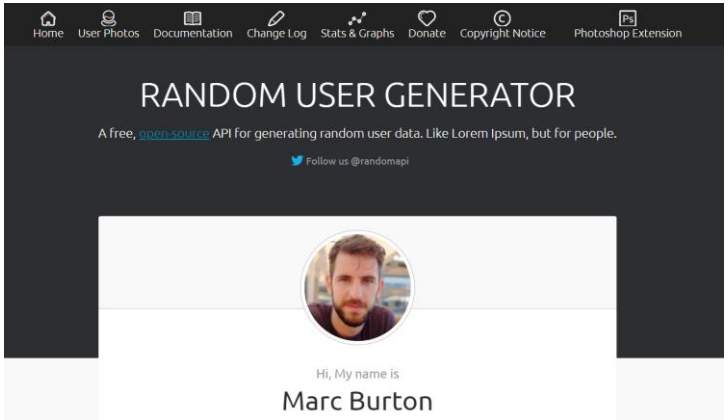
Ejercicio guiado 2

Ciclo de vida de un componente

mounted

Uno de los usos más comunes del **mounted** es el consumo de una API REST para asignar una data inicial en nuestra aplicación.

Veamos un ejemplo donde mostremos 30 fotos de usuarios aleatorios obtenidos de la API random user.



Ciclo de vida de un componente

mounted

1. Crea una variable en el estado llamada **usuarios**

```
data(){
   return {
     usuarios: []
   }
 }
```

Esta variable se crea como un arreglo vacío pero en el **mounted** se sustituirá por el arreglo de objetos obtenidas de la API

Ciclo de vida de un componente

mounted

2. Agrega en un bloque en el template que se renderice dinámicamente usando el arreglo usuarios y muestre la foto de cada usuario usando el atributo **picture.large**

```
<template>
 <div id="App">
   <div v-for="(usuario, i) in usuarios" :key="i">
     <img :src="usuario.picture.large" />
   </div>
 </div>
</template>
```

Ciclo de vida de un componente

mounted

3. Agrega un pequeño estilo para estilizar las fotos en forma de grilla usando CSS Grid

```
<style>
#App {
 display: grid;
 grid-template-columns: repeat(6, 1fr);
 gap: 10px;
}
</style>
```

Ciclo de vida de un componente

mounted

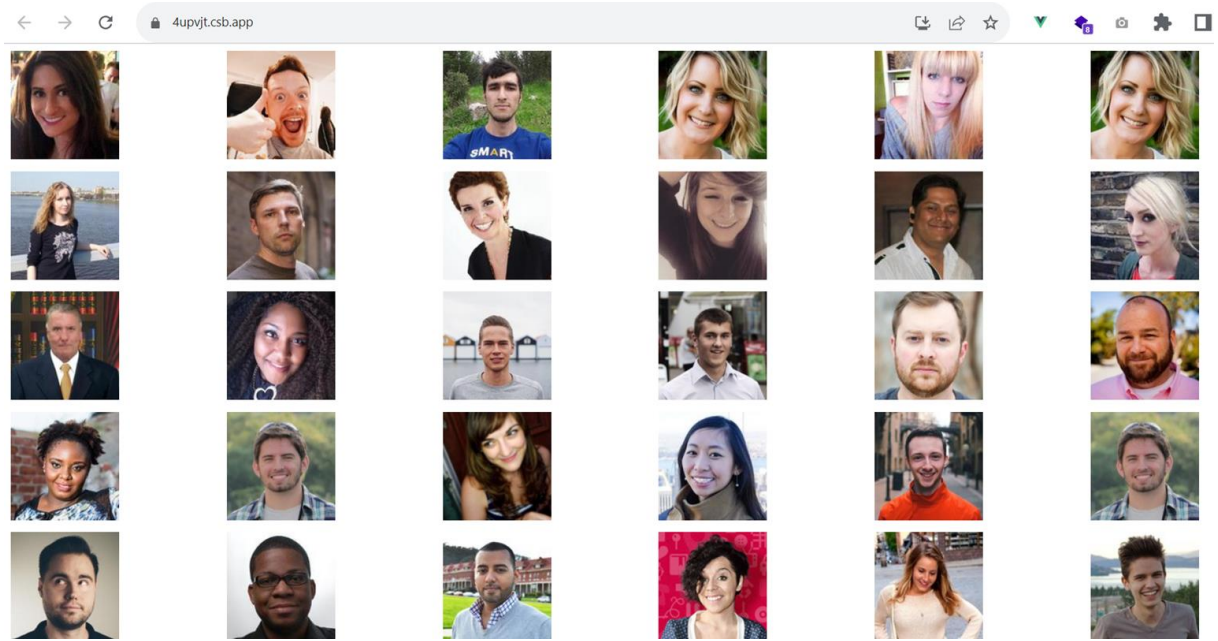
4. Utiliza el mounted y axios para consultar la API random user usando el siguiente endpoint:
   https://randomuser.me/api?results=30

Asigna el atributo **results** de la **data** como valor del arreglo usuarios del estado

```
async mounted() {
 const url = "https://randomuser.me/api?results=30";
 const { data } = await axios.get(url);
 this.usuarios = data.results;
},
```

Ciclo de vida de un componente

mounted



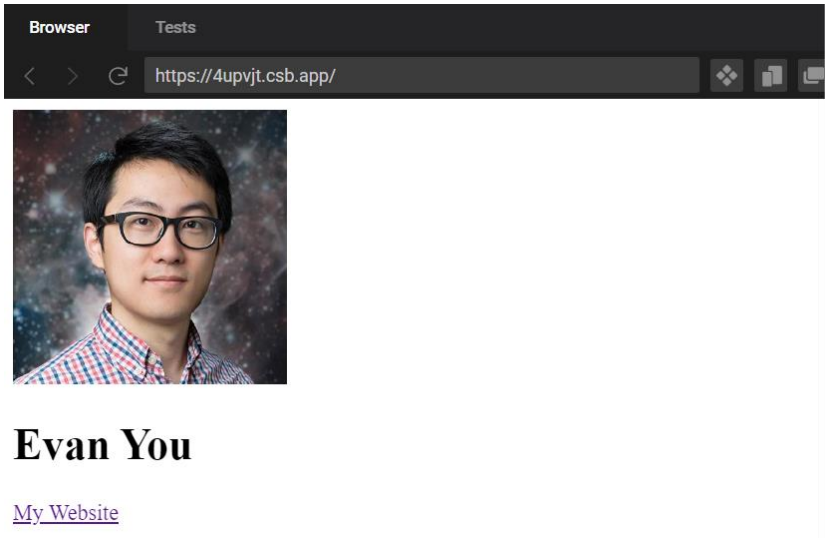
Ahora en el navegador podremos ver al abrir nuestra aplicación nuestra grilla con fotos de usuarios aleatorios.

Ejercicio propuesto 1

Ciclo de vida de un componente

mounted

- Crea una aplicación que al cargar muestre el detalle tu cuenta de Github usando el siguiente endpoint: https://api.github.com/users/**:username**

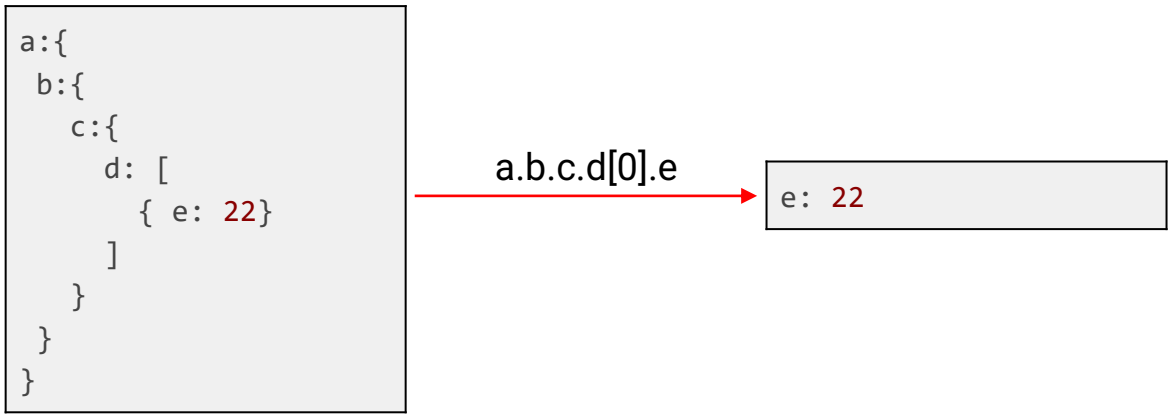


{desafío} latam_

/* computed properties */

Computed properties

Cuando consumimos datos de las API's en muchos casos sucede que recibimos mucha más información de la que realmente necesitamos y/o debemos mapear la información para acceder a los datos que necesitamos.



Computed properties

Random User por ejemplo nos provee un JSON con varios niveles anidados entre objetos, arreglos, números y strings.

Las computed properties son métodos reactivos que podemos ocupar para obtener valores depurados o procesados del estado.

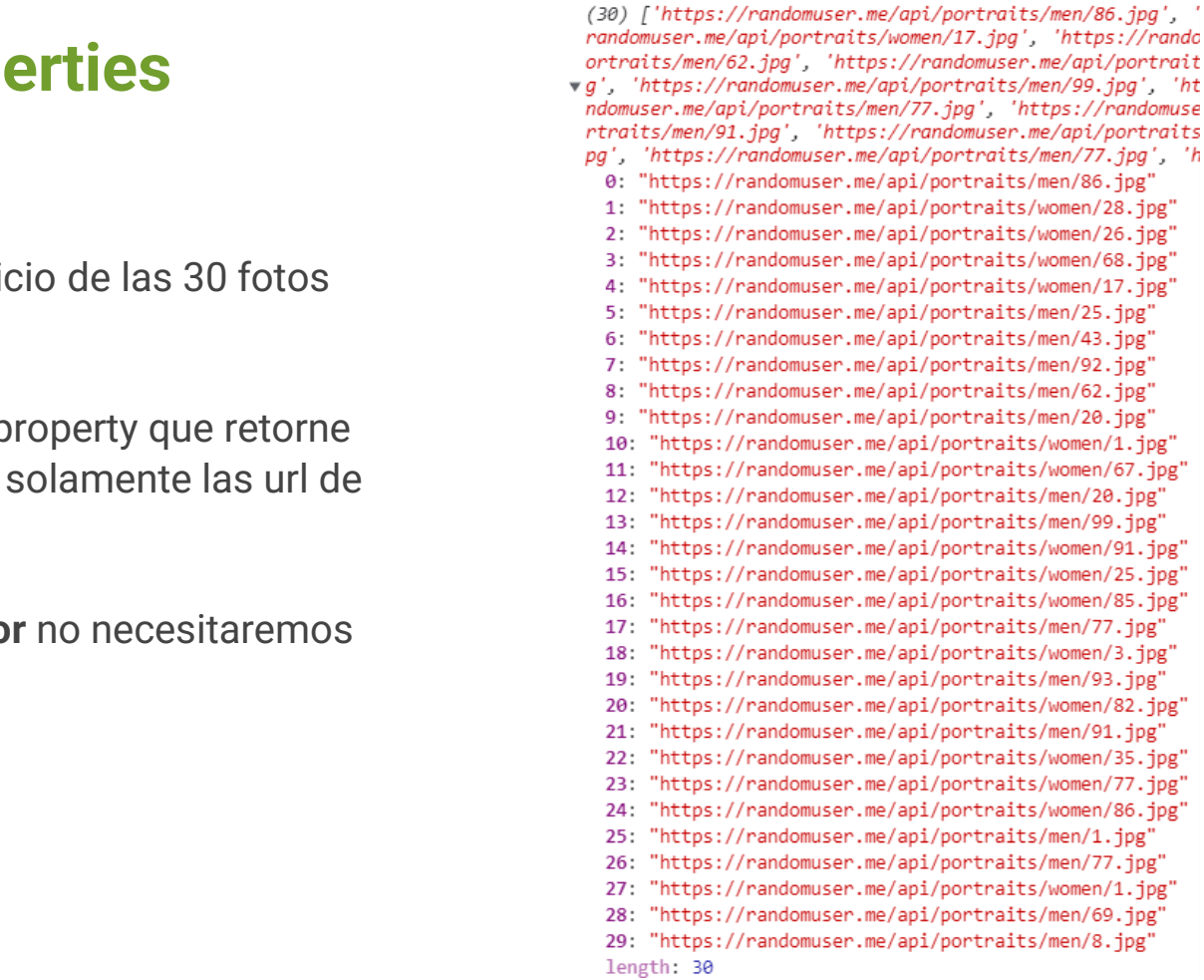
Ejercicio guiado 3

Computed properties

Continuando con el ejercicio de las 30 fotos de usuarios aleatorios…

Creemos una computed property que retorne un arreglo de strings con solamente las url de las fotos.

De esta manera en el **v-for** no necesitaremos mapear el objeto **usuario**



{desafío} latam_

Computed properties

1. Agrega en el objeto de exportación del componente el atributo computed con un método llamado **fotosDeLosUsuarios.**

```
computed: {
 fotosDeLosUsuarios() {
   return this.usuarios.map((usuario) => usuario.picture.large);
 },
},
```

Todos los computed deben retornar un valor

Computed properties

2. Ahora modifiquemos el v-for para utilizar la computed property **fotosDeLosUsuarios**

```
<template>
 <div id="App">
   <div v-for="(foto, i) in fotosDeLosUsuarios" :key="i">
     <img :src="foto" />
   </div>
 </div>
</template>
```

De esta manera lograremos que nuestro template sea más fácil de leer

Ejercicio propuesto 2

Computed properties

- Crea una computed property que retorne un objeto con solo los siguientes campos:
  - **nombre, img, id**

  Con el JSON que se obtiene al consultar el siguiente endpoint:
  https://pokeapi.co/api/v2/pokemon/25


```
▼ {nombre: 'pikachu', img: 'https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/25.png', id: 25}
    id: 25
    img: "https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/25.png"
    nombre: "pikachu"
```

Hemos terminado el contenido del módulo

¡Felicidades!

Próxima sesión...

- *Guía de ejercicios.*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales