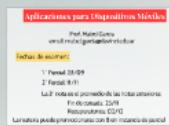


ADM

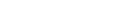
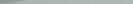
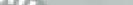
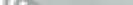
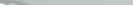
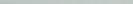
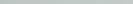
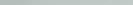
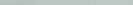
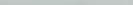
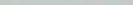
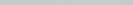
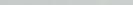
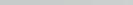
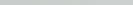
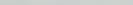
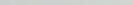
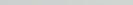
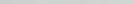
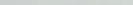
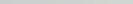
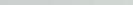
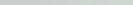
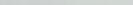
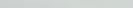
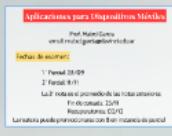


Dudas?



ADM

Dudas?



Aplicaciones para Dispositivos Móviles

Prof. Mabel García

email: mabel.garcia@davinci.edu.ar

Fechas de examen:

1º Parcial: 23/09

2º Parcial: 11/11

La 3º nota es el promedio de las notas anteriores

Fin de cursada: 25/11

Recuperatorios: 02/12

La materia puede promocionarse con 8 en instancia de parcial

Surgimiento

- Creado por Evan You en 2014, trabajando en Google Labs
- inspirado en Angular.js , Reactive.js y Rivets.js ofrece un diseño simplificado de interfaz de usuario.
- Es un framework progresivo para construir interfaces de usuario
- Pogresivo significa que podemos usarlo para algo muy básico, o agregarle más elementos para aplicaciones más complejas.

VueJS

- Trabaja solo en la capa de la vista, siendo sencillo integrar con otros proyectos o bibliotecas existentes.

Características:

- **Framework progresivo de código abierto :**

Podemos incluir las herramientas que necesitemos según precisemos.

- **Virtual Dom:**

Es cualquier tipo de representación de un DOM real. Trata de evitar cambios innecesarios en el DOM, que son caros en cuanto a rendimiento.

La re-representación solo ocurre una vez que se aplica un conjunto de cambios al DOM.

VueJS

Representación declarativa:

Podemos crear variables en nuestras aplicaciones e imprimirlas en la vista.

Componentes :

Pieza de código independiente que representa una parte de la página. Los componentes tienen sus propios datos, su propio JavaScript, etc.

Adaptabilidad:

Se puede añadir a aplicaciones web ya existentes y aprovecharlo sin mayores problemas de instalación.

Patrón MVVM (Modelo Vista - Vista Modelo)

Patrón de diseño o modelo de abstracción usados para definir y estructurar los componentes necesarios en el desarrollo de software.

VueJS

El modelo

Representa la capa de datos y/o la lógica de negocio, contiene información.

La vista

Debe representar la información a través de los elementos visuales que la componen.

Sistema Reactivo:

- Mantiene interacción constante con su entorno. por medio de eventos

Referencias a tener presente

URL Documentación Oficial : <https://vuejs.org/guide/introduction>

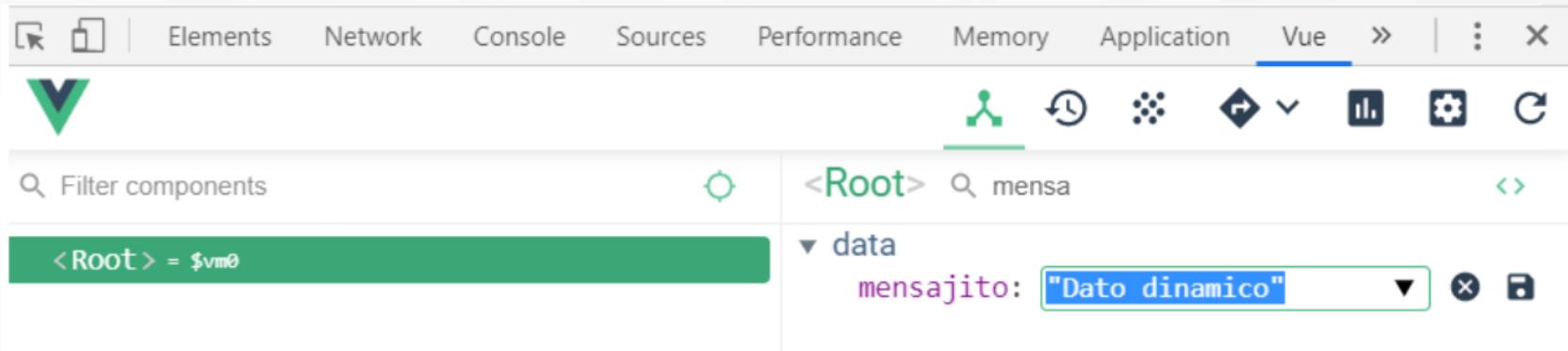
Versión a usar : v3.x Api Options



VueJS

Herramienta para debug

- Extensión para el navegador (chrome y Firefox)



VueJS

Quién usa vuejs?



euronews.

PLAYCODE

Code Sandbox. Online Code Editor.



Carrefour

LARAC ➔STS

Manos a la obra

- Necesitamos incorporar por cdn este script para poder trabajar con Vue

<!-- versión de desarrollo, incluye advertencias de ayuda en la consola -->
<script src="<https://unpkg.com/vue@3/dist/vue.global.js>"></script>
- Nosotros vamos a ir a esta dirección y vamos a tenerlo de manera local entre nuestros archivos.
- Es la forma más sencilla para empezar y necesitamos que vue funcione, más allá de la conexión de red.
- Vamos a tener **otro archivo js** en el cual vamos a desarrollar la lógica para poder trabajar.

VueJS

- Este archivo tendrá la función `createApp` retorna, un componente que se almacena en la constante `app`.

```
const app = Vue.createApp({  
  // Opciones de configuración  
});  
app.mount('#app');
```

- Cada aplicación requiere un "componente raíz" que pueda contener otros componentes como sus hijos.
- Tenemos que establecer en donde va a trabajar `Vue`, y para ello, necesitamos usar un método `.mount()`.
- El argumento que espera que le pasemos es un string con sintaxis CSS que apunte a su respectivo id, class o selector.
- "`#app`" debe existir en nuestro HTML

VueJS

- Primera opción del objeto createApp será la **función data** que debe retornar siempre un Object. Vue guardará todas las variables locales que deba manipular.
- En la vista tendremos un **div con el id app** y dentro de un párrafo unas **dobles llaves** con el nombre de la propiedad del objeto data que queremos mostrar.
- Esto se conoce como **interpolación {{ }}**, es un **enlace de datos**, que mostrará el valor de la variable.

```
<div id="app">  
  <p>{{mensaje}}</p>  
</div>
```

```
const app = Vue.createApp({  
  data() {  
    return {  
      mensaje: "Aprendiendo Vue"  
    }  
  },  
});  
app.mount('#app');
```

- Desde la apertura del div "app" hasta el cierre de la etiqueta será el **ámbito de trabajo de Vue, en donde se monta la instancia**.

VueJS

- Los **datos** y el DOM ahora están vinculados
- Si vamos a la consola del navegador, podremos ver que ahora este elemento es **reactivo**.
- Si le **cambiamos el contenido a la variable mensaje**, veremos **inmediatamente** el cambio.

Aprendiendo Vue

The screenshot shows the Vue.js DevTools interface. The title bar says "Aprendiendo Vue". The top navigation bar includes "Inspector", "Consola", "Depurador", "Red", "Editor de estilo", and "Vue". Below the title bar, there are two tabs: "Components" (which is selected) and "Timeline". A search bar at the top has fields for "Find apps..." and "Find components...". The main area displays the state of the "App 1" component, which is version 3.4.19. A green box highlights the "data" section. Inside "data", the "mensaje" key is set to the value "Aprendiendo Vue".

Hola Chicos!

This screenshot shows the same Vue.js DevTools interface as the previous one, but with a change in the "data" section of the "App 1" component. The "mensaje" key is now set to the value "Hola Chicos!". The rest of the interface remains the same, with the title bar, tabs, and search bar visible.

- Se pueden realizar interpolaciones únicas que no se actualizan en el cambio de datos, por medio de una directiva.

Directivas

- Las directivas ofrecen distintos tipos de comportamientos reactivos especiales al DOM representado sobre las etiquetas en las cuales se aplican.
- Son atributos especiales para un elemento con prefijo **v-algo**
- **v-once** --> Renderizará el elemento solo una vez. Esto puede ser utilizado para optimizar la performance de actualización.

```
<h1 v-once>Esto no va a cambiar: {{ titulo }}</h1>
```

VueJS

v-show --> Esta directiva permite mostrar de modo condicional un elemento.

- Este se renderizará y lo veremos en el DOM.
- Si el valor de la variable es true, mostrará el elemento, si es false le aplicará la propiedad **display:none**

Este texto se mostrará en tanto la variable ver tenga valor true, caso contrario se le aplicará display none

```
<p v-show="visible">Este texto ... </p>
```

```
return {  
  visible: true  
}
```

VueJS

- Los mustaches (llaves doble) interpretan los datos como texto plano, no HTML.
- Para generar HTML real, se utiliza la directiva **v-html**:

```
return {  
  codigo: "<h1> Escrito con etiquetas html se usa v-html </h1>"  
}
```

```
<div v-html="codigo"></div>
```

- Si dentro de la etiqueta **div** tuvieramos escrito algún texto, sería reemplazado por el valor que posee la variable **codigo**
- Después veremos como crear bloques de código de otra manera.

VueJS

v-cloak--> Se le agrega al elemento hasta que la instancia de Vue asociada finalice la compilación.

- Requiere usar reglas CSS como [v-cloak] { display: none }
- Se usa para **ocultar interpolaciones {{codigo}}** sin compilar hasta que la instancia de Vue esté lista.

```
<div id="app" v-cloak>
  <!-- todo lo que esté dentro no se mostrará hasta que se compile
  -->
</div>
```

VueJS

v-pre--> Omite la compilación para el elemento y todos sus elementos secundarios

- Sirve para mostrar mustaches sin procesar.
- Si se omiten grandes cantidades de nodos sin directivas se puede acelerar la compilación

```
<p v-pre>Directiva v-pre: {{mensajito}}</p>
```

- La variable existe, pero no la compilará

Directiva v-pre: {{mensajito}}

VueJS

- Los mustaches (llaves doble) **no se pueden utilizar** dentro de los **atributos HTML**.

v-bind--> Enlaza dinámicamente uno o más atributos, o una propiedad de un componente a una expresión.

Por ejemplo, si quisieramos enlazar una imagen desde nuestra lógica.

```
const app = Vue.createApp({  
  data() {  
    return {  
      mensaje:"Aprendiendo Vue",  
      visible:true,  
      imagen: "img/gato.jpg",  
      descripcion:"Gato jugando con una mariposa",  
      booleano:true,  
    }  
  }  
})
```

```
<img v-bind:src ="imagen" v-bind:alt = "descripcion"/>
```

VueJS

- También se puede usar la sintaxis abreviada:

```
<img :src ="imagen" :alt="descripcion"/>
```

Manipular clases y estilos

- Tanto los **class** como los **style** que usamos en html son atributos que podemos afectar desde vue.
- Por medio de **v-bind** podemos manejarlos, evaluando objetos o matrices
- Podemos pasar un **objeto** para poder **asignar clases** a un elemento.

VueJS

HTML

```
<div v-bind:class="{ red: esRojo }">
```

clase

variable

CSS

```
.red{color:red;}
```

js

```
return {  
    esRojo: true  
}
```

- También permite pasar un **array** para aplicar una lista de clases:

```
<p v-bind:class="[mal, error]">
```

variable

variable

VueJS

Dato dinámico



Elements Network Console Sources Performance Memory App

> <head>...</head>

▼ <body>

 ▼ <div id="mostrar">

 <h1>Esto no va a cambiar: Soy un título</h1>

 ▼ <div class="red">

 <p class="malClass errorClass"> Dato dinámico</p> == \$0

 </div>

Styles Computed Event Listeners

Filter

element.style {

}

.errorClass {

background: ▢ red;

color: □ white;

}

.malClass {

 text-decoration: ▢ line-through;

Resumiendo

- Una aplicación de Vue consiste en una **instancia raíz** creada con la función **createApp**.
- A futuro va a estar organizada en un **árbol de componentes anidados y reutilizables**.
- Al crearse una instancia de Vue, se **agregan todas las propiedades encontradas la función data** al sistema de reactividad de Vue .
- Cuando los **valores** de esas propiedades **cambian**, la vista "reaccionará" y **se actualizará** para coincidir con los nuevos valores.

Resumiendo

- La **vista** y el **modelo** se vinculan por medio de **interpolaciones**
- Permiten que **los valores** de las **variables** declaradas en el modelo, se puedan **ver en la vista**.
- Las **directivas** otorgan nuevos comportamientos a los elementos del html.
- Podemos tener **más de una directiva afectando un mismo elemento**.



Dudas?

ADM



Dudas?

