



Clase 08

Diseño y Programación Web

Materia:

Aplicaciones para Dispositivos Móviles

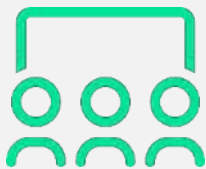
Docente contenidista: GARCIA, Mabel

Revisión: Coordinación

Contenido

Enrutamiento	04
Vue-router.....	05
Componentes RouterLink y RouterView	06
Configuración Sistema de enrutamiento	09
Configurar página de error	10
Componente Transition	16
Transitions: modos de transición	22
Transition-group.....	24
Aplicar transition a router-view	26
Bibliografía	29

Clase 8



iTe damos la bienvenida a la materia
Aplicaciones para Dispositivos Móviles!

En esta clase vamos a ver los siguientes temas:

- Concepto de Enrutamiento.
- Vue-router (componentes router-link, componente router view, configuración sistema de enrutamiento).
- Configurar página de error.
- Componente transition : clases CSS de Vue.
- Transitions modo de transition.
- Transition-group.
- Aplicar transition a router-view.

Enrutamiento

En el contexto de las aplicaciones web, el enrutamiento se refiere a la gestión de las rutas o URLs que un usuario puede visitar dentro de la aplicación. Existen dos enfoques principales para el enrutamiento: el enrutamiento del lado del servidor y el enrutamiento del lado del cliente.

El **enrutamiento del lado del servidor** implica que el servidor es responsable de manejar las rutas y enviar una respuesta basada en la URL solicitada por el usuario. Cada vez que se accede a una URL diferente, el servidor genera y envía una nueva página HTML al navegador, lo que resulta en la recarga completa de la página.

Por otro lado, en el **enrutamiento del lado del cliente**, el JavaScript en el navegador asume el control de las rutas. Cuando un usuario hace click en un enlace dentro de la aplicación, el JavaScript intercepta la navegación y carga dinámicamente los datos necesarios para actualizar la página actual sin recargar por completo. Esto es especialmente útil para aplicaciones de una sola página (SPA) donde se busca una experiencia de usuario más fluida y ágil.

Vue.js es una herramienta poderosa para construir **aplicaciones de una sola página (SPA)** gracias a su capacidad para gestionar de forma eficiente las actualizaciones dinámicas de la interfaz de usuario. Para implementar un **enrutamiento en Vue.js**, se recomienda utilizar la librería oficial **Vue Router**.

Vue Router nos brinda un **sistema de enrutamiento** flexible del lado del **cliente** y fácil de usar, permitiendo crear aplicaciones SPA con **múltiples vistas y transiciones** suaves entre páginas.

Vue-router

Vue Router es la solución oficial de enrutamiento del lado del cliente para Vue.

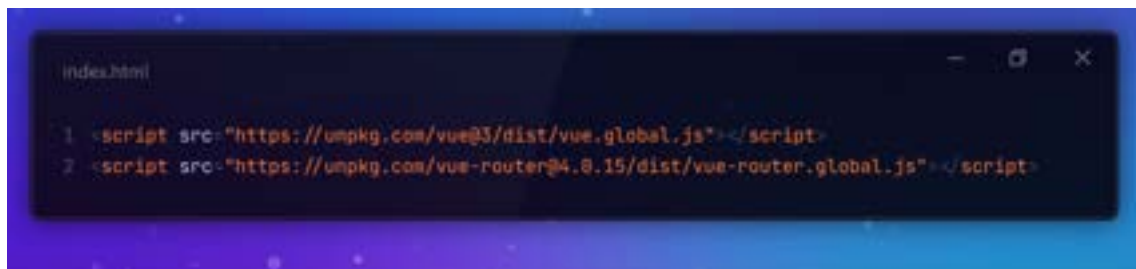
El enrutamiento del lado del cliente es **utilizado por aplicaciones de una sola página (SPA)** para vincular la URL del navegador al contenido que ve el usuario. A medida que los usuarios navegan por la aplicación, la URL se actualiza en consecuencia, pero la página no necesita volver a cargarse desde el servidor.

Vue Router está construido sobre el sistema de componentes de Vue. **Configuramos rutas para indicarle a Vue Router qué componentes mostrar para cada ruta de URL.**

Para poder trabajar con Vue router hay que agregarlo a nuestro proyecto ya que no viene dentro del core de funcionalidades de Vue. Por lo tanto, debemos incorporarlo con un script por CDN, luego de haber insertado la llamada a Vue en nuestro index.

Hay que aclarar, que al estar trabajando con Vue3, necesitamos trabajar con Vue Router 4, esta es la versión compatible para usar.

Además, precisamos un servidor para que las rutas se manejen correctamente. En tal caso, nos sirve usar el plugin de Visual Studio Code (Live Server).

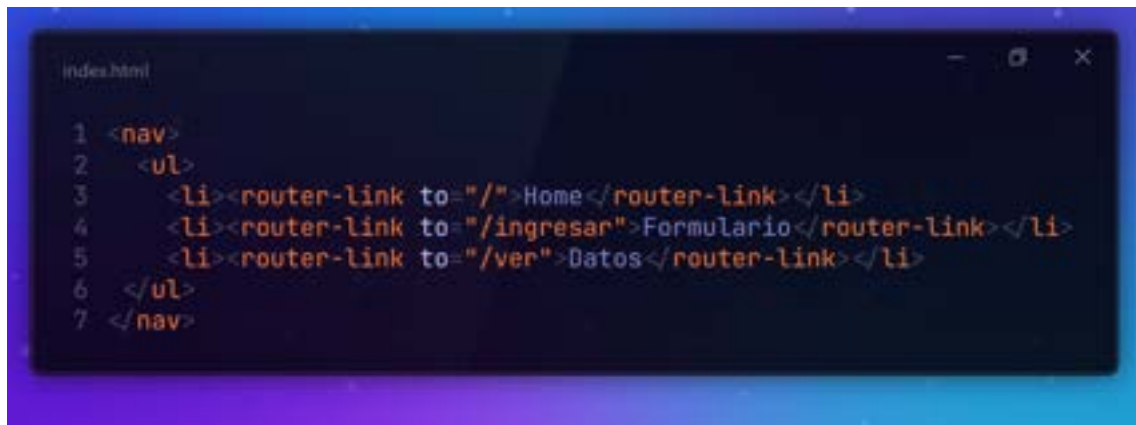


```
index.html
1 <script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
2 <script src="https://unpkg.com/vue-router@4.8.15/dist/vue-router.global.js"></script>
```

Componentes RouterLink y RouterView

Una vez incorporado los scripts al index.html, vamos a necesitar en nuestro HTML usar dos componentes proporcionados por Vue Router, **RouterLink** y **RouterView**.

En lugar de usar etiquetas **<a>** regulares, usamos el componente personalizado **RouterLink para crear enlaces**. Esto permite que **Vue Router cambie la URL sin recargar la página**, maneje la generación de URL, codificación y diversas funciones adicionales. Detallaremos más sobre RouterLink en secciones posteriores.



El componente **RouterLink** agrega **dos clases CSS a los enlaces activos, router-link-active y router-link-exact-active**. Para comprender la diferencia entre ellas, primero debemos considerar cómo Vue Router determina que un enlace está activo.

Un enlace está activo en Vue Router si:

- Coincide con la misma ruta configurada que la ubicación actual.
- Tiene los mismos valores para los parámetros que la ubicación actual.
- Si estás usando rutas anidadas, cualquier enlace a rutas superiores también estará activo si los parámetros coinciden.

Por lo tanto, para determinar un link activo tenemos que tener la coincidencia exacta de la ruta. Para tal caso, usaremos la clase **router-link-exact-active** para que desde nuestro css establezcamos estilos visuales.



```

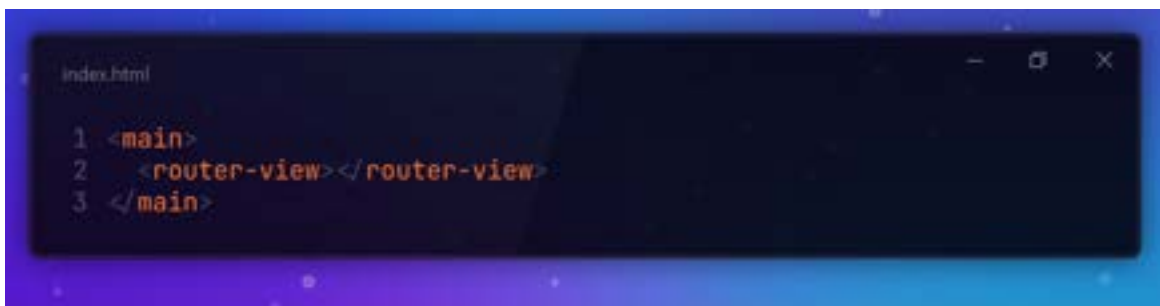
1 .router-link-exact-active {
2   color: white;
3   background: blue;
4   width: 150px;
5   text-decoration: none;
6   padding: 10px;
7 }
8

```

De esta forma le informaremos al usuario en qué parte del sistema se encuentra navegando.

El componente **RouterView** indica a Vue Router **dónde renderizar el componente de ruta actual**. Este componente corresponde a la ruta de URL actual. Se puede colocar en cualquier lugar para adaptarlo a nuestro diseño, pero debe estar incluido en algún lugar, de lo contrario, Vue Router no renderizará nada.

Es común que las aplicaciones tengan un componente de navegación que renderiza una lista de componentes RouterLink. Dentro de esa lista, es posible que deseemos aplicar estilos diferentes a los enlaces de la ruta actualmente activa en comparación con los demás.



```

1 <main>
2   <router-view></router-view>
3 </main>

```

Cuando creamos nuestros archivos con los componentes que necesitamos, también debemos agregarlos al archivo index.html para que Vue los reconozca y pueda utilizarlos.

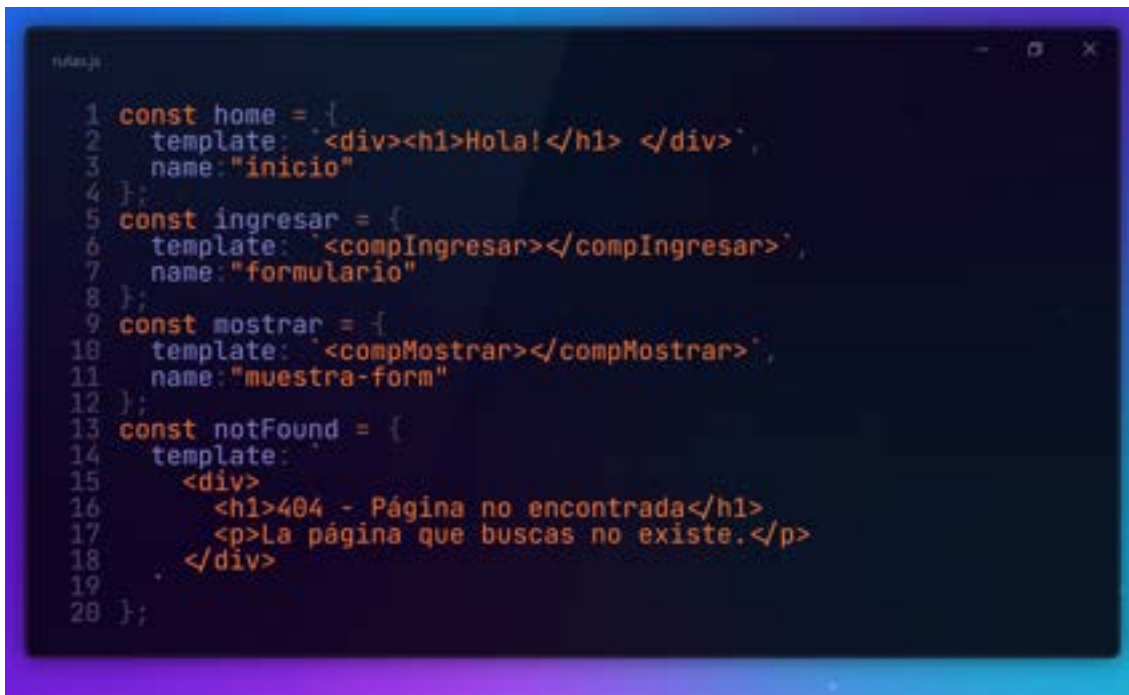
Por ejemplo, si tenemos componentes en la carpeta "**componentes**", como **ingresar.js** y **mostrar.js**, debemos referenciarlos por medio de la etiqueta `<script>` en nuestro archivo HTML. Además, el archivo **rutas.js** contendrá toda la configuración de las rutas para trabajar con el enrutamiento en Vue Router.

```
index.html

1 <script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
2 <script src="https://unpkg.com/vue-router@4.0.15/dist/vue-router.global.js"></script>
3 <script src="componentes/ingresar.js"></script>
4 <script src="componentes/mostrar.js"></script>
5 <script src="scripts/rutas.js"></script>
```


Configuración Sistema de enrutamiento

Para entender cómo funcionan las rutas en Vue.js, es necesario comprender la relación entre las rutas y los componentes asociados a ellas. En el archivo de rutas.js, definimos varios componentes que utilizaremos a lo largo de la aplicación.



```
1 const home = {
2   template: `<div><h1>Hola!</h1> </div>`,
3   name: "inicio"
4 };
5 const ingresar = {
6   template: `<compIngresar></compIngresar>`,
7   name: "formulario"
8 };
9 const mostrar = {
10  template: `<compMostrar></compMostrar>`,
11  name: "muestra-form"
12 };
13 const notFound = {
14   template: `
15     <div>
16       <h1>404 - Página no encontrada</h1>
17       <p>La página que buscas no existe.</p>
18     </div>
19   `,
20  };
21 
```

Cada **constante** representa **un componente de Vue que asociaremos a una ruta específica de nuestra aplicación**.

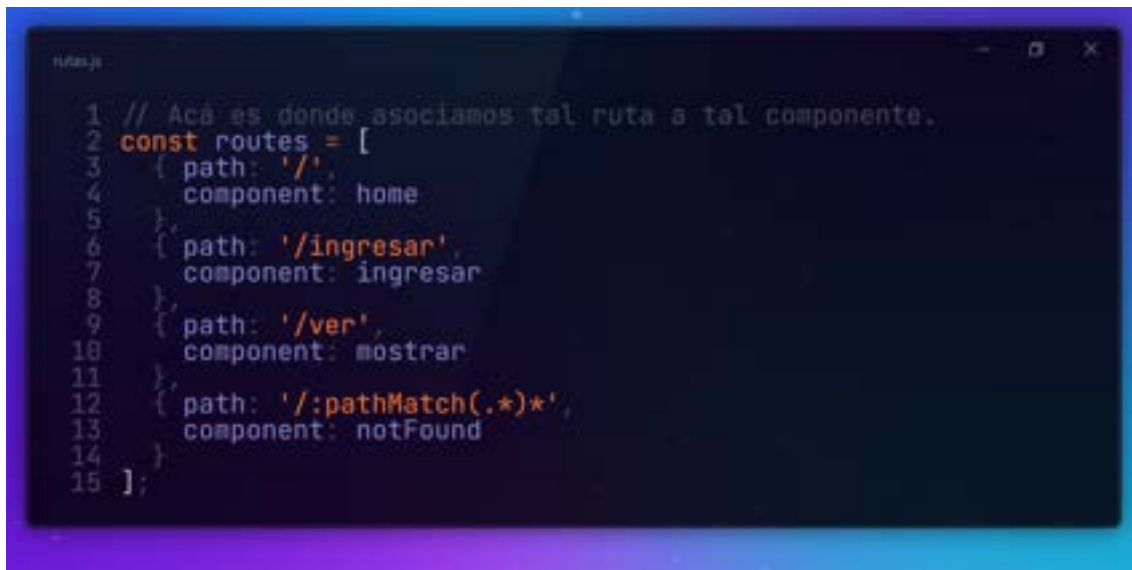
La propiedad **template** en cada definición de componente contiene la **estructura HTML y posiblemente el componente en sí**, si está desarrollado en su propio archivo.

El atributo **name** asigna un nombre legible al componente, lo cual es útil para identificarlo y referenciarlo en el código de Vue.

A medida que definimos más rutas y componentes, construimos la arquitectura de nuestra aplicación Vue de manera modular y escalable.

Una vez que hemos definido nuestros componentes y sus respectivos template en el archivo rutas.js, debemos **asociarlos con las rutas de nuestra aplicación**. Esto se logra mediante la definición de un **arreglo de rutas dentro de la constante routes**.

javascript



```
1 // Acá es donde asociamos tal ruta a tal componente.
2 const routes = [
3   { path: '/',
4     component: home
5   },
6   { path: '/ingresar',
7     component: ingresar
8   },
9   { path: '/ver',
10    component: mostrar
11   },
12   { path: '/*',
13     component: notFound
14   }
15 ];
```

Cada objeto dentro de **routes** representa una **ruta específica y su correspondiente componente**. Por ejemplo, la ruta '/' está asociada al componente **home**, la ruta '/ingresar' está asociada al componente **ingresar**, y así sucesivamente. **El valor de cada componente es el nombre de cada constante anteriormente definida.**

Configurar página de error

Además, en el caso de que el usuario acceda a una ruta no definida en nuestra aplicación, mostraremos el **componente notFound**, que representa una **página de error 404**.

Para esto necesitamos usar la expresión **pathMatch**:

pathMatch: Indica que la ruta capturada corresponde al parámetro **pathMatch**, que es un parámetro dinámico en Vue Router.

- **(.*)*** : Este patrón de coincidencia es **una expresión regular** que captura cualquier ruta que no haya sido manejada por las rutas definidas anteriormente.
- **(.*)** : Captura cualquier secuencia de caracteres, incluyendo barras /, que esté presente en la ruta.


- * : Indica que este patrón puede aparecer cero o más veces, lo que significa que puede coincidir con cualquier cantidad de caracteres en la ruta después de la parte capturada por (.*)).

Este arreglo de rutas es esencial para la navegación dentro de nuestra aplicación Vue, ya que define **qué componente se debe mostrar en cada ruta específica**.

Por lo tanto, La ruta **/:pathMatch(.*)*** capturará todas las rutas que no coincidan con **/, /ingresar, o /ver**, y mostrará el **componente notFound** para manejar estas **rutas no encontradas**.

Para completar la configuración del archivo y asegurarnos de que todo funcione correctamente, es importante seguir con los pasos que faltan.

Siguiendo con la **configuración del sistema de enrutamiento**:



```

1 // Instanciamos el Router con las routes.
2 // Crear enrutador
3 const router = VueRouter.createRouter({
4   history: VueRouter.createWebHistory(),
5   routes
6 });
7
8 // Crear la aplicación Vue
9 const app = Vue.createApp({});
10
11 // Montar el enrutador en la aplicación Vue
12 app.use(router);
13
14 app.component('compIngresar', compIngresar); // Registra el
15   componente CompIngresar
16 app.component('compMostrar', compMostrar); // Registra el
17   componente CompMostrar
18
19 // Montar la aplicación en el elemento con id="app"
20 app.mount('#contenedor');
  
```

El constante router nos permite **crear el enrutador** (router) usando la función **VueRouter.createRouter**. Esta función es proporcionada por **Vue Router** y nos permite configurar las rutas y el comportamiento de la navegación en nuestra aplicación.

Al crear el enrutador, es importante configurar el modo de historial que deseamos utilizar. En este caso, estamos utilizando **createWebHistory()** como **método para configurar el modo de historial en Vue Router**. Este método configura el enrutador para

utilizar la **API de historial HTML5**, la cual proporciona una experiencia de navegación más moderna y amigable al usuario.

Esto nos permite usar características como la **navegación hacia adelante y hacia atrás en la barra de navegación del navegador**, la capacidad de compartir enlaces directos a rutas específicas en nuestra aplicación, y una gestión más eficiente de las URL y sus parámetros.

Después de crear el enrutador, debemos **montarlo en nuestra aplicación Vue utilizando `app.use(router)`**. Esto permite que **nuestra aplicación utilice las rutas definidas en el enrutador y renderice los componentes correspondientes según la URL actual**.

Además, **registramos los componentes `compIngresar` y `compMostrar` en la aplicación Vue utilizando `app.component`**, lo cual es necesario para que **Vue reconozca y utilice estos componentes** en nuestra aplicación.

Finalmente, **montamos la aplicación Vue en el elemento HTML con la clase `contenedor` utilizando `app.mount('.contenedor')`**, lo que inicia la aplicación Vue y la conecta con el DOM, permitiendo que Vue controle la renderización y funcionalidad de nuestra aplicación dentro de ese elemento.

Con estos pasos completos, hemos configurado correctamente nuestro archivo para utilizar Vue Router y los componentes en nuestra aplicación Vue de manera efectiva.

La configuración detallada de rutas en Vue.js desempeña un papel fundamental en la estructura y funcionalidad de nuestra aplicación. Al definir claramente las relaciones entre las rutas y los componentes asociados, logramos una navegación fluida y coherente para los usuarios.

Para entender un poco mejor cómo se hace esta configuración veamos con la consola y el navegador como se renderizan estos componentes dentro de **<router-view>** en nuestro index:

Ejemplo Router

Ruta actual: /



Cuando el usuario haga click en Ingresar, se mostrará el formulario desarrollado en el archivo **ingresar.js** para que el usuario cargue sus películas. Este archivo, tiene su propio template, methods, etc .

Ejemplo Router

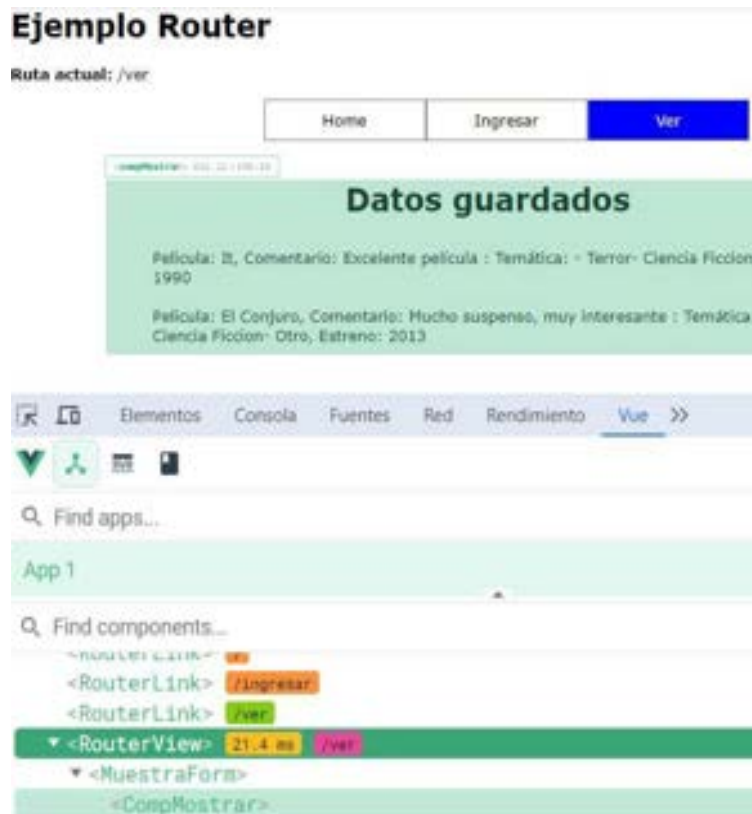
Ruta actual: /ingresar



Estos datos se almacenan por medio de localStorage, cuando ejecute la función de guardado, y luego se **redireccionará al usuario hacia la vista Ver**, esto lo haremos gracias a Vue-router, por medio de esta sintaxis: **this.\$router.push("/ver")**

- **\$router** : Es una instancia de Vue Router que se proporciona automáticamente cuando usamos Vue.js junto con Vue Router. **Representa el enrutador** de la aplicación Vue y proporciona métodos para navegar entre las diferentes rutas definidas en Vue Router.
- **push** : Es un método proporcionado por Vue Router que **permite cambiar la ruta actual del enrutador y navegar a una nueva ruta**.
- Al usar **this.\$router.push("/ver")**, estamos indicando al enrutador de Vue Router que navegue a la ruta especificada, en este caso, la ruta **"/ver"**.

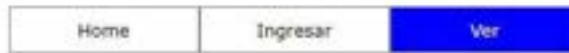
Finalmente cuando el usuario haga click en **ver**, se mostrarán los datos que fueron cargando desde el formulario de **películas**, obteniendo esta información en el ciclo de vida mounted. Tiene el mismo comportamiento si viene por la redirección de la vista ingresar.



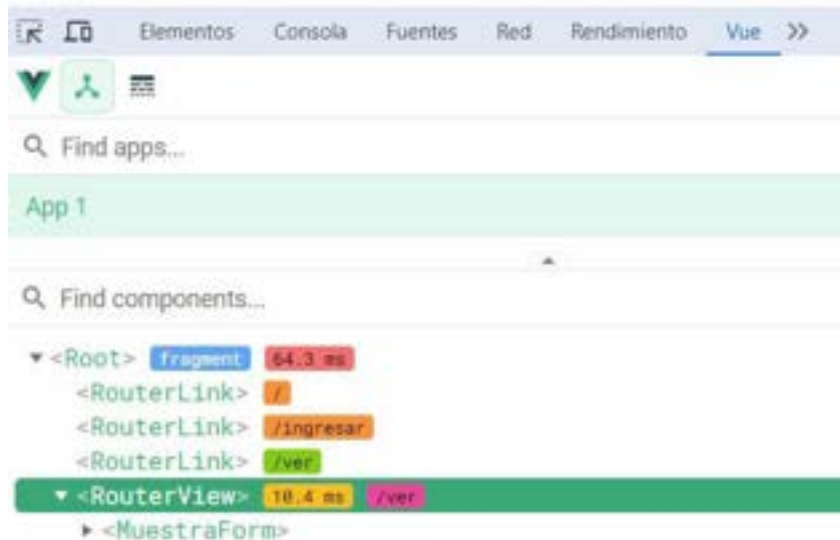
En el caso que no hubieran datos almacenados en localStorage, le mostraremos un **mensaje para incentivar al usuario** a iniciar la carga de sus películas.

Ejemplo Router

Ruta actual: /ver



Todavía no cargaste tus películas! Empezá ya!



Componente Transition

Vue ofrece dos componentes integrados que pueden ayudar a trabajar con transiciones y animaciones en respuesta a cambios de estado:

<transition> para aplicar **animaciones** cuando un elemento o componente **entra y sale del DOM**.

<transition-group> para aplicar **animaciones** cuando un elemento o componente **se inserta, se elimina o se mueve dentro de una lista v-for**.

Aparte de estos dos componentes, también podemos aplicar animaciones en Vue utilizando otras técnicas como alternar clases CSS o animaciones basadas en estado a través de enlaces de estilo.

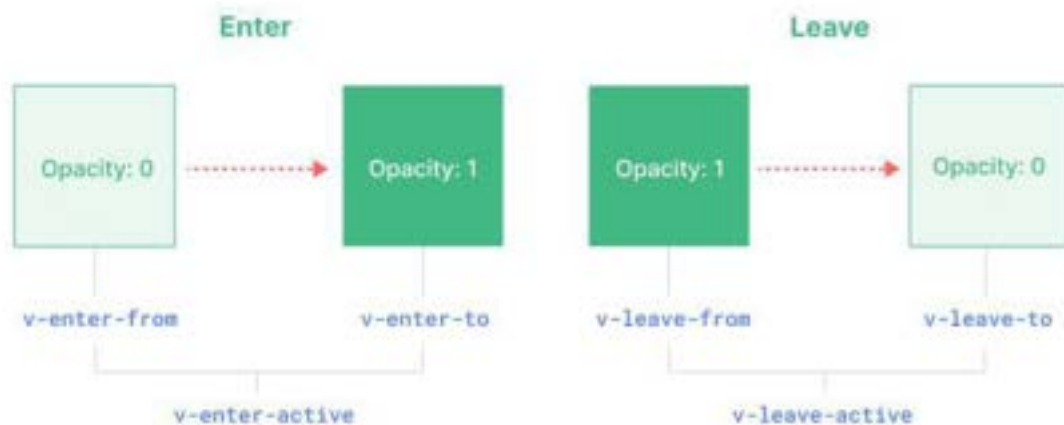
El componente **<transition>** es un componente integrado, lo que significa que está disponible en cualquier template de componente sin necesidad de registrarlo. La entrada o salida puede ser activada por uno de los siguientes métodos:

- Renderizado condicional mediante v-if.
- Visualización condicional mediante v-show.
- Alternancia de componentes dinámicos mediante el elemento especial <component>.
- Cambio del atributo especial key.

Hay que tener presente que si el contenido es un componente, este componente debe tener sólo un único elemento raíz.

Cuando un elemento dentro de un componente **<transition>** es insertado o eliminado, Vue automáticamente detectará si el elemento objetivo tiene transiciones o animaciones CSS aplicadas. Si las tiene, se agregarán/eliminarán una serie de clases de transición CSS en los momentos adecuados.

Vamos a centrarnos en las **6 clases CSS que Vue nos ofrece**. Las 3 primeras establecen clases para la entrada de los elementos y las 3 siguientes aplican clases para la salida de los elementos:



- **v-enter-from: Estado inicial para la entrada.** Se agrega antes de que se inserte el elemento, se elimina un fotograma después de que se inserta el elemento.
- **v-enter-active: Estado activo para la entrada.** Se aplica durante toda la fase de entrada. Se agrega antes de que se inserte el elemento, se elimina cuando finaliza la transición/animación. Esta clase se puede usar para definir la duración, el retraso y la curva de aceleración para la transición de entrada.
- **v-enter-to: Estado final para la entrada.** Se agrega un fotograma después de que se inserta el elemento (al mismo tiempo que se elimina v-enter-from), se elimina cuando finaliza la transición/animación.
- **v-leave-from: Estado inicial para la salida.** Se agrega inmediatamente cuando se activa una transición de salida, se elimina después de un fotograma.
- **v-leave-active: Estado activo para la salida.** Se aplica durante toda la fase de salida. Se agrega inmediatamente cuando se activa una transición de salida, se elimina cuando finaliza la transición/animación. Esta clase se puede usar para definir la duración, el retraso y la curva de aceleración para la transición de salida.
- **v-leave-to: Estado final para la salida.** Se agrega un fotograma después de que se activa una transición de salida (al mismo tiempo que se elimina v-leave-from), se elimina cuando finaliza la transición/animación.

Las clases `v-enter-active` y `v-leave-active` nos dan la capacidad de especificar diferentes curvas de aceleración para transiciones de entrada / salida.

```
1 .v-enter-active,  
2 .v-leave-active {  
3   transition: opacity 0.7s ease, background-color 0.5s;  
4 }  
5 .v-enter-from,  
6 .v-leave-to {  
7   opacity: 0;  
8   background-color: rgba(181, 33, 33, 0);  
9 }
```

Podemos aplicar algunas clases para ejemplificar de forma simple el uso de estos elementos aplicado a la renderización condicional:



Cuando el usuario haga click al botón dependiendo del valor de la variable, mostraremos u ocultaremos un mensaje. Además el texto del botón también cambiará:



```
1 <button @click="show = !show" >
2 {{ show ? ' Ocultar Mensaje' : ' Mostrar Mensaje' }}
3 </button>
4 <transition>
5   <div class="cont" v-if="show">
6     <p class="mensaje">
7       <span> Acá está tu sorpresa </span>
8     </p>
9   </div>
10 </transition>
```

Explicuemos cómo funciona nuestro ejemplo:

- **<button @click="show = !show" >{{ show ? ' Ocultar Mensaje' : ' Mostrar Mensaje' }}</button>**
 - El botón se utiliza para mostrar u ocultar el mensaje. El texto del botón cambia dinámicamente según el estado de la variable show.
 - **@click="show = !show"** Al hacer click en el botón, se cambiará el valor de show de verdadero a falso o viceversa.
 - **{{ show ? ' Ocultar Mensaje' : ' Mostrar Mensaje' }}** es una expresión que muestra el texto "Ocultar Mensaje" cuando show es verdadero y "Mostrar Mensaje" cuando show es falso.
- **El componente <transition>** envuelve el contenido que queremos que tenga un efecto de transición al aparecer o desaparecer. Aplicará las clases definidas en el CSS, para la entrada y salida del mensaje.
- **<div if="show"></div>** Contenedor que engloba el mensaje a mostrar u ocultar. Sólo se muestra si el valor de show es verdadero

Puede darse el caso que queramos usar distintas transiciones para diferentes parte del código, en ese caso Vue nos permite otorgarles un **nombre** para poder referenciarlas desde CSS y en nuestro componente **<transition>**.

Si lo aplicamos al ejemplo anterior, nuestro HTML tendrá en el componente **transition** el **atributo name** y su **valor**, será el

nombre que le daremos a ese efecto:

```
index.html
1 <button @click="show = !show">
2 {{ show ? '#128070; Ocultar Mensaje' : '#x1f447; Mostrar Mensaje' }}
3 </button>
4 <transition name="fade">
5   <div class="cont" v-if="show">
6     <p class="mensaje">
7       <span>#x1f47b;</span> Acá está tu sorpresa <span>#x1f47d;</span>
8     </p>
9   </div>
10 </transition>
```

Y desde nuestro CSS deberíamos reemplazar la el prefijo **v-** por el nombre otorgado **"fade"**:

```
estilo.css
1 .fade-enter-active,
2 .fade-leave-active {
3   transition: opacity 0.7s ease, background-color 0.5s;
4 }
5 .fade-enter-from,
6 .fade-leave-to {
7   opacity: 0;
8   background-color: rgba(181, 33, 33, 0);
9 }
```

Hay que tener en cuenta, que **las animaciones se aplican de la misma forma que las transiciones CSS**. La única diferencia es que **-enter-from** no se eliminan inmediatamente después de insertar el elemento, sino en el evento **anmationend**.

Por ejemplo, si quisiéramos hacer **una animación con efecto de pulsación** nuestro CSS tendría que tener la creación de la animación y su respectiva asociación a las clases de vue para la entrada activa y la salida activa:

```
estilo.css

1 .animacion-enter-active {
2   animation: animacion 1s;
3 }
4 .animacion-leave-active {
5   animation: animacion 1s reverse;
6 }
7 @keyframes animacion {
8   0%, 50%, 100% {opacity: 1;}
9   25%, 75% {opacity: 0;}
10 }
```

En el index, debemos colocarle su respectivo nombre como valor del name:

```
index.html

1 <transition name="animacion">
2   <!--Código a aplicar la animación-->
3 </transition>
```

Más adelante seguiremos profundizando con animaciones. Siempre hay que tener criterio al aplicarlas, fundamentalmente se usan para acompañar acciones que el usuario pueda llegar a realizar en la interfaz para notificarle cambios de estado a nivel visual.

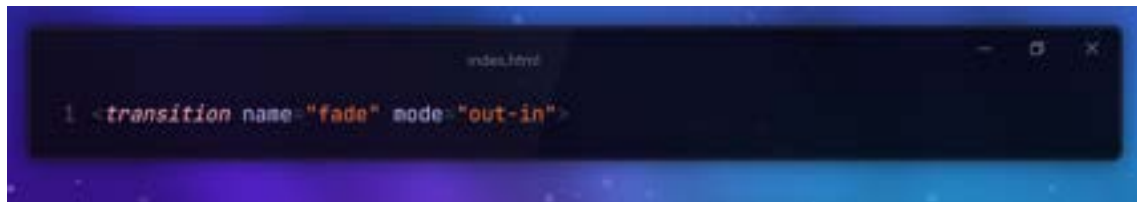
Transitions: modos de transición

Cuando Vue aplica efectos de animación para la entrada y salida de elementos en el DOM utilizando el componente **<transition>**, estos tienen un comportamiento predeterminado establecido con **position:absolute**, diseñado para cuando **ambos elementos están presentes en el DOM**.

Sin embargo, hay casos en los que este comportamiento predeterminado no es el deseado. Es probable que queramos que el **elemento saliente se anime primero y que el elemento entrante se inserte sólo después de que haya finalizado la animación de salida**. Realizar esta animación manualmente suele ser complejo.

Por este motivo, Vue ha habilitado un comportamiento específico para hacer que la animación sea más fluida y para evitar que estos elementos se muestren casi simultáneamente.

El componente **<transition>** tiene una propiedad llamada **mode**, cuyo valor podemos establecer en **"out-in"**. Este modo es el más utilizado para lograr el efecto deseado.



Si queremos un **comportamiento inverso, donde el elemento entrante se anime primero antes de que el elemento saliente se elimine**, el valor de **mode** sería **"in-out"**. Sin embargo, este modo se utiliza con menos frecuencia.

mode = "in-out"	mode = "out-in"
Ambos elementos se muestran en el cambio de la vista. Produce un salto visual, hasta que se retira el texto "Ingresar dato"	Produce una animación fluida sin saltos.

<ul style="list-style-type: none"> • Ingresar Dato • <input type="text"/> • Otro vínculo sin setear <p>Ingresar dato</p> <p>Datos guardados</p>	<ul style="list-style-type: none"> • Ingresar Dato • <input type="text"/> • Otro vínculo sin setear <p>Datos guardados</p>
---	--

Transition-group

El componente **<transition-group>** está diseñado para animar la inserción, eliminación y cambio de orden de elementos o componentes que se representan en una lista.

Además hay que establecer que hay ciertas diferencias con el componente **<transition>**:

- No renderiza por defecto un elemento contenedor, pero se puede especificar con la propiedad **tag**.
- No admite modos de transición debido a que no alterna entre elementos mutuamente exclusivos.
- Los elementos dentro de **<transition-group>** deben tener un atributo **key** único.
- Las clases de transición CSS se aplican a elementos individuales en la lista, no al grupo/container en sí.

Por ejemplo, si el usuario ingresara en un formulario un registro de películas para ver o almacenar reseñas, cada vez que envíe el formulario se crearía un objeto con las propiedades asignadas a cada **v-model** de los controles del formulario. Por medio del método **push** insertaríamos en un array.

El componente **transition-group** sería el responsable de crear el elemento **** por medio del atributo **tag**. Le otorgamos un **name** a la transición.

Y sobre el **list-item**, por medio de la directiva **v-for** recorreremos de cada ítem de la iteración las propiedades que le vamos a mostrar al usuario por medio de las interpolaciones.

Cada **list item** que se vaya creando se irá mostrando en la vista con el efecto de transición deseado. En cuyo caso, veamos cómo se vería esto en el código:


```
index.html
1 <transition-group name="fade" tag="ul">
2   <li v-for="item in arr" :key="item">
3     {{ item.titulo }} , {{ item.comentario }} -
4     <span v-for="x in item.selected">{{ x }}</span>
5     {{ item.estreno }}
6   </li>
7 </transition-group>
```

Además podríamos agregar un botón que nos permita borrar cada elemento. Por lo tanto, cuando se borre algún ítem, también aplicaría el efecto fade, al retirar ese list-item del DOM.

Tener presente que las funciones de guardar, resetForm de los v-model de cada control del formulario y eliminarItem, deben estar desarrolladas en la lógica.

Por ejemplo:

```
logica.js
1 methods: {
2   guardar: function(form_data) {
3     this.arr.push(Object.assign({}, this.form_data)); // Clonar el objeto
4     // antes de agregarlo a this.arr
5     this.resetForm(); // Restablecer el formulario después de agregar los
6     // datos
7   },
8   resetForm: function() {
9     this.form_data = {
10       titulo: '',
11       comentario: '',
12       selected: [],
13       estreno: null
14     };
15   },
16   eliminarItem: function(index) {
17     this.arr.splice(index, 1); // Eliminar el elemento en la posición index de
18     // this.arr
19   }
20 }
```

Aplicar transition a router-view

Es común que deseemos mantener una consistencia visual al mostrar los componentes utilizando **<router-view>**. Esto permite que el usuario comprenda el comportamiento del sistema de manera más predecible y evita sorpresas con transiciones inesperadas.

Por esta razón, es habitual implementar **transiciones que se apliquen de manera uniforme tanto en la entrada como en la salida de los componentes a medida que el usuario navega por el sitio**. Esto ayuda a crear una experiencia de usuario más coherente y agradable, facilitando la comprensión de los cambios en la interfaz.



Veamos qué es lo que hace este código:

- **v-slot="{ Component }"** : Es una directiva de Vue que permite acceder al componente que se está renderizando en ese momento dentro de **<router-view>**. Al usar **{ Component }**, estamos extrayendo el componente actual que se está mostrando en la ruta.
- **<transition name="rutas"></transition>**:
 - **<transition>** : Envuelve al componente que se está mostrando en la ruta.
 - **name="rutas"**: Indica que se aplicará un efecto de transición llamado "rutas".
- **<component :is="Component" />**:
 - Se usa el componente dinámico de Vue para renderizar dinámicamente el componente actual de la ruta.
 - **:is="Component"**: Indica que el componente a renderizar será el que esté almacenado en la variable Component.

Esta forma de aplicación en las rutas es necesaria para mantener transiciones consistentes y significativas en **<router-view>** para proporcionar una experiencia de usuario coherente y comprensible. Aspectos clave a considerar incluyen la duración y efectos de las transiciones, su adaptación a dispositivos y la retroalimentación visual que ofrecen.

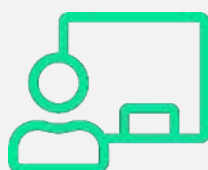
Más adelante exploraremos estos aspectos con mayor detalle para optimizar la experiencia de navegación del usuario.

¡Practiquemos el enrutamiento y transiciones con CSS!



Hemos llegado así al final de esta clase en la que vimos:

- Concepto de Enrutamiento.
- Vue-router
- Componentes RouterLink y RouterView.
- Configuración Sistema de enrutamiento.
- Configurar página de error.
- Componente transition.
- Transitions modo de transition.
- Transition-group.
- Aplicar transition a router-view.



Te esperamos en la **clase en vivo** de esta semana.
No olvides realizar el **desafío semanal**.

¡Hasta la próxima clase!

Bibliografía

Vue-router. (s. f.). Guía. En Documentación Oficial de Vue-router. Recuperado de <https://router.vuejs.org/guide/>

Vue.js. (s. f.). Componente transition. En Documentación Oficial de Vue.js. Recuperado de <https://vuejs.org/guide/built-ins/transition/>

Vue-router. (s. f.). Guía avanzada en transitions. En Documentación Oficial de Vue-router. Recuperado de <https://router.vuejs.org/guide/advanced/transitions.html/>