

EXERCISES QUE TRABAJAREMOS EN EL CUE:

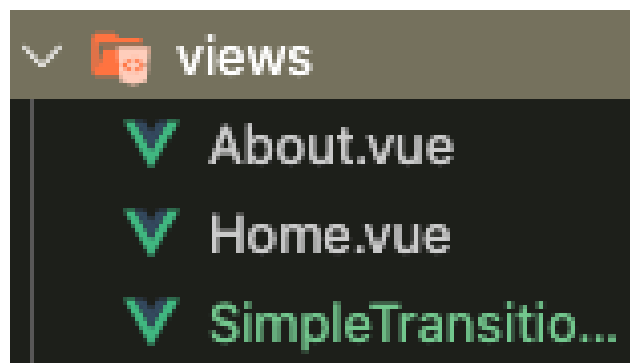
- EXERCISE 1: TRANSICIÓN A ELEMENTOS HTML.
- EXERCISE 2: ANIMACIONES.
- EXERCISE 3: TRANSICIÓN CON LIBRERÍA EXTERNA.
- EXERCISE 4: RUTAS LAZY-LOADING.

EXERCISE 1: TRANSICIÓN A ELEMENTOS HTML.

Para empezar a poner en práctica el concepto de transiciones, crearemos un nuevo proyecto al que se le debe agregar router en la instalación.

Cuando agregamos u ocultamos elementos en el DOM, esto provoca que el cambio sea brusco al renderizar nuevamente la sección. Para que sea más sutil, Vue provee la forma de poder agregar estilos CSS, cuando el elemento entra y/o sale del DOM.

Vamos a crear una vista en la carpeta “views”, llamada “simpleTrasition.vue”.



TEMPLATE

Comenzaremos por la sección de template.

```
1 <template>
2   <div>
3     <h1>Transitions</h1>
4     <button @click="show = !show">Mostrar/Ocultar</button>
5     <transition name="fade">
6       <div v-if="show">
7         <h2>Mi titulo</h2>
8         <p>Lorem ipsum dolor sit amet consectetur adipisicing elit.
9 Possimus necessitatibus voluptatem accusantium nemo iusto qui,
10 quidem beatae deleniti magni ipsam, adipisci, harum magnam atque
11 quia repellat ratione. Aperiam, fuga maxime. Laudantium hic ipsam
12 eveniet temporibus distinctio maiores, sit deleniti architecto
13 vero. Doloribus sit, iusto quidem ipsam, beatae porro ratione minima
14 quas numquam provident fugit autem exercitationem corporis
15 praesentium, error sint.</p>
16       </div>
17     </transition>
18   </div>
19 </template>
```

El botón “Mostrar/ocultar”, nos va a servir para cambiar de valor el dato “show”, y éste, al tener la negación al momento de asignarse, podrá cambiar el valor booleano por cada clic.

Usaremos la etiqueta **<transition>**, para envolver al div que tiene el condicional **v-if="show"**, el cual mostrará el contenido cuando show sea igual a “true”.

Lo importante en la etiqueta transition es asignar un nombre, para luego poder utilizar las clases CSS.

SCRIPT

En el objeto data, agregaremos la variable asignada en el template “show”, y partirá con valor “false” al crear el componente.

```
1 <script>
2 export default {
3   name: 'Simple-transition',
4   data: function() {
```

```
5     return {  
6         show:false,  
7     }  
8 },  
9 }  
10 </script>
```

CSS

En la sección de style, vamos a emplear el nombre asignado a la transición, combinado con las clases de vue.

Utilizaremos:

- v-enter
- v-leave-to
- v-enter-active
- v-leave-active

```
1 .fade-enter-active, .fade-leave-active{  
2     transition: opacity 0.5s;  
3 }  
4  
5 .fade-enter, .fade-leave-to{  
6     opacity:0;  
7 }
```

Ahora, fade-enter y fade-leave-to, nos servirán para setear la opacidad en 0 en el primer frame antes de entrar el componente, y en el último frame al salir el componente.

Luego, en fade-enter-active y fade-leave-active, vamos a ocupar la propiedad CSS transition para darle un tipo de duración; de pasar de opacity 0 a 1 cuando el elemento entre, y de 1 a 0 cuando éste sale.

ROUTER LAS VISTAS:

Vamos a importar las SFC, creando en la vista:

```
1 import SimpleTransition from '@/views/SimpleTransition.vue'
```

Dentro del Array routes, vamos a agregar el path y el componente:

```
1 {  
2   path: '/simple',  
3   name: "Simple",  
4   component: SimpleTransition  
5 },
```

Archivo index.js final:

```
1 import Vue from 'vue'  
2 import VueRouter from 'vue-router'  
3 import Home from '../views/Home.vue'  
4 import SimpleTransition from '@/views/SimpleTransition.vue'  
5 Vue.use(VueRouter)  
6  
7  
8 const routes = [  
9   {  
10    path: '/',  
11    name: 'Home',  
12    component: Home  
13  },  
14  {  
15    path: '/simple',  
16    name: "Simple",  
17    component: SimpleTransition  
18  },  
19  {  
20    path: '/about',  
21    name: 'About',  
22    // route level code-splitting  
23    // this generates a separate chunk (about.[hash].js) for this  
24    route  
25    // which is lazy-loaded when the route is visited.
```

```
26     component: () => import(/* webpackChunkName: "about" */
27     '../views/About.vue')
28   }
29 ]
30
31 const router = new VueRouter({
32   mode: 'history',
33   base: process.env.BASE_URL,
34   routes
35 })
36
37 export default router
```

AGREGANDO VISTA A NAV DE APP.VUE

En el archivo App.vue, vamos a agregar el link de transición, le llamaremos /simple:

```
1 <div id="nav">
2   <router-link to="/">Home</router-link> |
3   <router-link to="/simple">Simple</router-link> |
4   <router-link to="/about">About</router-link>
5 </div>
```

Si observamos la aplicación levantada en el navegador, deberíamos ver:

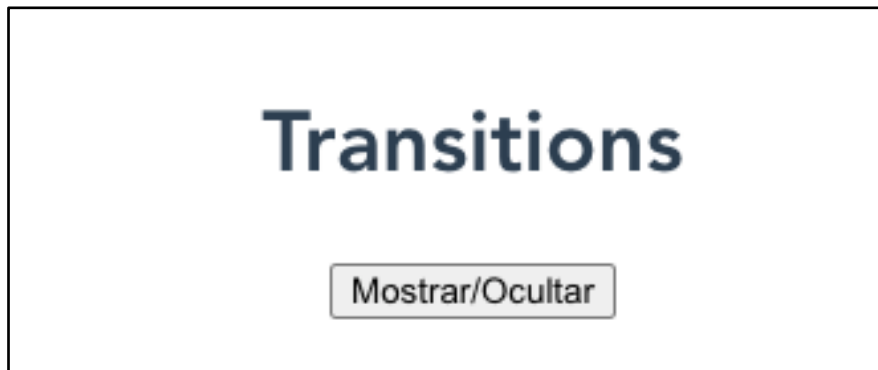
[Home](#) | [Simple](#) | [About](#)



Welcome to Your Vue.js App

For a guide and recipes on how to configure / customize this project,
check out the [vue-cli documentation](#).

Luego, si vamos al enlace “Simple”:



Al momento de hacer clic, se mostrará el contenido con una transición de entrada y de salida.



EXERCISE 2: ANIMACIONES.

Para realizar el siguiente ejercicio, continuaremos trabajando en el proyecto del anterior.

Cuando utilizamos vue-router, al momento de cambiar de vista, podemos agregar animaciones envolviendo con la etiqueta `<transition>` `<router-view>`, así, cada vez que el usuario cambie la vista, la transición se aplica.

Vamos a abrir el archivo principal App.vue, y a envolver router-view.

```
1 <template>
2   <div id="app">
3     <div id="nav">
4       <router-link to="/">Home</router-link> |
5       <router-link to="/simple">Simple</router-link> |
6       <router-link to="/about">About</router-link>
7     </div>
8     <transition name="router-anim" mode="out-in">
9       <router-view/>
10    </transition>
11  </div>
12 </template>
```

Le asignaremos el nombre de la transición "router-anim", y el mode="out-in".

Modo out-in: el elemento, en este caso la ruta actual, realiza su transición de salida, y cuando haya terminado, el nuevo elemento realiza su transición de entrada.

AGREGANDO ANIMACION:

Dentro de la etiqueta `<style>` del componente App.vue, vamos a agregar los siguientes estilos:

```
1 .router-anim-enter-active{
2   animation: coming 1s;
3   animation-delay: .5s;
4   opacity:0;
5 }
6
7 .router-anim-leave-active{
8   animation: going 1s;
9 }
10
```

```
11 @keyframes going{
12     0%{
13         transform: translateX(0)
14     }
15     100%{
16         transform: translateX(-50px);
17         opacity:0;
18     }
19 }
20
21 @keyframes coming{
22     from{
23         transform: translateX(-50px);
24         opacity:0;
25     }
26     to{
27         transform: translateX(0px);
28         opacity:1;
29     }
30 }
31 }
```

Las animaciones going y coming, serán las encargadas de realizar las acciones de entrada y salida, es por ello que solo utilizaremos las clases “enter-active y leave-active”, y en ellas agregaremos las animaciones.

En este caso de entrada, la vista partirá -50px hacia la izquierda con opacidad 0 (no visible), al terminar el estado de entrada, éste quedará en su posición original y con opacidad 1 (visible).

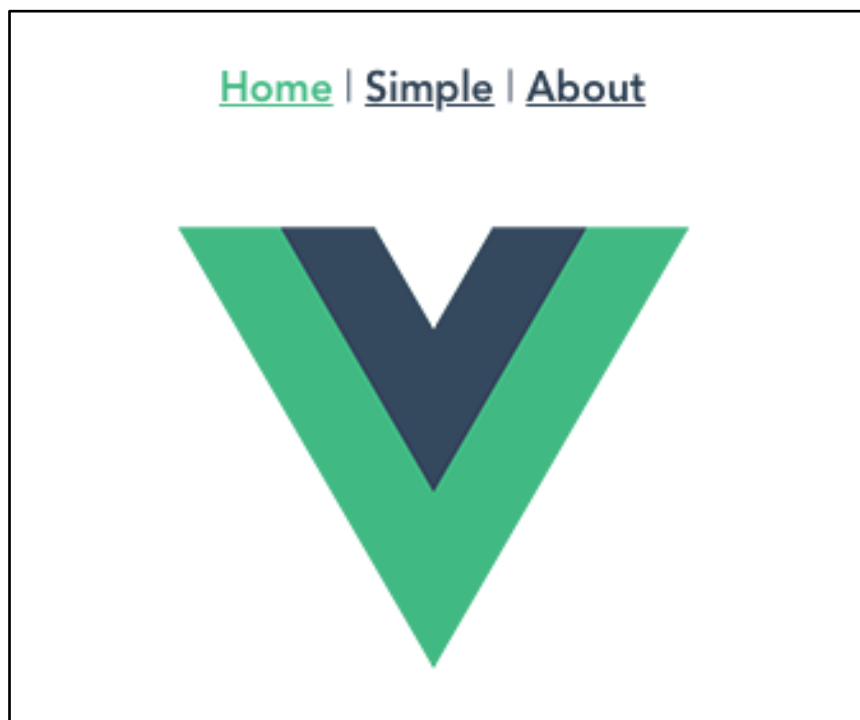
La animación “going” partirá el estado de salida, en la posición original, y terminará -50px hacia la izquierda con opacidad 0 (no visible).

Archivo App.vue completo:

```
1 <template>
2   <div id="app">
3     <div id="nav">
4       <router-link to="/">Home</router-link> |
5       <router-link to="/simple">Simple</router-link> |
6       <router-link to="/about">About</router-link>
7     </div>
8
9     <transition name="router-anim" mode="out-in">
10      <router-view/>
11    </transition>
12
13  </div>
14 </template>
15
16 <style>
17
18 #app {
19   font-family: Avenir, Helvetica, Arial, sans-serif;
20   -webkit-font-smoothing: antialiased;
21   -moz-osx-font-smoothing: grayscale;
22   text-align: center;
23   color: #2c3e50;
24 }
25
26 #nav {
27   padding: 30px;
28 }
29
30 #nav a {
31   font-weight: bold;
32   color: #2c3e50;
33 }
34
35 #nav a.router-link-exact-active {
36   color: #42b983;
37 }
38
39 .router-anim-enter-active{
40   animation: coming 1s;
41   animation-delay: .5s;
42   opacity:0;
43 }
44
45 .router-anim-leave-active{
46   animation: going 1s;
47 }
48
49 @keyframes going{
```

```
50 0%{
51   transform: translateX(0)
52 }
53 100%{
54   transform: translateX(-50px);
55   opacity:0;
56 }
57 }
58
59 @keyframes coming{
60   from{
61     transform: translateX(-50px);
62     opacity:0;
63   }
64   to{
65     transform: translateX(0px);
66     opacity:1;
67   }
68 }
69 </style>
```

Si guardamos y vamos cambiando de rutas, podemos observar como la transición por cada vista se va aplicando.



EXERCISE 3: TRANSICIÓN CON LIBRERÍA EXTERNA.

Para realizar el siguiente ejercicio, continuaremos con el proyecto que hemos utilizado hasta ahora.

Como vimos anteriormente, para poder aplicar una transición, debemos emplear estilos CSS para dar efectos. Es aquí donde se hace uso de librerías externas para realizar el trabajo. En este ejercicio, revisaremos cómo aplicar la librería animate.css.

CDN ANÍMATE:

<https://cdn.jsdelivr.net/npm/animate.css@3.5.1>

Primero, crearemos un nuevo archivo en la carpeta "views", llamado "LibraryTransition.vue", y en este ejemplificaremos el uso de animate css.

Ahora, vamos a escribir la sección de template.

```
1 <template>
2   <div>
3     <h1>Animate css</h1>
4     <div class="container">
5       <div class="tabs">
6         <div><button @click="show(1) "
7 class="tab">Info1</button></div>
8         <div><button @click="show(2) "
9 class="tab">Info2</button></div>
10      </div>
11      <div class="panel">
12        <transition name="panel-anim" mode="out-in" enter-
13 active-class="animated fadeIn" leave-active-class="animated
14 fadeOut" >
15          <div key="info1" v-if="activeTab ===1">
16            <h1>Info1</h1>
17
18            <p>Lorem ipsum dolor sit amet consectetur
19 adipiscing elit. Voluptatum ut corporis, non quisquam
20 asperiores officiis, deserunt impedit omnis a dolor excepturi
21 provident, temporibus dolores labore optio id numquam
22 consequatur sint!
23            Voluptatem quisquam minus, optio id nihil, minima
24 doloremque assumenda repellendus aliquid quia, sequi ea quasi
25 non quo veniam quas sunt. Ab error cupiditate nihil? Velit ea in
26 doloribus reprehenderit optio?
27            Rerum labore soluta quis cumque, tempora sapiente
28 dolor. Qui architecto, ab dignissimos doloribus, omnis, officia
```

```

29 natus laboriosam a reiciendis quam recusandae rem aperiam
30 repellat. Exercitationem a commodi totam sequi facere.</p>
31 </div>
32 <div key="info2" v-if="activeTab ===2">
33   <h1>Info2</h1>
34   
36 </div>
37 </transition>
38 </div>
39 </div>
40 </div>
41 </template>
  
```

Esta sección es un panel de tabs, al cual le aplicaremos transición cada vez que el usuario cambie de tab.

Las partes principales son: “tabs” y “panel”.

Será en “panel” donde aplicaremos las transition, utilizando las props: “enter-active-class” y “leave-active-class”.

SECCIÓN SCRIPT:

En ésta, vamos a crear en data un Array llamado “tabs”, el cual nos servirá para determinar el tab activo.

```

1 <script>
2 export default {
3   name: 'Library-animate',
4   // props: {},
5   data: function() {
6     return {
7       tabs:[
8         {id:1, tab:'info1', active:true},
9         {id:2, tab:'info2', active:false},
10      ]
11     }
12   },
13   computed: {
14     activeTab() {
15       let tab = this.tabs.find(tab=>tab.active === true);
16       return tab.id;
17     }
18   },
  
```

```
19  methods: {
20      show(tab_id) {
21          this.tabs.forEach(tab=>{
22              if(tab.id == tab_id){
23                  tab.active = true;
24              }
25              else{
26                  tab.active = false;
27              }
28          })
29      }
30  },
31  // components: {},
32 }
33 </script>
```

Para saber que tab está activo actualmente, crearemos una propiedad computada llamada "activeTab", la cual retorna la id del tab activo.

Y para poder hacer el cambio de tabs, será a través de un método llamado "show", que es el encargado de setear el tab activo dentro del Array.

SECCIÓN STYLE:

En primer lugar, aquí adjuntaremos la librería en cdn, con la palabra reservada @import, acompañada de las clases para .container, .tabs y .tab.

```
1  <style scoped>
2  @import "https://cdn.jsdelivr.net/npm/animate.css@3.5.1";
3  .container{
4      border: 1px solid black;
5      height:500px;
6      width:90%;
7      margin: 0 auto;
8      background: rgb(61,60,60);
9      color:white;
10 }
11 .tabs{
12     display:flex;
13     border: 1px solid green;
14     padding: 5px 0;
15     background: white;
16     color:rgb(61,60,60);
17 }
18 .tab{
19     width:100px;
20     border: 1px solid white;
```

```
21 background:green;
22 border-radius:10px;
23 padding:10px 0;
24 }
25 </style>
```

CREANDO RUTA PARA VISTA:

Ya teniendo creado el componente, es momento de crear su ruta para poder visitarlo, es por esto que iremos a la carpeta “router” ubicada en el archivo index.js, e importaremos la nueva vista y crearemos su path.

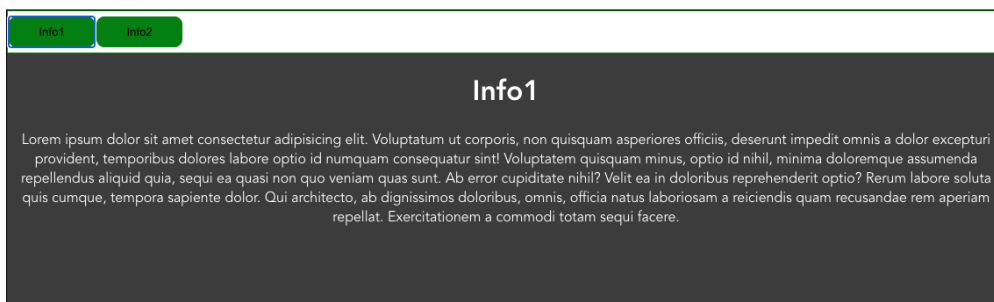
```
1 import LibraryTransition from '@/views/LibraryTransition.vue'
```

Dentro del Array routes, vamos a crear el siguiente objeto:

```
1 {
2   path: '/libreria',
3   name: 'Library',
4   component: LibraryTransition
5 },
```

Si guardamos nuestra aplicación, y visitamos la ruta /librería, veremos que al cambiar de tab, éste lo hace con los efectos agregados de la librería.

Animate css



EXERCISE 4: RUTAS LAZY-LOADING.

Para realizar el ejercicio, seguiremos con el proyecto que hemos estado utilizando. En específico, trabajaremos en el archivo `index.js` de la carpeta `router`.

Cuando desarrollamos `vue`, al compilar nuestra app para ser visualizada, éste empaqueta todos los archivos Javascript generados por los componentes, en un solo archivo que carga al iniciar nuestro sitio, llamado `app.js`.

Esto podría provocar una carga inicial lenta, si tuviéramos muchos componentes asociados a rutas.

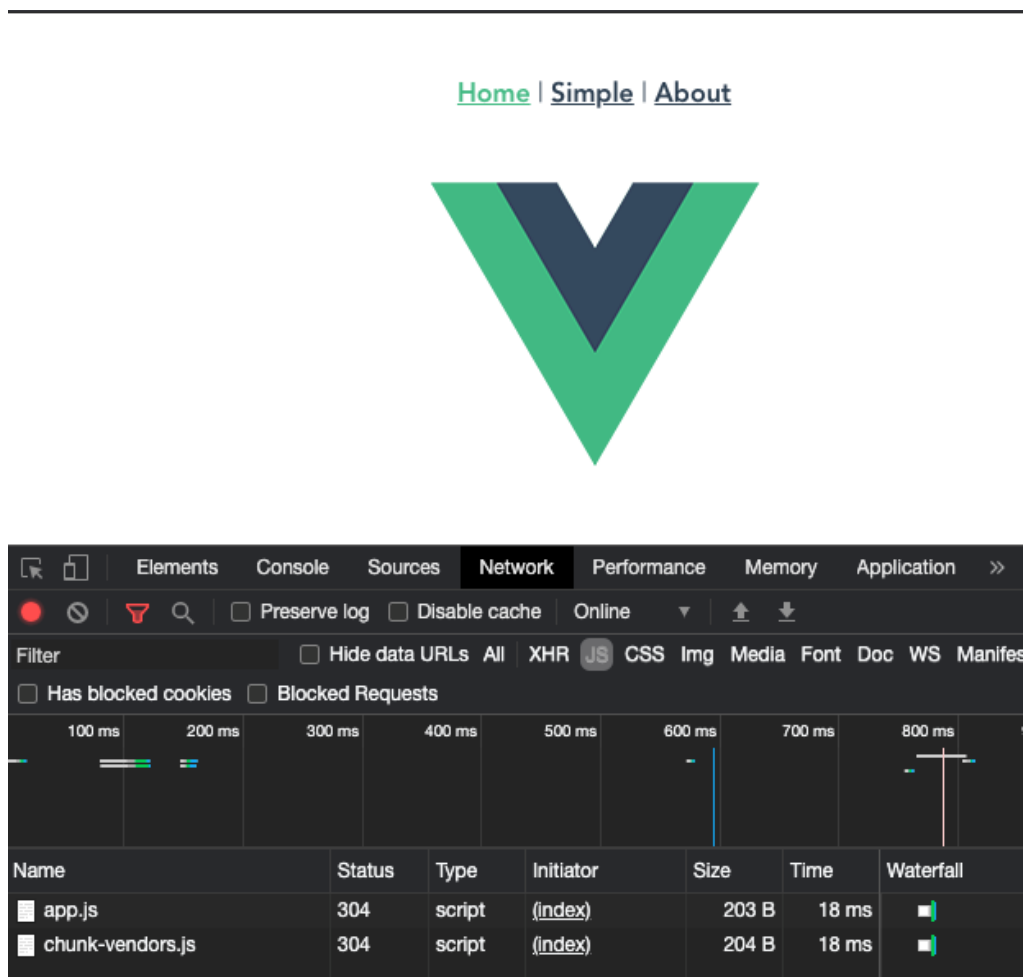
Es por esta razón, que `vue` provee una manera fácil de poder decidir si la carga de una ruta será cuando el usuario cargue el sitio, o cuando éste visite la ruta en específico del archivo.

Abriremos `index.js` de la carpeta `route`, y nos dirigiremos al path `"/about"`, ya que la ruta está utilizando `lazy-loading`.

```
1 {  
2   path: '/about',  
3   name: 'About',  
4   // route level code-splitting  
5   // this generates a separate chunk (about.[hash].js) for  
6   this route  
7   // which is lazy-loaded when the route is visited.  
8   component: () => import(/* webpackChunkName: "about" */  
9   '../views/About.vue'),  
10 }
```

Cuando visitamos `"/about"`, es en el momento que se carga un archivo llamado `about.js`.

Si levantamos nuestra aplicación, y abrimos el inspector del navegador, iremos a network y se verá así:



Al iniciar la página, carga el archivo app.js. Si visitamos `/about`, podremos ver que hace una carga llamando al archivo about.js



This is an about page

Name	Status	Type	Initiator	Size	Time	Watermark
app.js	304	script	(index)	203 B	18 ms	
chunk-vendors.js	304	script	(index)	204 B	18 ms	
about.js	200	script	app.js:919	(prefet...	11 ms	

APLICANDO LAZY-LOADING A LAS RUTAS CREADAS

```
1 import Vue from 'vue'
2 import VueRouter from 'vue-router'
3 import Home from '../views/Home.vue'
4
5 const ABOUT = () => import(/* webpackChunkName: "about" */
6   '../views/About.vue');
7 const SIMPLE = () => import(/* webpackChunkName: "simple" */
8   '@/views/SimpleTransition.vue')
9 const LIBRARY = () => import(/* webpackChunkName: "library" */
10  '@/views/LibraryTransition.vue')
11
12 Vue.use(VueRouter)
13
14 const routes = [
15   {
16     path: '/',
```



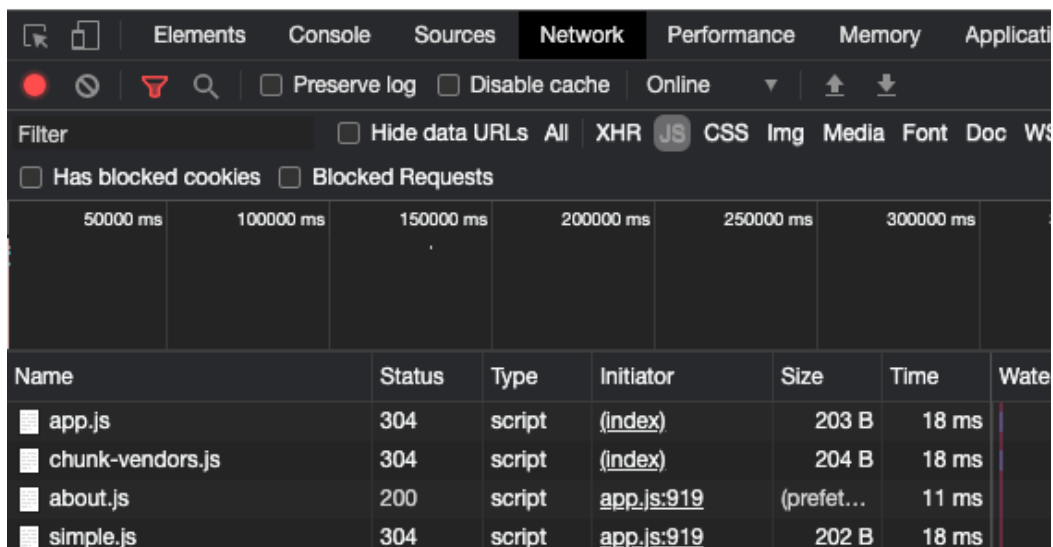
```
17     name: 'Home',
18     component: Home
19   },
20
21   {
22     path: '/simple',
23     name: "Simple",
24     component: SIMPLE,
25   },
26
27   {
28     path: '/libreria',
29     name: 'Library',
30     component: LIBRARY,
31   },
32
33   {
34     path: '/about',
35     name: 'About',
36     // route level code-splitting
37     // this generates a separate chunk (about.[hash].js) for
38     this route
39     // which is lazy-loaded when the route is visited.
40     component: ABOUT,
41   }
42 ]
43
44 const router = new VueRouter({
45   mode: 'history',
46   base: process.env.BASE_URL,
47   routes
48 })
49
50 export default router
```





Ya hemos dejado las rutas creadas en los ejercicios anteriores “lazy-loading”.

[Home](#) | [Simple](#) | [About](#)

Transitions

Mostrar/Ocultar



Timeline							
50000 ms		100000 ms		150000 ms		200000 ms	
Name		Status	Type	Initiator	Size	Time	Waterfall
 app.js		304	script	(index)	203 B	18 ms	
 chunk-vendors.js		304	script	(index)	204 B	18 ms	
 about.js		200	script	app.js:919	(prefet...	11 ms	
 simple.js		304	script	app.js:919	202 B	18 ms	