



Desarrollo web en entorno cliente





Tema 10. Vue Router

1. Vue Router

Una de las ventajas de Vue que se señaló en la primera unidad de este framework, es que facilita la creación de SPAs. Esto es, la aplicación tendrá una sola página (el ya conocido index.html) pero tendrá, además, una serie de vistas que el usuario final percibirá como páginas diferentes.

Para controlar la navegación entre las diferentes vistas Vue proporciona la librería Vue-router.

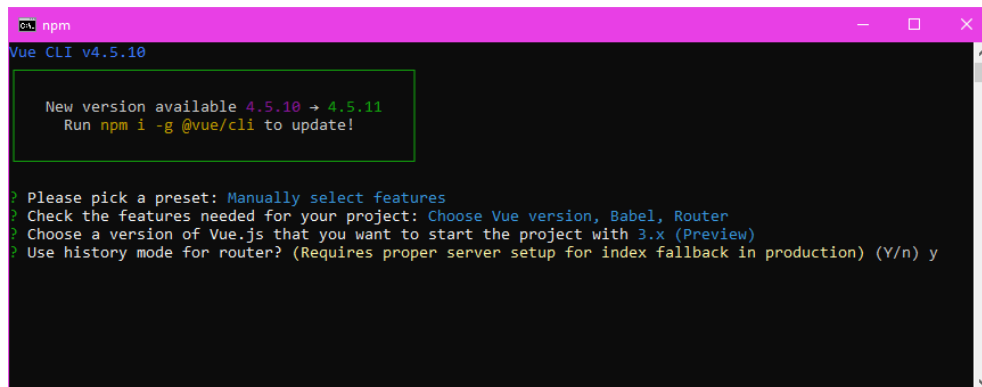
Básicamente, con vue-router, la estructura será la siguiente:

- En App.vue (aunque es posible cargarla en otro componente) se incluirá una etiqueta `<view-router>` en la que, vue-router, cargará un componente u otro en función de la ruta (ruta de la barra del navegador). Esa relación ruta-componente la indicará el desarrollador.
- La configuración de vue-router supone definir rutas que mapean componentes de la aplicación a rutas URL, o subrutas que mapeen subcomponentes dentro de otros.

2. Generar un proyecto con vue-router

Si se crea un proyecto desde 0 con vue CLI, se seleccionará la opción Router en el momento de seleccionar características, además de las ya conocidas “Choose Vue version” y Babel.

Tras la selección de la versión de Vue, el generador preguntará si se va a usar, o no, el modo history para router. Si no se selecciona, las urls incorporarán un # que ya se verá más adelante. En principio, indicamos que si al modo history:



```
npm
Vue CLI v4.5.10

New version available 4.5.10 → 4.5.11
Run npm i -g @vue/cli to update!

? Please pick a preset: Manually select features
? Check the features needed for your project: Choose Vue version, Babel, Router
? Choose a version of Vue.js that you want to start the project with 3.x (Preview)
? Use history mode for router? (Requires proper server setup for index fallback in production) (Y/n) y
```

El proyecto generado contendrá dos nuevas carpetas:

- router: carpeta que contendrá el archivo index.js, para la configuración de rutas. Este fichero, además de la configuración necesaria, se modificará incluyendo todas las rutas de la página.
- views: Carpeta para almacenar las distintas vistas de la aplicación, que serán componentes que renderizan una “página” de la misma. Los componentes que hagan las veces de vistas se almacenarán en esta carpeta, los que no seguirán en la carpeta components.

EJERCICIO 1

- Genera un nuevo proyecto que incluirá Router entre sus características. Llámalo primer_proy_router.
- Una vez generado ábrelo en VSC, mira el contenido de las nuevas carpetas y en conjunto comentamos los distintos componentes.
- Ejecuta el proyecto e intenta ver, al abrir la página en el navegador, que parte es componente y que parte vista.



3. Trabajando con Router

3.1. Configuración inicial

En este apartado se van a ir viendo los diferentes elementos, ficheros, etc. necesarios para hacer uso de vue-router. Vamos a ver los ejemplos partiendo del proyecto del ejercicio 1.

<router-link>

Con router-link se indicará cada una de las opciones de menú.

```
<router-link to="/">Home</router-link> |  
<router-link to="/about">About</router-link>
```

El valor del atributo **to** deberá estar declarado en el fichero de configuración de rutas.

Cuando se accede a una ruta, el elemento `<router-link>` toma la clase `.router-link-active`. En el ejemplo la opción de menú seleccionada se ve verde.

<router-view>

En este elemento se cargará la vista que en cada momento marque la ruta seleccionada. Cada vez que se cambie la URL en el navegador, el layout permanece cargándose únicamente el componente indicado para dicha ruta. En el fichero `Aapp.vue`, dentro de `<template>`, se incluirá esta etiqueta en la parte del HTML en que se quiera cargar las diferentes vistas.

index.js

En este fichero estará la configuración necesaria para hacer uso de router.

Primero de todo habrá que importar las dependencias de vue-router necesarias (si se ha generado el proyecto con CLI ya estará incluido) y, opcionalmente, los componentes de la aplicación que se cargan en cada ruta (los componentes considerados vistas).

La definición de cada ruta incluida tendrá los siguientes datos:

- **path**: La ruta en sí.
- **name**: El nombre de la ruta (normalmente se pone el mismo que el componente).
- **component**: El componente que carga dicha ruta. Este componente puede:
 - importarse al principio del fichero y referenciarse en este punto.
 - importarse directamente en el objeto de configuración de la ruta.

Por último, se crea el objeto Router que usará la aplicación, y se exporta para que pueda ser usado. (Esto también se genera automáticamente cuando el proyecto se crea con Vue CLI)

main.js

En este fichero se ha de importar el objeto router que se ha creado previamente y declararlo en la instancia de Vue para que sea accesible a todos los componentes:

```
import router from './router'  
createApp(App).use(router).mount('#app')
```

EJERCICIO 2

Partiendo del proyecto generado en el ejercicio 1, realiza las siguientes acciones:

- Crea un componente llamado Bibliografía. Teniendo en cuenta que será una vista, créalo en la carpeta correcta.
- Dicho componente tendrá una lista numerada de 10 filas y, para cada una de esas filas, una lista no numerada de 5 filas. El texto de cada una de las 10 filas numeradas será un país mientras que, para las filas de las listas no numeradas, el texto a incluir será el nombre de escritores nacidos en dicho país:

1.España

*Carlos Ruiz Zafón

*Miguel Delibes

...

2.Reino Unido

*J.K. Rowling

...

- La lista, en el componente anterior, estará centrada.
- Incluye la definición de la ruta en el fichero correcto.
- Incluye la ruta en App.vue.

Una vez comprobado que funciona todo lo anterior:

- Crea un componente cuyo contenido será un enlace a la página de Casa del Libro.
- Incluye ese componente en la bibliografía para que se muestre tras la lista.

3.2. Rutas con parámetros

**El ejemplo de este apartado lo encontrarás en el proyecto “Rutas con Parámetros” en el Aula Virtual.

Existe la posibilidad de configurar una ruta como dinámica y pasarle valores por parámetro.

Suele ser muy frecuente la necesidad de una vista que pueda leer parámetros de la ruta, por ejemplo en el caso de un blog cuando se busca un artículo concreto.

El parámetro se definirá con la ruta, añadiendo tras la misma `/:nombreParam`:

`path: '/ruta/:nombreParam',`

Y para, en el componente llamado, recoger el valor pasado:

`$route.params.nombreParam`



EJERCICIO 3

Partiendo del ejercicio 2:

- Crea una nueva vista que vaya a mostrar un texto. Crea, en ese componente, un array con 3 textos diferentes.
- Dicha vista mostrará el texto 1, 2 o 3 dependiendo del valor que le llegue como parámetro a través de la ruta:

`http://localhost:8080/ruta/1`

`http://localhost:8080/ruta/2`

`http://localhost:8080/ruta/3`

- En caso de que le llegue cualquier otro valor, mostrará el texto “No hay valor almacenado para ” y el valor llegado como parámetro.

*****Posible error en despliegue con modo History:***

Es posible que, al estar haciendo uso del modo history, si se despliega la aplicación en un hosting como netlify de un error 404. Para solucionar esto bastaría con realizar la siguiente incorporación al proyecto:

1. Agregar archivo `.htaccess` a la carpeta public del proyecto.
2. Incluir como contenido del archivo anterior lo siguiente:

```
<IfModule mod_rewrite.c>
RewriteEngine On
RewriteBase /
RewriteRule ^index\.html$ - [L]
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule . /index.html [L]
</IfModule>
```

3. Compilar de nuevo y actualizar en el hosting la carpeta dist.

4. Saltar a una ruta

Al hacer `createApp(App).use(router)` en `main.js`, se hace disponible router desde todos los componentes a través de `this.$router`. Con esto es posible acceder al router desde un componente para, por ejemplo, cambiar a una ruta.

Algunas de las opciones que ofrece router:

- `.push(newUrl)`: carga la vista a la que dirige la ruta pasada como parámetro y la añade al historial.
- `.replace(newURL)`: carga la vista a la que dirige la ruta pasada como parámetro pero reemplaza en el historial la ruta actual por esta.
- `.go(num)`: permite saltar el número de páginas, indicadas en el parámetro pasado, adelante (ej. `this.$route(go1)`) o atrás (`.go(-1)`) por el historial.

Y, de la misma forma que podemos declarar rutas con parámetros, podemos pasarle parámetros haciendo uso de estos métodos. Por ejemplo, para saltar a una ruta `/articulo`, con nombre `articulo`, pasándole el número de artículo a mostrar:

```
this.$router.push({name:"articulo", params: {id:2}});
```

Y en la barra de direcciones se verá: `/articulo/2`

4.1. Refresco de vista

Si se salta a la misma ruta pero cambiando el valor del parámetro, o parámetros, que recibe, Vue-router reutilizará la instancia ya creada del componente (ya que cachea éste último) y no vuelve a lanzar las *hooks* del mismo, y por ello no se ejecutará el código en dichas funciones incluido.

Para solucionar este tipo de problemas Vue Router dispone de los llamados **navegation guards** o interceptores de navegación. Estos interceptores son una serie de funciones que posibilitan realizar diferentes acciones durante la navegación de una ruta a otra, y pueden ser globales o locales.

Estos interceptores se declaran pasándoles los siguientes parámetros:

- **to**: objeto router con información de la ruta destino.
- **from**: objeto router con la información de la ruta origen.
- **next**: función que permite reanudar la navegación. La acción a ejecutar dependerá de los parámetros que se le pasen a `next()`:
 - sin parámetros: la navegación se reanudará hacia la ruta indicada en `to`.
 - Otra ruta: la navegación se redirigirá a la ruta indicada.
 - `false`: se abortará la redirección y se quedará en la ruta indicada en `from`.
 - Instancia de `Error`: se aborta la navegación y el error se pasará a la callback registrada via `router.onError()`.

Interceptores globales

Este tipo de interceptores se ejecutan para toda las rutas a las que se navegue, antes, durante o después de que se produzca la navegación, dependiendo del interceptor registrado.

El más utilizado de este tipo es `beforeEach()`:

- se ejecutará cada vez que se realice un cambio de ruta justo antes de que se produzca la navegación.
- Se registra tras la instanciación de vue router.
- Se puede registrar mas de uno, y serán llamados en el orden en que se hayan declarado.

Un ejemplo típico de uso es la comprobación de autenticación. Se comprueba si se va a navegar a una ruta diferente a la de login y, si es así, comprueba que tenga un token de sesión con la API. Si lo tiene, continúa con la navegación normal(`next()`). En caso contrario, redirige a la vista de login:

```
router.beforeEach((to, from, next) => {  
  if (to.name !== 'Login' && !isAuthenticated)  
    next({ name: 'Login' });  
  else  
    next();  
});
```

Interceptores locales a una ruta

Se incluyen en la configuración de la ruta.

Interceptores locales a nivel de componente

En este caso, los interceptores se definen directamente en las vistas. Las opciones son:

- `beforeRouteEnter()`: se ejecuta cuando la navegación ha sido confirmada, pero todavía no se ha creado, ni renderizado el componente.
- `beforeRouteUpdate()`: se ejecuta al cambiar la ruta y el componente se reutiliza para la siguiente ruta o por la misma. Por ejemplo, cuando cambia la ruta solo en el parámetro que se le pasa.
- `beforeRouteLeave()`: se ejecuta antes de cambiar de ruta y que el componente no va a ser utilizado. Como se ejecuta antes de ir a la nueva navegación, se suele utilizar para evitar navegaciones involuntarias.

Con la opción `beforeRouteUpdate()`, podemos solucionar el problema mencionado de actualización de la misma ruta con valores distintos pasados en un parámetro. Por ejemplo, si se está llamando a una ruta que está declarada con un parámetro llamado `num`:

```
beforeRouteUpdate (to, from, next) {  
  this.numPag = to.params.num; //numPag es variable del componente  
}
```




Flujo de ejecución

Los distintos interceptores se ejecutan según el siguiente orden:

- Se activa la navegación.
- Se llama a todos los `beforeRouterLeave` que se hayan registrado y que no van a ser reutilizados en la siguiente ruta.
- Se llama a todos los interceptores globales `beforeEach`.
- Se llama a todos los `beforeRouteUpdate` de los componentes que van a ser reutilizados en la siguiente ruta.
- Se llama a `beforeEnter` de la ruta destino, en caso de que se haya incluido.
- Se resuelven toda la asincronía de componentes de esa ruta.
- Se llama a `beforeRouteEnter` de los componentes que van a estar activos.
- Se da la navegación como confirmada.
- Se llama al interceptor `afterEach`.
- Se llama a los callbacks pasados a `next` en `beforeRouteEnter`.

******En el Aula Virtual encontrarás un ejemplo de uso de interceptores en el proyecto “Router Navigation Guards”

5. Axios

Axios es una de las librerías que ofrece Vue mas utilizadas. Esta librería permite hacer peticiones Ajax a un servidor de forma sencilla, ya que es un client HTTP basado en promesas.

5.1. Instalación y uso de axios

Para instalar esta librería como dependencia del proyecto, seguimos haciendo uso del gestor de paquetes de Node, npm.

EJERCICIO 4

- En un proyecto generado con Vue CLI, instala axios con npm y la opción `-S` o `--save`. Genera un nuevo proyecto o usa uno que ya tengas, pero si usas uno ya existente sería recomendable limpiarlo de desarrollos anteriores.
- Una vez instalado, para poder hacer uso de axios deberás importarlo. La importación puede hacerse:
 - de forma general en el fichero `main.js`
 - En el componente concreto que se vaya a utilizar: `import axios from "axios"`.

En el Aula Virtual encontrarás un enlace con un ejemplo.

5.2. Realizar peticiones

Axios incluye una serie de métodos para realizar peticiones:

- **.get(url)**: método para realizar una petición GET a la url pasada como parámetro.
- **.post(url, objeto)**: método para realizar una petición POST a la url pasada como parámetro.
- **.put(url, objeto)**: método para realizar una petición PUT a la url pasada como parámetro.
- **.delete(url)**: método para realizar una petición DELETE a la url pasada como parámetro.

Todos estos métodos devuelven promesas, por lo que usaremos los métodos, ya conocidos, `then` y `catch`.

La respuesta del servidor tiene, entre otras propiedades:

- **data**: con los datos devueltos por el servidor.
- **status**: código de la respuesta del servidor (200, 404, ...)
- **statusText**: el texto de la respuesta del servidor ('Ok', 'Not found', ...)
- **message**: mensaje del servidor en caso de producirse un error
- **headers**: las cabeceras HTTP de la respuesta

Por ejemplo, en la siguiente petición se están solicitando unos datos que se almacenarán en un array incluido en el data() del componente:

```
axios.get(url)
  .then(response => this.array = response.data)
  .catch(response => alert("Error al recuperar datos" + response.status));
```

EJERCICIO 5

- Modifica el ejercicio 1 de los ejercicios adicionales para que las peticiones realizadas se hagan haciendo uso de la librería axios.

5.3. Añadir cabeceras a la petición

Si fuera necesario incluir alguna cabecera en una petición axios, ésta se puede pasar en un objeto como tercer parámetro. Por ejemplo, si es necesario un token para la autenticación en la API:

```
ver () {
  const config = {
    headers: {
      'Authorization' = 'Bearer ' + localStorage.token;
    }
  }
  const data = ...;
  axios.post(baseUrl, data, config)
    .then(...)
```

Otra posibilidad es añadir una cabecera que se incluya en todas las peticiones de un tipo concreto que se realicen:

```
axios.defaults.headers.post['cabecera1'] = 'valor'
```

Y, por último, es posible añadir un cabecera a todas las peticiones(sean del tipo que sean) que se vayan a realizar:

```
axios.defaults.headers.common['cabecera1'] = 'valor'
```

5.4. Control de errores

Cuando se realiza un petición a una API, pueden producirse errores debidos a múltiples causas. Por ejemplo:

- Que la petición se haya realizado de forma incorrecta.
- Que la API se encuentre fuera de servicio.
- Que el formato en que la información es devuelta no sea el esperado.

Al realizar cualquier petición, hay que contar con esta circunstancia, e implementar el control de los posibles errores para asegurar el correcto funcionamiento de la página:

- Si algo fallo durante la llamada a la API, el control de errores se hace incluyendo la



sentencia `catch()` y, en ella, implementando las acciones necesarias para solventar el problema surgido de la falta de datos.

- En cambio, si el error no se produce en la llamada a la API sino que, por ejemplo, los datos están estropeados o la API fuera de servicio, habría que ir un paso mas allá. En el enlace incluido en el Aula Virtual “Errores al realizar petición” podemos ver un ejemplo.