Guía de ejercicios - Vue Router (II)

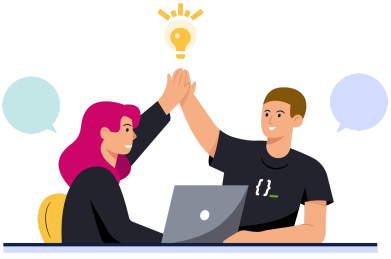
 ¡Hola! Te damos la bienvenida a esta nueva guía de estudio.

¿En qué consiste esta guía?

La siguiente guía de estudio tiene como objetivo practicar y ejercitar los contenidos que hemos visto en clase.

¡Vamos con todo!

Tabla de contenidos



¡Comencemos!

Contexto antes de iniciar

En vue debemos tener en cuenta que existe una diferencia conceptual entre qué es un componente y qué es una vista.

- Cuando creamos componentes estamos desarrollando piezas de código reutilizables a lo largo de nuestras aplicaciones. Estas piezas de código reutilizables pueden ser un menú de navegación, botones, cards, entre otros elementos que logremos identificar como reutilizables.
- Cuando generamos vistas, nos referimos a componentes que muestran secciones específicas dentro de una aplicación. Entonces, **¿las vistas también son componentes?** la respuesta es sí, su comportamiento y sintaxis es de componente, pero la diferencia radica en que no necesariamente son piezas de código reutilizables, solo tienen la tarea concreta de mostrar contenido a partir de una ruta especificada.

Dado este contexto previo, pasemos entonces a seguir trabajando con vue router y algunos otros elementos que nos provee Vue JS.

Actividad guiada: Rutas nombradas en vue router

A continuación, realizaremos un ejercicio en el cual le asignaremos nombres a las rutas para optimizar los tiempos de modificación que vayan surgiendo en la medida que se desarrollan las SPA en vue. El objetivo a lograr es el siguiente:

La imagen es un resultado parcial de la aplicación que construiremos, consta de un menú de navegación implementado con Bootstrap. Este, utiliza `<router-link>` para generar la navegación entre las vistas de Productos y Contacto. Además, esta SPA podrá generar datos dinámicos que se envíen a través de `props` a un componente.

Vamos entonces al código y sigamos los pasos.

- **Paso 1:** Creamos una aplicación asignando el nombre rutas-nombradas.

```
npm create vue@latest
```

- **Paso 2:** Definimos el setup de la aplicación y seleccionamos el preset para vue router.
- **Paso 3:** Abrimos la aplicación en el editor de código.

Para este ejercicio utilizaremos Bootstrap integrado mediante CDN, en caso de que quieras implementar los estilos con los archivos fuente no hay ningún inconveniente.

- **Paso 4:** En el archivo `index.html` insertamos el CDN de Bootstrap. Recuerda que el archivo index.html se encuentra en la carpeta `/public`.

```
<link rel="stylesheet" href="
https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
```

- **Paso 5:** Limpiamos la aplicación, en este sentido eliminamos el componente `HelloWorld.vue` y eliminamos su referencia en la vista `HomeView.vue`. Al realizar esto veremos el siguiente resultado.

Imagen 1. Limpieza de la aplicación
Fuente: Desafío Latam

- **Paso 6:** A continuación, vamos a integrar en la aplicación en el directorio `/components` uno asociado al Navbar. Te preguntarás **¿por qué en components?**, bajo la lógica de la aplicación, este componente es una pieza que se reutilizará en toda la aplicación, es por ello que tiene sentido que esté configurado como componente y no como una simple vista generada por vue router.

Código del componente Navbar seleccionado en Bootstrap:

```
<template lang="">
  <nav class="navbar navbar-expand-lg bg-body-tertiary">
    <div class="container-fluid">
      <a class="navbar-brand" href="#">Navbar</a>
      <button
        class="navbar-toggler"
        type="button"
        data-bs-toggle="collapse"
        data-bs-target="#navbarNavAltMarkup"
        aria-controls="navbarNavAltMarkup"
        aria-expanded="false"
        aria-label="Toggle navigation"
      >
        <span class="navbar-toggler-icon"></span>
      </button>
      <div class="collapse navbar-collapse" id="navbarNavAltMarkup">
        <div class="navbar-nav">
          <a class="nav-link active" aria-current="page" href="#">Home</a>
          <a class="nav-link" href="#">Features</a>
          <a class="nav-link" href="#">Pricing</a>
          <a class="nav-link disabled">Disabled</a>
        </div>
      </div>
    </div>
  </nav>
</template>
```

Como podemos apreciar, este componente está utilizando la etiqueta `<a>` de HTML para enlazar las secciones del sitio, recuerda que con vue router debemos utilizar la etiqueta `<router-link>` para redireccionar. Realiza la sustitución entonces de todas las anclas del menú por `<router-link>`, además recuerda modificar los `href` y cambiarlos por `to`.

Código del Navbar modificado:

```
<template>
  <nav class="navbar navbar-expand-lg bg-body-tertiary">
    <div class="navbar-nav">
      <router-link class="nav-link active" aria-current="page"
to="/">Home</router-link>
      <router-link class="nav-link" :to="{name:
'productos'}">Productos</router-link>
      <router-link class="nav-link" :to="{name:
'contacto'}">Contacto</router-link>
    </div>
  </nav>
</template>
```

- **Paso 7:** Ha llegado el momento de mostrar nuestro Navbar, para ello debemos ir a el componente principal `App.vue` e importarlo.

Código del App.vue:

```
<template>
  <Navbar />
  <router-view/>
</template>

<script>
  import Navbar from './components/Navbar.vue'

  export default {
  components: {
    Navbar
    }
  }
</script>
```

¡Nótese el código resaltado!
Te preguntarás qué es >router-view />

1. Esta etiqueta que se integra en el `App.vue` lo que indica es que pintará todos los componentes(vistas) que estén definidos en el archivo de router.
2. En este sentido, todas estas vistas serán cargadas dinámicamente y de manera automática por el componente principal `App.vue`.

Por los momentos nuestra aplicación se ve como muestra la siguiente imagen:

Imagen 2. Resultado parcial con menú de navegación
Fuente: Desafío Latam

¡Excelente!, hasta este punto ya nuestro Navbar está disponible en toda la aplicación. Veamos entonces el proceso para enlazar con las rutas y asignarles un nombre.

- **Paso 8:** Crearemos las rutas en el archivo `router/index.js` para poder enlazar nuestro menú y que funcione correctamente. Recuerda que en el código del menú definimos una sección para Productos y otra para Contacto, en este sentido, debemos generar las rutas y las vistas.

Código del arreglo de objetos que contienen las rutas:

```
const routes = [
  {
    path: '/',
    name: 'home',
    component: HomeView
  },
  {
    path: '/productos',
    name: 'productos',
    component: () => import(/* webpackChunkName: "productos" */
'../views/ProductosView.vue')
  },
  {
    path: '/contacto',
    name: 'contacto',
    component: () => import(/* webpackChunkName: "contacto" */
'../views/ContactoView.vue')
  }
]
```

Código de la vista `ProductosView.vue`.

```
<template lang="">
  <h1>
    Listado de productos
  </h1>
</template>
```

Código de la vista `ContactoView.vue`.

```
<template lang="">
  <h1>
    Contáctanos
  </h1>
</template>
```

Rutas nombradas

Cuando trabajemos con rutas es una buena práctica de optimización de código, asignarle o referenciarlas mediante un nombre. Este nombre viene determinado en la propiedad del objeto de rutas llamado `name`.

Esto surge a partir de una problemática, que consiste en que si tenemos una ruta que está siendo referenciada en múltiples componentes, sería complicado hacerle mantenimiento si decidimos cambiarle el enlace a la cual se redirige, en ese sentido, debemos ir a la **n** archivos que contiene la ruta y modificarlos.

Ante ese escenario, existe una solución a través de las rutas nombradas en los enlaces o <router-link>.

- **Paso 9:** Asignemos entonces a cada `<router-kink>` de nuestro menú el nombre de las rutas definidas.

```
<router-link class="nav-link" :to="{name: 'productos'}">Productos</router-link>
<router-link class="nav-link" :to="{name: 'contacto'}">Precios</router-link>
```

Nótese lo resaltado en verde; en el `to` asignamos dos puntos antes, esto hará que asignemos el name definido en las rutas de manera nombrada y no de manera estática como lo teníamos en un principio.

Otras formas de redireccionar

En el bloque anterior, vimos cómo generar redirecciones a través de this.$router.push y esta acción se pasaba a un botón a través de un method. A continuación, veremos otra forma de redirigir al visitante a una sección específica del sitio.

- **Paso 10:** En la vista ProductosView.vue añadiremos un <router-linlk> que permita redireccionar al visitante a la vista home.

```
<template lang="">
  <h1>
    Listado de productos
  </h1>

  <router-link to='/'>Volver al inicio</router-link>
</template>
```

Nótese el código resaltado; esta es otra forma de redirigir, esta opción y la vista en la sesión anterior son válidas, puedes utilizar la que más te haga sentido o sea cómoda de implementar.

Veamos el siguiente resultado de implementación en la siguiente imagen.

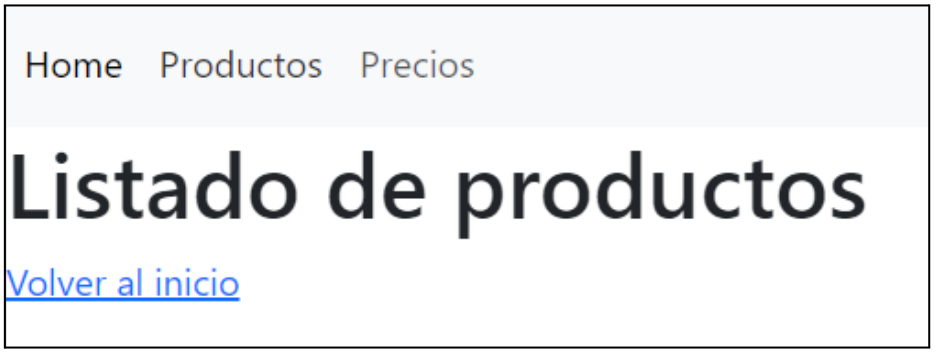


Imagen 3. Resultado de ejecución de redirección con <router-lik>
Fuente: Desafío Latam

Rutas dinámicas con parámetros recibidos por props

Es posible pasar props a través del archivo de rutas a nuestras vistas o componentes que muestran dicha vista. Veamos cuál es el procedimiento trabajando con el mismo ejercicio anterior.

- **Paso 11:** Crearemos una nueva vista llamada `ProductoDetalle.vue`, esta contendrá el siguiente código.

```
<template>
  <h1>Detalle del producto</h1>
  <p>{{ id }} - {{ nombre }} - {{ precio }}</p>
</template>

<script>
export default {
  props: {
    id: {
      type: Number,
      required: true
    },
    nombre: {
      type: String,
      required: true
    },
    precio: {
      type: Number,
      required: true
    }
  }
```

```
}
</script>
```

Entendamos el código

1. En nuestro template tenemos un h1 y en un párrafo mostramos datos que se obtienen de manera dinámica a través de `props`.
2. Las `props` que están definidas dentro del bloque del `export default` están definidas de tal manera para que estas tengan un formato establecido:
    a. Se espera que el `id` sea recibido como un número y además es obligatorio, esto último se logra con la propiedad `required` y su valor en `true`.
    b. Esto lo hacemos con cada una de las propiedades y varía como es el caso del nombre, en el cual se espera que sea un `String`.

- **Paso 12:** Ahora en el archivo `routes/index.js` agregamos una nueva ruta llamada `ProductoDetalle` y recibirá un conjunto de `props` que se enviarán al componente que definimos en el paso anterior. Veamos el código del `index.js`.

Código para Ruta /ProductoDetalle

```
{
    path: '/producto/:id',
    name: 'producto',
    component: () => import(/* webpackChunkName: "producto_detalle" */
'../views/ProductoDetalleView.vue'),
    props: (route) => {
      return {
        id: Number(route.params.id),
        nombre: "Tomate",
        precio: 2000
      }
    }
  },
```

¡Nótese el código resaltado!

1. El `path` de la ruta es dinámica, esto se logra con la sintaxis de dos puntos antes de la palabra id `(:id)`.
2. Podemos generar rutas que envíen `props` con información que se mostrará dinámicamente. En el caso de la propiedad id, estamos diciendo que envíe a través de la función `Number` los parámetros de la ruta que se ingresen. En este sentido, si escribimos en el navegador la siguiente ruta http://localhost:8080/producto/100 veremos el siguiente resultado

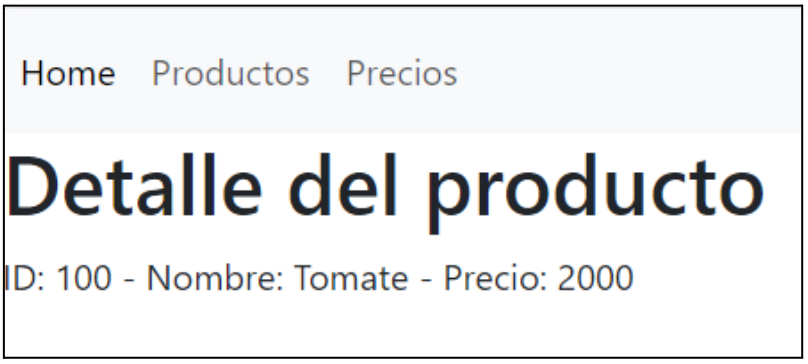


Imagen 4. Pasando props mediante rutas a una vista
Fuente: Desafío Latam

Rutas anidadas

En un sitio web es muy común mostrar vistas que dependen una de otra. Los sitios web se conforman de múltiples niveles en sus URL's, ya que las piezas que conforman el sitio pueden ser encasilladas dentro de otras vistas. Imaginemos el caso de los posteos que realiza un usuario en una red social o las reservas de horas de un usuario en un portal de agendamientos. Es por esto que Vue Router nos provee de las rutas anidadas, que son estructuras de rutas que respetan una jerarquía definida por nosotros.

Veamos cómo implementar esto utilizando el mismo ejercicio que venimos desarrollando. Esta vez, haremos una modificación en la ruta /productos, en este caso le añadiremos una ruta ancestro o conocida también como children.

- **Paso 13:** Añadir una ruta children a /producto/:id. Este ancestro lo que hará será mostrar comentarios asociados a dicho producto.

Código de la ruta modificado:

```
{
    path: '/producto/:id',
    name: 'producto',
    component: () => import(/* webpackChunkName: "producto_detalle" */
'../views/ProductoDetalleView.vue'),
    props: (route) => {
      return {
        id: Number(route.params.id),
        nombre: "Tomate",
        precio: 2000
      }
    },
    children: [
      {
```

```
        path: 'comentarios',
        component: () => import(/* webpackChunkName: "comentarios" */
'../views/ComentariosView.vue'),
        name: 'comentarios'
      },
    ]
  },
```

¡Nótese el código resaltado!

1. Esta es la manera de agregar una ruta anidada en vue router.
2. A través de `children`, definimos entre un arreglo el objeto anidado a la ruta
3. Dado que definimos una nueva vista `ComentariosView`, debemos entonces crear su respectivo componente en la carpeta `/views`.

- **Paso 14:** Generamos la el código de la vista `ComentariosView.vue`

```
<template lang="">
  <div>
    Comentarios
  </div>
</template>
```

- **Paso 15:** Ahora, en el componente `ProductoDetalleView.vue` agregamos el siguiente código en el <template>.

```
<template>
  <h1>Detalle del producto</h1>
  <p>ID: {{ id }} - Nombre: {{ nombre }} - Precio: {{ precio }}</p>

  <router-link :to="{name: 'comentarios'}">Ver comentarios</router-link>
  <router-view />
</template>
```

¡Nótese el código resaltado!

1. En el <router-link> definimos la ruta nombrada la cual nos llevará a comentario de este producto.
2. Es importante visualizar que se agregó `<router-view />`
3. Esto último lo agregamos dado que el componente ProductoDetalleView puede tener rutas hijas, esas rutas hijas entonces renderízalas en este mismo componente.

Si accedemos entonces a la siguiente url http://localhost:8080/producto/1 y damos clic en ver comentarios veremos el siguiente resultado.

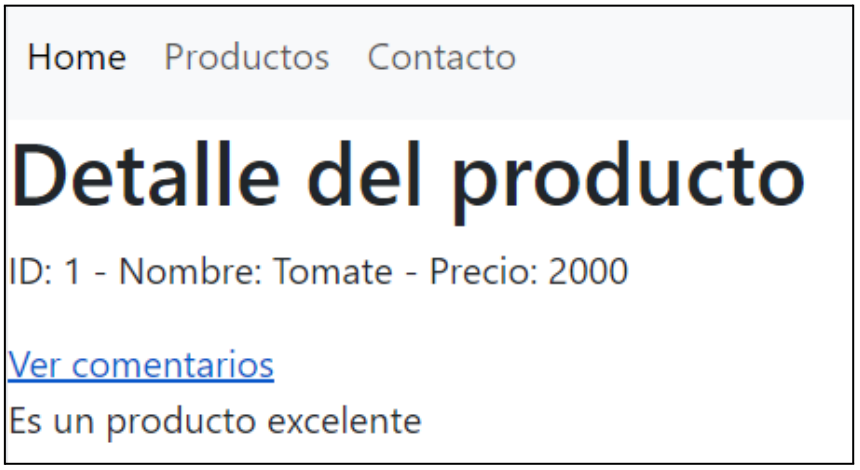


Imagen 5. Visualizando los comentarios de un producto
Fuente: Desafío Latam

¡Manos a la obra! - Integra un enlace de redirección en la vista de contacto

Para generar esta redirección puedes utilizar el proceso implementado en este material utilizando <router-link>. Al dar clic sobre el link deberá redimir al visitante al home.

¡Manos a la obra! - Pasar props a la vista de contacto

Para esta actividad, deberás pasarle props desde la definición de la vista contacto hasta su componente que genera la vista. Recuerda implementar los pasos 11 y 12 del ejercicio anterior para poder lograrlo.

Solución Manos a la Obra

Integra un enlace de redirección en la vista de contacto

```
<template lang="">
  <h1>
    Contáctanos
  </h1>
  <router-link to='/'>Volver al inicio</router-link>
</template>
```

Pasar props a la vista de contacto

```
Vista Contacto.vue
<template lang="">
  <h1>
    Contáctanos a través de la siguiente información
  </h1>
  <p>Teléfono: {{telefono}}</p>
  <p>Correo: {{email}}</p>
  <router-link to='/'>Volver al inicio</router-link>
</template>

<script>
export default {
  props: {
    telefono: {
      type: String,
      required: true
    },
    email: {
      type: String,
      required: true
    },
  }
}
</script>
```

Archivo index.js para la ruta /contacto.

```
{
    path: '/contacto',
    name: 'contacto',
    component: () => import(/* webpackChunkName: "contacto" */
'../views/ContactoView.vue'),
    props: () => {
      return {
        telefono: "56922222222",
        email: "correo@correo.com"
     }
   }
}
```