

Solución Prueba - Games Opinion Site

Requerimiento 1

- **Paso 1:** Ingresar a la terminal de nuestro computador y escribir el comando "vue create prueba_vue"
- **Paso 2:** Seleccionar y hacer un enter sobre la opción: "Manually select features".
- **Paso 3:** Seleccionar por medio de la barra espaciadora, las opciones de: Babel, Router, Vuex, Linter / Formatter, Unit Testing, E2E Testing.
- **Paso 4:** Elegir la versión 2.x
- **Paso 5:** Indicar con una "Y" que si utilizaremos el modo historia para el Router.
- **Paso 6:** Presionar la tecla enter para la primera opción: "ESLint with error prevention only".
- **Paso 7:** Presionar la tecla enter para la primera opción que parece: "Lint on save".
- **Paso 8:** Seleccionar la solución para el test unitario de tu preferencia, ya sea Jest o Mocha + Chai.
- **Paso 9:** Seleccionar la solución para el test E2E de tu preferencia, ya sea Cypress, Nightwatch o WebdriverIO.
- **Paso 10:** Indicar que trabajaras con "In package.json" para tener un solo archivo de configuración general.
- **Paso 11:** Finalmente queda a tu preferencia si guardas o no el preset para futuras instalaciones.

Los pasos anteriores generarán una carpeta con todos los archivos necesarios para iniciar el trabajo, como se muestra a continuación:

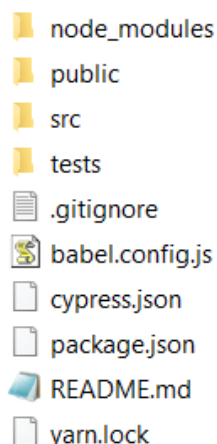


Imagen 1. Directorio principal generado con Vue CLI

Fuente: Desafío Latam.

- **Paso 12:** Ahora, nuevamente en la terminal instala las dependencias de tu preferencia y configuralas en el archivo main.js. En este caso se instaló y configuró como dependencia externas: Bootstrap y Fontawesome. Siendo los comandos:


```
npm install bootstrap@next @popperjs/core @fortawesome/fontawesome-free  
@fortawesome/fontawesome-svg-core @fortawesome/free-solid-svg-icons  
@fortawesome/vue-fontawesome
```

Archivo main.js

```
import Vue from 'vue';  
import App from './App.vue';  
import router from './router';  
import store from './store';  
// se debe importar la librería de bootstrap en conjunto con su archivo  
.css  
import 'bootstrap';  
import 'bootstrap/dist/css/bootstrap.min.css';  
// para trabajar con los iconos, se deben importar tanto el core de la  
librería de fontawesome, como los iconos que se van a utilizar y el  
fontawesome para vue.  
import { library } from '@fortawesome/fontawesome-svg-core';  
import { faUsersCog, faComments, faHome, faHeadset } from  
'@fortawesome/free-solid-svg-icons';  
import { FontAwesomeIcon } from '@fortawesome/vue-fontawesome';  
//luego se debe añadir cada icono a la librería para indicar su uso.  
library.add(faUsersCog);  
library.add(faComments);  
library.add(faHome);  
library.add(faHeadset);  
//finalmente se crea el componente para fontawesome.  
Vue.component('font-awesome-icon', FontAwesomeIcon);  
  
Vue.config.productionTip = false  
  
new Vue({  
  router,  
  store,  
  render: h => h(App)  
}).$mount('#app')
```

Requerimiento 2

- **Paso 1:** Crear dos archivos en la carpeta de components, uno con el nombre de "ListaAdministrada.vue" y otro con el nombre de "TheNavBar.vue".
- **Paso 2:** Crear la estructura básica de cada archivo.
- **Paso 3:** En el caso del archivo para el menú de navegación "TheNavBar.vue", agrega la estructura de un NavBar según bootstrap (<https://getbootstrap.com/docs/5.0/components/navbar/#nav>). Realiza las modificaciones pertinentes si deseas lograr que se vea como el solicitado (ver "Imagen 2").

 Games Opinion




 Home  Opiniones  Administración

Imagen 2.Menú de navegación.
Fuente: Desafío Latam.

Archivo TheNavBar.vue

```
<template>
  <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
    <div class="container-fluid">
      <a class="navbar-brand" href="#"><font-awesome-icon
:icon="['fas', 'headset']" /> Games Opinion</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 ms-auto">
<!-- Se utiliza el router link para sustituir las etiquetas "a",
haciendo el llamado a las rutas por nombre, que se agregara
posteriormente en el archivo de router -->
          <router-link class="nav-link" :to="{name:
'Home'}"><font-awesome-icon :icon="['fas', 'home']" />
Home</router-link>
          <router-link class="nav-link" :to="{name:
'Opiniones'}"><font-awesome-icon :icon="['fas', 'comments']" />
Opiniones</router-link>
          <router-link class="nav-link" :to="{name:
'Administracion'}"><font-awesome-icon :icon="['fas', 'users-cog']" />
Administración</router-link>
        </div>
      </div>
    </div>
  </nav>
</template>
```

- **Paso 4:** En el caso del archivo para mostrar la lista de opiniones "ListaAdministrada.vue", agrega la estructura de una tabla según bootstrap (<https://getbootstrap.com/docs/5.0/content/tables/#overview>). Realiza las modificaciones pertinentes si deseas lograr que se vea como el solicitado, como se muestra en la "Imagen 3". Puedes agregar información falsa de prueba para observar el comportamiento del componente.

Administrando la Lista de Opiniones

#	Persona	Juego	Opinion		
1	Evan You	Grand Theft Auto V	Nueva Opinión sobre el juego GTAV	Eliminar	Editar

Imagen 3. Tabla con lista de comentarios.
Fuente: Desafío Latam.

Archivo ListaAdministrada.vue

```
<template>
  <div class="container">
    <h2 class="text-center mt-5">Administrando la Lista de
Opiniones</h2>
    <div class="mt-5">
      <div class="table-responsive">
        <table class="table table-hover">
          <thead class="table-light">
            <tr>
              <th>#</th>
              <th>Persona</th>
              <th>Juego</th>
              <th>Opinión</th>
              <th></th>
              <th></th>
            </tr>
          </thead>
          <tbody>
            <tr>
              <td></td>
              <td></td>
              <td></td>
              <td></td>
              <td><button type="button" class="btn
btn-danger">Eliminar</button></td>
              <td><button type="button" class="btn
btn-info">Editar</button></td>
            </tr>
          </tbody>
        </table>
      </div>
    </div>
    <div class="alert alert-danger mt-5" role="alert">
```

```
    No existen opiniones para administrar.  
  </div>  
</div>  
</template>
```

- **Paso 5:** Crear un archivo en la carpeta de “view” que servirá como padre del componente creado anteriormente, con el nombre de: “Administracion.vue”.
- **Paso 6:** En el componente padre denominado “Administración”, se debe importar el componente hijo denominado “ListaAdministrada.vue” e implementarlo mediante el comando “import / from”. Luego utilizarlo dentro del template como una etiqueta HTML, como se enseñó a lo largo del módulo.

Archivo Administracion.vue

```
<template>  
  <div>  
    <!-- Llamado al componente mediante la etiqueta en el template -->  
    <lista-administrada></lista-administrada>  
  </div>  
</template>
```

```
<script>  
  //utiliza la instrucción de ES6 import/from para traer el componente  
  hijo  
  import ListaAdministrada from '../components/ListaAdministrada.vue';  
  
  export default {  
    name: 'Administracion',  
    // utiliza la propiedad components para indicar cuál será el componente  
    a usar  
    components: {  
      ListaAdministrada  
    }  
  }  
</script>
```

- **Paso 7:** Ahora, en el caso del archivo App.vue, se debe importar el componente “TheNavBar” y dejar el “created” del ciclo de vida listo para usar en los siguientes requerimientos. (Más adelante se explicará cómo realizar el llamado a la API, pero recuerda que desde el inicio puedes adelantar este proceso para probar con los datos directos de la API o agregar información falsa para después sustituir).

Archivo App.vue

```
<template>
  <div>
    <div>
      <TheNavBar />
    </div>
    <!--se emplea el router-view para que cuando se configure y aplique el
    router, las vistas se muestran en esta sección.-->
    <router-view/>
  </div>
</template>

<script>
import TheNavBar from '@components/TheNavBar.vue';

export default {
  components: {
    TheNavBar
  },
  created() {
    //aquí se aplicará el llamado a la acción de vuex para comunicarse
    con la API.
  }
}
</script>
```

Requerimiento 3

- **Paso 1:** En el archivo Home.vue se desplegará toda la información que trae la API. Por lo que utilizaremos tarjetas “card” para mostrarla, las cuales se pueden implementar desde bootstrap (<https://getbootstrap.com/docs/5.0/components/card/#example>). Esto nos obliga a implementar la directiva “v-for” para recorrer un arreglo con los distintos objetos que podría traer la API. Para ello, puedes aplicar el ciclo de vida en el componente “home” de forma temporal, para tener los datos disponibles o aplicar información falsa temporalmente.
- **Paso 2:** Vamos a probar la conexión a la API y así utilizar los datos momentaneamente, recuerda que este proceso se realizará posteriormente utilizando Vuex, pero si lograste adelantar todo ese proceso, puedes probar perfectamente los datos desde la store.

Archivo Home.vue

```
<template>
  <!-- para lograr que las tarjetas se muestran en forma de columnas, se
  debe utilizar la grilla de bootstrap con sus propiedades básicas -->
  <div class="container">
    <h1 class="text-center mt-5">Lista de Juegos Disponibles</h1>
    <div class="row mt-5">
      <!-- aquí podemos utilizar directamente la data de la API o cualquier data
      falsa pasandola como una variable directa desde la data o desde la
      store-->
      <div class="col-12 col-sm-12 col-md-6 col-lg-4 col-xl-4 my-4"
      v-for="(juego, index) in dataAPI" :key="index">
        <div class="card">
          
          <div class="card-body">
            <h5 class="card-title" v-text="juego.name"></h5>
            </div>
            <ul class="list-group list-group-flush">
              <li class="list-group-item">Rating: {{juego.rating}}</li>
              <li class="list-group-item">Released:
              {{juego.released}}</li>
              <li class="list-group-item">Updated:
              {{Intl.DateTimeFormat('CLP').format(new Date(juego.updated))}}</li>
            </ul>
            <div class="card-body text-center">
              <!--el data-bs-target del botón se hace dinámico usando la directiva
              v-bind y concatenando el index generado en v-for. Esto permitirá generar
              un botón independiente por cada card-->
              <button type="button" class="btn btn-primary"
              data-bs-toggle="modal" :data-bs-target="'#modal-'+index">Opinar</button>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
</template>

<script>

export default {
  //aquí se guardará la data momentáneamente de la API.
```



```
data(){
  return{
    dataAPI: []
  }
},
async created() {
  //aquí se aplicará el llamado momentáneo para comunicarse con la API y
  guardar la información en la data local del componente.
  let result = await fetch('https://api.rawg.io/api/games');
  let json = await result.json();
  this.dataAPI = json.results;
}
}
</script>
```

- **Paso 3:** En el mismo archivo Home.vue, se debe aplicar la ventana modal que permita a los usuarios del sitio web agregar una opinión sobre cualquier juego, puedes utilizar la ventana modal que facilita bootstrap en: ["https://getbootstrap.com/docs/5.0/components/modal/#live-demo"](https://getbootstrap.com/docs/5.0/components/modal/#live-demo). Para ello, se aprovechan las variables del ciclo generado con la directiva "v-for" en el paso anterior y hacer dinámica la modal, agregando la directiva "v-bind" tanto al "data-bs-target" del botón de la "card" como la "id" de la ventana modal, pasando el "index" del ciclo for como argumento a cada elemento.

Archivo Home.vue

```
<template>
  <div class="container">
    <h1 class="text-center mt-5">Lista de Juegos Disponibles</h1>
    <div class="row mt-5">
      <div class="col-12 col-md-6 col-lg-4 col-xl-4 my-4"
        v-for="(juego, index) in dataAPI" :key="index">
        <div class="card">
          
          <div class="card-body">
            <h5 class="card-title" v-text="juego.name"></h5>
            </div>
            <ul class="list-group list-group-flush">
              <li class="list-group-item">Rating: {{juego.rating}}</li>
              <li class="list-group-item">Released:
                {{juego.released}}</li>
              <li class="list-group-item">Updated:
```

```
{{Intl.DateTimeFormat('CLP').format(new Date(juego.updated))}}</li>
</ul>
<div class="card-body text-center">
<!--el data-bs-target del botón se hace dinámico usando la directiva
v-bind y concatenando el index generado en v-for.-->
<button type="button" class="btn btn-primary"
data-bs-toggle="modal" :data-bs-target="'#modal-'+index">Opinar</button>
</div>
</div>
<!-- modal -->
<!--el id de la modal se hace dinámico usando la directiva v-bind y
concatenando el index generado en v-for.-->
<div class="modal fade" :id="'modal-'+index" tabindex="-1"
:aria-labelledby="'exampleModalLabel'+index" aria-hidden="true"
ref="modalOpinion">
<div class="modal-dialog modal-dialog-centered">
<div class="modal-dialog">
<div class="modal-content">
<div class="modal-header">
<h5 class="modal-title">Escribe tu opinión para el
juego: {{juego.name}}</h5>
<button type="button" class="btn-close"
data-bs-dismiss="modal" aria-label="Close"></button>
</div>
<div class="modal-body">
</div>
<div class="modal-footer">
<!-- botón de cancelar y botón de guardar, este último con las
directivas que activan los eventos y ejecutan los métodos.-->
<button type="button" class="btn btn-secondary"
data-bs-dismiss="modal">Cerrar</button>
<button type="button" class="btn btn-primary"
data-bs-dismiss="modal" @click="guardarOpinion(juego)">Guardar</button>
</div>
</div>
</div>
</div>
</div>
</div>
</template>
```

- **Paso 4:** Dentro de la ventana modal, aplicaremos el formulario para que el usuario ingrese su nombre y comentario. A cada input se le debe agregar la directiva v-model con la variable respectiva que se agregará en la data del componente.
- **Paso 5:** Ahora, al botón en el footer de la modal que nos permite guardar las opiniones, se le agregará el evento "click" para llamar a un método, por ejemplo: "guardarOpinion".

Archivo Home.vue

```
<template>
  <div class="container">
    .
    .
    .
    <!-- modal -->
    <!--el id de la modal se hace dinámico usando la directiva v-bind y
    concatenando el index generado en v-for.-->
    <div class="modal fade" :id="'modal-'+index" tabindex="-1"
    :aria-labelledby="'exampleModalLabel'+index" aria-hidden="true"
    ref="modalOpinion">
      <div class="modal-dialog modal-dialog-centered">
        <div class="modal-dialog">
          <div class="modal-content">
            <div class="modal-header">
              <h5 class="modal-title">Escribe tu opinión para el
              juego: {{juego.name}}</h5>
              <button type="button" class="btn-close"
              data-bs-dismiss="modal" aria-label="Close"></button>
            </div>
            <div class="modal-body">
              <div class="mb-3">
                <!-- formulario para que el usuario ingrese el nombre y la opinión
                pertinente al juego seleccionado.-->
                <label for="nombre"
                class="form-label">Nombre:</label>
                <input type="text" class="form-control"
                placeholder="Evan You" v-model="nombre">
              </div>
              <div class="mb-3">
                <label for="opiniones"
                class="form-label">Opiniones</label>
                <textarea class="form-control" rows="3"
                v-model="opiniones" placeholder="Tu opinión debe ir aquí..."></textarea>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
</template>
```

```
</div>
<div class="modal-footer">
<!-- botón de cancelar y botón de guardar, este último con las
directivas que activan los eventos y ejecutan los métodos.-->
    <button type="button" class="btn btn-secondary"
data-bs-dismiss="modal">Cerrar</button>
    <button type="button" class="btn btn-primary"
data-bs-dismiss="modal" @click="guardarOpinion(juego)">Guardar</button>
</div>
</div>
</div>
</div>
.</div>
.</template>
```

- **Paso 6:** En el método “guardarOpinion”, se debe validar como mínimo que el usuario ingrese un nombre y una opinión.

Archivo Home.vue

```
<script>

export default {
  name: 'Home',
  data() {
    return {
      nombre: '',
      opiniones: '',
      dataAPI: []
    }
  },
  computed: {
    //aquí se traen los estados necesarios para mostrar en el componente
  },
  methods: {
    //método que permite verificar si el usuario ingresa un nombre y opinión
    //antes de guardar la opinión mediante la ejecución de una acción en el
    //vuex (se ejecutará posteriormente).
    guardarOpinion(juego){
```

```
        if (this.nombre && this.opiniones) {
            alert("Opinión agrega exitosamente");
            this.nombre = "";
            this.opiniones = "";
        } else {
            alert("Error al ingresar los datos.");
        }
    }
}
}
async created() {
//aquí se aplicará el llamado momentáneo para comunicarse con la API y
guardar la información en la data local del componente.
    let result = await fetch('https://api.rawg.io/api/games');
    let json = await result.json();
    this.dataAPI = json.results;
}
}
</script>
```

- **Paso 7:** Para el archivo de "ListaAdministrada.vue" se debe trabajar con la directiva v-for para iterar los datos que lleguen desde el vuex, es decir, las opiniones. En este caso utilizaremos datos falsos de prueba que serán borrados posteriormente. Igualmente se debe utilizar la directiva v-if/v-else para intercambiar entre los mensajes a mostrar por si no existe ninguna opinión que mostrar aún.
- **Paso 8:** Agregar los botones para eliminar o editar los comentarios. Estos botones deben estar con el evento "click" y llamar un método en específico para cada uno.
- **Paso 9:** El método para eliminar, llamado por ejemplo "eliminando" debe recibir el index o posición del elemento para saber cuál se borrará en el Vuex.

Archivo ListaAdministrada.vue

```
<template>
  <div class="container">
    <h2 class="text-center mt-5">Administrando la Lista de
    Opiniones</h2>
    <div class="mt-5" v-if="dataOpinion.length > 0"
    data-cy="administrandoLista">
      <div class="table-responsive">
        <table class="table table-hover">
          <thead class="table-light">
            <tr>
              <th>#</th>
              <th>Persona</th>
```

```
        <th>Juego</th>
        <th>Opinión</th>
        <th></th>
        <th></th>
    </tr>
</thead>
<tbody>
<!-- Aquí se iterara el getters que viene con la información de las
opiniones desde la store de Vuex. Si ya tienes esta parte lista, la
puedes utilizar directamente. Por los momentos utilizaremos datos falsos
falsa directamente desde la data del componente -->
    <tr v-for="(item,index) in dataOpinion" :key="index">
        <td>{{index+1}}</td>
        <td v-text="item.nombre"></td>
        <td v-text="item.juego.name"></td>
        <td v-text="item.opinion"></td>
        <td><button type="button" class="btn btn-danger"
@click="eliminando(index)">Eliminar</button></td>
        <td><button type="button" class="btn btn-info"
@click="editando(item)">Editar</button></td>
    </tr>
</tbody>
</table>
</div>
</div>
<div class="alert alert-danger mt-5" role="alert" v-else>
    No existen opiniones para administrar.
</div>
</div>
</template>

<script>
export default {
    name: 'ListaAdministrada',
    data(){
        return{
// Datos falsos solo con propósito de prueba. Luego serán borrados.
            dataOpinion: [
                {
                    nombre: 'Alison',
                    juego: {
                        name: 'GTAV',
                    },
                    opinion: "consequuntur autem libero",
```

```
        id: 111
      }
    ]
  },
  computed: {
    //aquí se traen los estados necesarios para mostrar en el
    componente
  },
  methods: {
    eliminando(index){
      let resultado = confirm("Seguro que desea eliminar");
      if (resultado) {
        //aquí se aplicará la activación de la acción correspondiente en
        el Vuex para eliminar una opinión.
        console.log(index); // muestra del index momentánea
      }
    },
    editando(item){
      //aquí se aplicará el traslado a la ruta de edición de manera
      dinámica, mediante el id que posee cada comentario.
      console.log(item); // muestra del ítem momentánea
    }
  }
}
</script>
```

Requerimiento 4

- **Paso 1:** Configurar el archivo que se encuentra dentro de la carpeta “router” con el nombre “index.js”. En este archivo se deben agregar los componentes que se utilizan como rutas.
- **Paso 2:** La importación de las rutas puede ser mediante lazy load, para realizar la carga perezosa de las rutas que no se están invocando al momento.
- **Paso 3:** La ruta para editar, debe ser una ruta dinámica que reciba parametros, en este caso utilizaremos el “id” que se genera cuando se crean las opiniones. Para lo cual nos apoyaremos con los props para hacer más rápido el manejo del dato que se envíe en la URL.
- **Paso 4:** Crear el componente en la carpeta de “view” para la ruta “Editar” con su estructura básica de template por los momentos.

- **Paso 5:** Crear una opción que permita capturar las rutas no encontradas mediante el path "*".
- **Paso 6:** Ahora debemos crear el componente en la carpeta "view" para la ruta "NotFound" y agregar una imagen o mensaje indicando el "404 not found".

Archivo index.js

```
import Vue from 'vue';
import VueRouter from 'vue-router';
import Home from '../views/Home.vue';

Vue.use(VueRouter)

const routes = [
  {
    path: '/',
    name: 'Home',
    alias: ['/inicio', '/home'],
    component: Home
  },
  {
    path: '/administracion',
    name: 'Administracion',
    component: () => import(/* webpackChunkName: "Administracion" */
'../views/Administracion.vue')
  },
  {
    path: '/editando/:id',
    name: 'Editando',
    props: true,
    component: () => import(/* webpackChunkName: "Editando" */
'../views/Editando.vue')
  },
  {
    path: '*',
    name: 'NotFound',
    component: () => import(/* webpackChunkName: "NotFound" */
'../views/NotFound.vue')
  }
]

const router = new VueRouter({
  mode: 'history',
  base: process.env.BASE_URL,
```



```
routes
  })

export default router
```

- **Paso 7:** Ahora, en el archivo “ListaAdministrada.vue”, en el método creado con el nombre “editando”, agregar la línea necesaria para llevar el usuario a la vista de edición. Para ello debemos utilizar el objeto “\$router”, que nos provee del “push” donde podemos indicar el nombre de la ruta y los parámetros que deseamos pasar.

Archivo ListaAdministrada.vue

```
.
.
.
  methods: {
    // el ítem es el elemento con toda la información, dentro de él se
    // encuentra un id que se utilizará como parámetro de la ruta dinámica.
    editando(item){
      this.$router.push({name: 'Editando', params: {id: item.id}});
    }
  }
.
.
.
.
```

- **Paso 8:** Ahora, trabajaremos con la vista Editando.vue que permitirá al usuario editar la opinión seleccionada. Para ello, se debe hacer uso de los props para recibir el dato enviado mediante la URL. Luego, en el ciclo de vida del componente, específicamente en el “created()” para comprobar si realmente ese “id” recibido existe. Si existe, cargamos los datos en las variables locales, de lo contrario, se deja un mensaje indicando el error. Agregando un botón con la opción para regresar a la ruta de administración.
- **Paso 9:** Agregar un formulario para mostrar los datos de la opinión que se desea modificar mediante un v-model y al mismo tiempo guardar los cambios mediante un botón que lleve a un método con el evento “click”.
- **Paso 10:** El método para guardar los cambios crea un objeto con los datos nuevos o existentes. Se deja el espacio para el llamado de la acción en el vuex que guardará los nuevos datos y se envía al usuario a la vista de administración.

Archivo Editando.vue

```
<template>
  <div class="container">
    <div v-if="juegoEditar">
      <h2 class="text-center mt-5">Editando la opinión de:
      {{juegoEditar.juego.name}}</h2>
      <form>
        <div class="mb-3">
          <label for="nombre" class="form-label">Nombre:</label>
          <input type="text" class="form-control" id="nombre"
v-model="juegoEditar.nombre">
        </div>
        <div class="mb-3">
          <label for="opiniones" class="form-label">Opiniones</label>
          <textarea class="form-control" id="opiniones" rows="3"
v-model="juegoEditar.opinion"></textarea>
        </div>
        <button type="button" class="btn btn-primary"
@click="$router.go(-1)">Regresar</button>
        <button type="button" class="btn btn-info mx-4"
@click.prevent="guardando">Guardar</button>
      </form>
    </div>
    <div v-else>
      <div class="alert alert-danger mt-5 text-center" role="alert">
        La opinión que deseas editar no existe.
        <button type="button" class="btn btn-primary"
@click="$router.push({name: 'Administracion'})">Regresar</button>
      </div>
    </div>
  </div>
</template>

<script>
export default {
  name: 'Editando',
  props: ['id'],
  data() {
    return {
      juegoEditar: null,
    }
  },
  created() {
```

```
//en este ciclo de vida se comprueba que el id que viene en el URL
pertenece a alguna opinión.
    let resultado = this.$store.getters.enviandoOpiniones.find(result
=> result.id === this.id);

    if (resultado !== undefined) {
    this.juegoEditar = resultado;
    } else {
    this.juegoEditar = null;
    }
  },
  methods: {
    guardando(){
//aquí se aplicará el llamado a la acción en el vuex para guardar los
cambios.
    this.$router.push({name: 'Administracion'});
    }
  }
}
</script>
```

Requerimiento 5

- **Paso 1:** Configurar el archivo que se encuentra dentro de la carpeta “store” con el nombre “index.js”. En este archivo se deben agregar los estados necesarios para el correcto funcionamiento del sitio web, uno para la información entregada por la API y otro para las opiniones realizadas por los usuarios.
- **Paso 2:** A cada estado creado se le debe crear un getter que entregue y deje disponible la información para cualquier componente.
- **Paso 3:** En las mutaciones, se creará una para mutar el estado de los juegos con la información recibida de la API, así como las mutaciones para crear, editar y eliminar opiniones.
- **Paso 4:** Mientras que en las acciones se debe crear las correspondientes a cada mutación. Teniendo en cuenta que la acción que se utilice para llamar la API puede ser con fetch o axios. Siendo ideal el empleo de async/await para recibir los datos de una manera más legible y ordenada.

Archivo index.js

```
import Vue from 'vue';
import Vuex from 'vuex';

Vue.use(Vuex);

export default new Vuex.Store({
  // en el state se tendrán las variables o estados para guardar las
  // opiniones creadas por los usuarios y los juegos entregados por la API.
  state: {
    opiniones: [],
    juegos: [],
  },
  // en los getters, dejaremos disponible el acceso a los estados para
  // cualquier componente o vista que necesite de ellos. Evitando así el
  // acceso directo al state.
  getters: {
    enviandoJuegos(state){
      return state.juegos;
    },
    enviandoOpiniones(state){
      return state.opiniones;
    }
  },
  // Las mutaciones servirán para modificar los estados. Sin tener que
  // acceder y modificar directamente un estado.
  mutations: {
    // mutación para modificar los juegos con los datos entregados por la
    // API.
    mutandoJuegos(state,datos) {
      state.juegos = datos.results;
    },
    // mutación para ir agregando las opiniones ingresadas por los usuarios.
    mutandoOpiniones(state, dataOpinion){
      let id = 1;
      while (id !== dataOpinion.id) {
        id = Math.floor((Math.random() * 100) + 1);
      }
      dataOpinion.id = id;
      state.opiniones.push(dataOpinion);
    },
    // mutación para ir borrando las opiniones.
    borrarOpinion(state,index){
```

```
        state.opiniones.splice(index,1);
    },
    // mutación para ir editando las opiniones.
    editandoOpinion(state,editado){
        let resultado = state.opiniones.find(valor => valor.id ===
editado.id);
        if (resultado !== undefined) {
            resultado.nombre = editado.nombre;
            resultado.opinion = editado.opinion;
        } else {
            alert("No se puede editar");
        }
    }
},
// las acciones servirán para ir activando cada mutación y ser llamadas
desde los componentes o vistas.
    actions: {
        // con esta acción nos traemos los datos de la API y se llama a la
mutación pertinente para almacenar los valores en el estado respectivo.
        async infoApi({commit}){
            let result = await fetch('https://api.rawg.io/api/games');
            let datos = await result.json();
            commit('mutandoJuegos',datos);
        },
        // con esta acción se llama a la mutación que permite guardar las
opiniones creadas por el usuario.
        guardandoOpinion({commit},dataOpinion){
            commit('mutandoOpiniones',dataOpinion);
        },
        // con esta acción se llama a la mutación que permite eliminar las
opiniones guardadas.
        eliminarOpinion({commit}, index){
            commit('borrarOpinion',index);
        },
        // con esta acción se llama a la mutación que permite editar las
opiniones guardadas.
        guardandoEdicion({commit},editado){
            commit('editandoOpinion',editado)
        }
    }
})
```

- **Paso 5:** Agregar los llamados a las acciones en los métodos y/o ciclos de vidas en los componentes requeridos, como lo son los componentes ListaAdministrada, Home, Editando y App.
- **Paso 6:** En el caso de la ListaAdminsitrada, se borra la data falsa y se trabaja con los valores traídos desde los getters. Por lo que debes modificar el template del componente cambiando la data falsa por el nuevo valor.

Archivo ListaAdministrada.vue

```
.  
.   
.   
<script>  
// se importan los mapas para los getters  
import { mapGetters } from "vuex";  
export default {  
  name: 'ListaAdministrada',  
  // se utilizan las propiedades computadas para traer la información de  
  // los getters de vuex  
  computed: {  
    ...mapGetters(['enviandoOpiniones']),  
  },  
  methods: {  
    //en este método se llama a la acción que permite eliminar una opinión  
    //en el vuex  
    eliminando(index){  
      let resultado = confirm("Seguro que desea eliminar");  
      if (resultado) {  
        this.$store.dispatch('eliminarOpinion', index);  
      }  
    },  
  },  
}  
</script>  
.   
.   
.
```

- **Paso 7:** Ahora en la "App.vue" se debe implementar el llamado a la acción de la store que se conecta con la API y trae la información para ser guarda en un estado.

Archivo App.vue

```
.  
.   
.   
// se utiliza el ciclo de vida para activar la acción que se comunica  
con la API.  
  created() {  
    this.$store.dispatch('infoApi');  
  }  
.   
.   
. 
```

- **Paso 8:** Ahora en el "Home.vue" se debe implementar el uso de las propiedades computadas para traer la información desde los getters. En este caso, la información de los juegos. Recuerda que debes borrar todas las rutinas para conectarnos a la API de forma momentánea, creadas en los pasos anteriores.

Archivo Home.vue

```
.   
.   
.   
<script>  
// se importan los mapas para los getters  
import { mapGetters } from "vuex";  
export default {  
  name: 'Home',  
  data() {  
    return {  
      nombre: '',  
      opiniones: '',  
    }  
  },  
  // se utilizan las propiedades computadas para traer la información de  
  los getters de vuex  
  computed: {  
    ...mapGetters(['enviandoJuegos']),  
  },  
  methods: {  
    guardarOpinion(juego){  
      if (this.nombre && this.opiniones) {
```

```
// se debe crear un objeto con los datos ingresados por el usuario al
momento de crear la opinión, para luego ser enviado en el llamado a la
acción en la store.
    let opinionUser = {
      nombre: this.nombre,
      opinion: this.opiniones,
      juego: juego,
      id: Math.floor((Math.random() * 100) + 1)
    };
// se emplea el objeto $store de vuex para hacer el llamado a la acción
mediante un dispatch
    this.$store.dispatch('guardandoOpinion', opinionUser);
    alert("Opinión agrega exitosamente");
    this.nombre = "";
    this.opiniones = "";
  } else {
    alert("Error al ingresar los datos.");
  }
}
}
</script>
.
.
.
```

- **Paso 9:** Ahora en el “Editando.vue”, se debe implementar el uso del ciclo de vida para traer la información a editar y verificar que realmente exista. igualmente, hacer el llamado a la acción que permite mutar el estado de las opiniones, pasando los nuevos valores guardados.

Archivo Editando.vue

```
.
.
.
<script>
export default {
  name: 'Editando',
  props: ['id'],
  data() {
    return {
      juegoEditar: null,
```



```
    }  
  },  
  // se emplea el objeto $store de vuex para hacer el llamado al estado  
  // mediante un getters en el ciclo de vida del componente.  
  created() {  
    let resultado = this.$store.getters.enviandoOpiniones.find(result  
=> result.id === this.id);  
  
    if (resultado !== undefined) {  
      this.juegoEditar = resultado;  
    } else {  
      this.juegoEditar = null;  
    }  
  },  
  methods: {  
    guardando(){  
      // se debe crear un objeto con los datos ingresados por el usuario al  
      // momento de editar la opinión para luego ser enviado en el llamado a la  
      // acción en la store.  
      let juegoEditado = {  
        nombre: this.juegoEditar.nombre,  
        opinion: this.juegoEditar.opinion,  
        juego: this.juegoEditar.juego,  
        id: this.juegoEditar.id  
      };  
      this.$router.push({name: 'Administracion'});  
      // se emplea el objeto $store de vuex para hacer el llamado a la acción  
      // mediante un dispatch  
      this.$store.dispatch('guardandoEdicion', juegoEditado);  
    }  
  }  
}  
</script>  
.  
.  
.
```

Requerimiento 6

- **Paso 1:** Crear un nuevo archivo dentro de la carpeta “tests/unit” con el nombre, por ejemplo, de testComponente.spec.js. En este archivo escribiremos nuestro test unitario para comprobar que el NavBar posea la ruta correcta, que las palabras “Games Opinión” tengan 13 caracteres y que exista la palabra “Administración” en el menú de navegación.
- **Paso 2:** Importar el componente del navbar, luego Vue, seguidamente los objetos mount y createLocalVue para realizar el montaje del componente y hacer una instancia local de Vue que nos servirá para agregar VueRouter y no tener advertencias sobre las rutas.
- **Paso 3:** Importar VueRouter y los objetos de las dependencias externas para los iconos. En este caso, como se implementaron iconos de FontAwesome en el componente, se deben también inicializar para que no genere advertencias el test.

Archivo testComponente.spec.js

```
import TheNavBar from '@components/TheNavBar.vue';
import Vue from 'vue';
import { mount, createLocalVue } from '@vue/test-utils';
import VueRouter from 'vue-router';
import { library } from '@fortawesome/fontawesome-svg-core';
import { faUsersCog, faComments, faHome, faHeadset } from
'@fortawesome/free-solid-svg-icons';
import { FontAwesomeIcon } from '@fortawesome/vue-fontawesome';

library.add(faUsersCog);
library.add(faComments);
library.add(faHome);
library.add(faHeadset);

Vue.component('font-awesome-icon', FontAwesomeIcon);
```

- **Paso 4:** Crear la nueva instancia de Vue con “createLocalVue()”, seguidamente se le indica a la instancia local de Vue que implementaremos VueRouter.
- **Paso 5:** Ahora creamos nuestras rutas para evitar las advertencias, pero sin importar los componentes.

Archivo testComponente.spec.js

```
import TheNavBar from '@components/TheNavBar.vue';
import Vue from 'vue';
import { mount, createLocalVue } from '@vue/test-utils';
import VueRouter from 'vue-router';
import { library } from '@fortawesome/fontawesome-svg-core';
import { faUsersCog, faComments, faHome, faHeadset } from
'@fortawesome/free-solid-svg-icons';
import { FontAwesomeIcon } from '@fortawesome/vue-fontawesome';

library.add(faUsersCog);
library.add(faComments);
library.add(faHome);
library.add(faHeadset);

Vue.component('font-awesome-icon', FontAwesomeIcon);

const localVue = createLocalVue();
localVue.use(VueRouter);
const router = new VueRouter({
  routes: [
    {
      path: '/',
      name: 'Home',
    },
    {
      path: '/opiniones',
      name: 'Opiniones',
    },
    {
      path: '/administracion',
      name: 'Administracion',
    }
  ]
});
```

- **Paso 6:** Ya se tiene configurado el archivo para el test, ahora si se puede iniciar la construcción del test unitario. Por lo que se utilizará la palabra reservada “describe” para indicar el test, y la palabra reservada “it” para indicar el inicio del test.
- **Paso 7:** Luego, dentro del objeto “it”, se debe montar el componente pasando como valores la instancia local de vue y de vue router.
- **Paso 8:** Utilizar la palabra reservada “expect” para aplicar los distintos test.

Archivo testComponente.spec.js

```
import TheNavBar from '@components/TheNavBar.vue';
import Vue from 'vue';
import { mount, createLocalVue } from '@vue/test-utils';
import VueRouter from 'vue-router';
import { library } from '@fortawesome/fontawesome-svg-core';
import { faUsersCog, faComments, faHome, faHeadset } from
'@fortawesome/free-solid-svg-icons';
import { FontAwesomeIcon } from '@fortawesome/vue-fontawesome';

library.add(faUsersCog);
library.add(faComments);
library.add(faHome);
library.add(faHeadset);

Vue.component('font-awesome-icon', FontAwesomeIcon);

const localVue = createLocalVue();
localVue.use(VueRouter);
const router = new VueRouter({
  routes: [
    {
      path: '/',
      name: 'Home',
    },
    {
      path: '/opiniones',
      name: 'Opiniones',
    },
    {
      path: '/administracion',
      name: 'Administracion',
    }
  ]
});

describe('Home.vue', () => {
  it('existe información en el navBar y la ruta es correcta', () => {
    const wrapper = mount(TheNavBar, {
      localVue,
      router
    });
    // aquí se espera que la longitud del texto que posee la clase
```

```
“navbar-brand” sea de 13 caracteres.  
    expect(wrapper.find('.navbar-brand').text().length).toBe(13);  
    // aquí se espera que el componente contenga la palabra  
    “Administración”.  
    expect(wrapper.text()).toContain('Administración');  
    // aquí se espera que la ruta del componente se encuentra en el path  
    “/”.  
    expect(wrapper.vm.$route.path).toBe('/');  
  })  
})
```

- **Paso 9:** Crear un nuevo archivo dentro de la carpeta “tests/unit” con el nombre, por ejemplo, de testStore.spec.js. En este archivo escribiremos nuestro test unitario para comprobar algunas acciones, mutaciones y estados en la store de vuex.
- **Paso 10:** Importar Vue, luego Vuex y finalmente el archivo que contiene la store. Lo cual nos permitirá inicializar Vuex en el test y tener acceso a los objetos de la store.

Archivo testStore.spec.js

```
import Vue from 'vue';  
import Vuex from 'vuex';  
import store from '@store/store';
```

- **Paso 11:** Crear la nueva instancia de Vuex, indicando a Vue que debe implementarla y utilizarla mediante la store.

Archivo testStore.spec.js

```
import Vue from 'vue';  
import Vuex from 'vuex';  
import store from '@store/store';  
  
describe('Prueba a la Store-VUEX', () => {  
  beforeEach(() => {  
    Vue.use(Vuex);  
    store = new Vuex.Store(store);  
  });  
});
```

- **Paso 12:** Crear los test dentro del mismo archivo que permitan agregar una nueva opinión. Para esto se puede crear localmente una variable con el objeto que contenga los datos de la opinión, llamar a la acción mediante el dispatch y verificar

que el estado y el getters del estado tenga una longitud de uno, ya que se cargará una sola opinión .

- **Paso 13:** Finalmente, se puede comprobar la edición correcta de una opinión. Cargando un nuevo objeto con la opinión anterior modificada y llamando a la acción correspondiente. Luego se verifica que el estado contenga los nuevos valores.

Archivo testStore.spec.js

```
import Vue from 'vue';
import Vuex from 'vuex';
import store from '@store/store';

describe('Prueba a la Store-VUEX', () => {
  beforeEach(() => {
    Vue.use(Vuex);
    store = new Vuex.Store(store);
  });
});

describe('Prueba store', () => {
  it('agregando una opinión', () => {

    let nueva_opinion = {
      nombre: "Antonio",
      opinion: "Buen Juego",
      juego: {
        name: "Juego 1",
        id: 1234,
      },
      id: 5
    }

    // aquí se activa la acción para guardar la opinión enviando el objeto
    // con la información.
    store.dispatch('guardandoOpinion', nueva_opinion);
    // aquí se espera que la longitud del arreglo en el estado que contiene
    // las opiniones sea de uno (1), ya que se envió una sola opinión.
    expect(store.state.opiniones).toHaveLength(1);
    expect(store.getters.enviandoOpiniones).toHaveLength(1);
  });

  it('editando opinión', () => {
    let opinionEditada = {
      nombre: "Martha",
      opinion: "Nuevo Comentario 2.0",
    }
  });
});
```

```
        juego: {
          name: "Juego 1",
          id: 1234,
        },
        id: 5
      };
    // aquí se activa la acción para guardar la opinión editada, enviando el
    // objeto con la información modificada.
    store.dispatch('guardandoEdicion', opinionEditada);
    // aquí se espera que el estado contenga la propiedad "opinion" con el
    // nuevo valor agregado.
    expect(store.state.opiniones[0]).toHaveProperty('opinion', 'Nuevo
    Comentario 2.0');
  });
});
```

Requerimiento 7

- **Paso 1:** Eliminar cualquier archivo de prueba existente instalado por defecto.
- **Paso 2:** Crear un nuevo archivo dentro de la carpeta "tests/e2e/specs" con el nombre, por ejemplo, de testRutas.js. En este archivo escribiremos nuestro test global para comprobar las rutas de navegación. Para estos test no es necesario importar nada por los momentos.
- **Paso 3:** Agregar los identificadores necesarios a los elementos que se utilizarán en las pruebas. Se recomienda utilizar el "data-cy" con su descriptor como atributo único para las pruebas con Cypress.

Archivo testRutas.js

```
describe('Primer test global', () => {
  it('Visitando Home', () => {
    cy.visit('/')
    cy.contains('h1', 'Lista de Juegos Disponibles');
    cy.location('href').should('equal', 'http://localhost:8080/');
    cy.pause();
  });
  it('Visitando Administración', () => {
    cy.visit('/administracion')
    cy.contains('h2', 'Administrando la Lista de Opiniones');
    cy.get('[data-cy="sinAdministrar"]').should('be.visible');
    cy.get('[data-cy="administrandoLista"]').should('not.exist');
  });
});
```

```
cy.location().should((loc) => {  
  expect(loc.pathname).to.eq('/administracion');  
});  
});  
})
```

- **Paso 4:** Crear un nuevo archivo dentro de la carpeta "tests/e2e/specs" con el nombre, por ejemplo, de testComponentes.js. En este archivo escribiremos nuestro test global para comprobar el funcionamiento de algunos componentes.

Archivo testComponentes.js

```
describe('Primer test global', () => {  
  it('Visitando Home', () => {  
    cy.visit('/')  
    cy.get('[data-cy="opinando1"]').click();  
    let nombre = "Maria";  
    let opinion = "Muy buen juego";  
    cy.pause();  
    cy.get('#nombre1').type(nombre).should('have.value', 'Maria');  
    cy.get('#opiniones1').type(opinion).should('have.value', opinion);  
    cy.pause();  
    cy.get('[data-cy="guardarOpinion1"]').click();  
    cy.pause();  
  });  
  it('Visitando Administración', () => {  
    cy.get('[data-cy="administraClick"]').click();  
    cy.contains('td', 'Maria');  
    cy.pause();  
    cy.get('[data-cy="eliminarOpinion0"]').click();  
    cy.pause();  
    cy.on('window:confirm', (str) => {  
      expect(str).to.eq('Seguro que desea eliminar');  
    });  
  });  
});  
})
```


Requerimiento 8 (Opcional)

- **Paso 1:** Crear un nuevo archivo dentro de la carpeta “components” con el nombre de ListaOpiniones.vue. que se mostraran las opiniones creadas por el usuario, y en el caso de no existir, debe mostrar un mensaje indicando que no existen opiniones. Para ellos se usarán las directivas v-for, v-if y v-bind. Todo esto se realizará en el template en conjunto con la estructura HTML del acordeón.
- **Paso 2:** En el script se utilizarán las propiedades computadas para traer las opiniones desde la store mediante los getters. En el caso de utilizar los mapas de Vuex, se debe importar el mapa respectivo antes de emplearlo.

Archivo ListaOpiniones.vue

```
<template>
  <div class="container">
    <h2 class="text-center mt-5">Lista de Opiniones</h2>
    <div v-if="enviandoOpiniones.length > 0"
data-cy="opinionesVisible">
      <div class="accordion" id="accordionExample">
        <div class="accordion-item mt-4" v-for="(item, index)
in enviandoOpiniones" :key="index">
          <h2 class="accordion-header" :id="'headingOne'+index">
            <button class="accordion-button" type="button"
data-bs-toggle="collapse" :data-bs-target="'#collapseOne'+index"
aria-expanded="true" :aria-controls="'collapseOne'+index">
              Opinión creada por: <span
:data-cy="'nombre'+index">{{item.nombre}}</span>. Para el juego:
{{item.juego.name}}
            </button>
          </h2>
          <div :id="'collapseOne'+index"
class="accordion-collapse collapse show"
:aria-labelledby="'headingOne'+index"
data-bs-parent="#accordionExample">
            <div class="accordion-body">
              <strong>Opinión: </strong> {{item.opinion}}
            </div>
          </div>
        </div>
      </div>
    <div class="alert alert-danger mt-5" role="alert" v-else
```

```
data-cy="noOpiniones">
    No existen opiniones para mostrar.
</div>
</div>
</template>

<script>
import { mapGetters } from "vuex";

export default {
  name: 'ListaOpiniones',
  computed: {
    ...mapGetters(['enviandoOpiniones']),
  }
}
</script>
```

Requerimiento 9 (Opcional)

- **Paso 1:** Se cuenta con el componente hijo creado para la vista de opiniones es momento de crear el componente padre en la carpeta de "views", con el nombre: "Opiniones.vue".
- **Paso 2:** En el componente padre se debe importar el componente hijo, que fue creado en el requerimiento anterior con el nombre: "ListaOpiniones.vue".

Archivo Opiniones.vue

```
<template>
  <div>
    <lista-opiniones></lista-opiniones>
  </div>
</template>

<script>
import ListaOpiniones from '@components/ListaOpiniones.vue'
export default {
  name: 'Opiniones',
  components: {
    ListaOpiniones
  },
}
```

```
</script>
```

- **Paso 3:** Declarar y activar la ruta con su componente en el archivo router.js. Para ello se debe utilizar el path “/opiniones”.

Archivo router.js

```
.  
.   
.   
  
const routes = [  
  {  
    path: '/opiniones',  
    name: 'Opiniones',  
    component: () => import(/* webpackChunkName: "Opiniones" */  
      '../views/Opiniones.vue')  
  },  
]  
  
.   
.   
. 
```

Requerimiento 10 (Opcional)

- **Paso 1:** Realizar un test unitario a la store de Vuex, por lo cual, podemos seguir implementando la primera prueba realizada denominada “testStore.spec.js”. En ella debemos agregar ahora un nuevo “It” con las instrucciones necesarias para borrar la opinión creada y editada en la prueba hasta el momento. Por lo tanto, el dispatch se realizará a la acción con el nombre de “eliminarOpinion”, pasando como payload el valor de la posición donde se encuentra la opinión a borrar. En este caso, como se creó y editó una sola opinión, se borrará la posición 0.

Archivo testStore.spec.js

```
.  
.   
.   
it('eliminando opinión',()=>{  
    const index = 0;  
    store.dispatch('eliminarOpinion',index);  
    expect(store.state.opiniones).toHaveLength(0);  
});  
.   
.   
.
```

Requerimiento 11 (Opcional)

- **Paso 1:** Crear esta prueba global, trabajaremos sobre el mismo archivo creado anteriormente, denominado “testRutas.js”. Por lo que solo debemos agregar el “It” con el nombre y código respectivo que nos permita visitar la ruta, comprobar que existe un título en una etiqueta “h2” con el texto “Lista de Opiniones”, y comprobar que la URL es la correcta.

Archivo testRutas.js

```
describe('Primer test global', () => {  
.   
.   
.   
    it('Visitando Opiniones', () => {  
        cy.visit('/opiniones')  
        cy.contains('h2', 'Lista de Opiniones');  
        cy.get('[data-cy="noOpiniones"]').should('be.visible');  
        cy.get('[data-cy="opinionesVisible"]').should('not.exist');  
        cy.url().should('include', '/opiniones');  
        cy.pause();  
    });  
.   
.   
.   
})
```

- **Paso 2:** Al igual que el paso anterior, vamos a utilizar el test global creado anteriormente con el nombre: “testComponentes.js”. En él debemos crear nuestra estructura “it” con un nombre y el bloque de código necesario para comprobar que existe una opinión en el acordeón en ese preciso momento con el nombre del usuario agregado al crear la opinión en la vista de Home.

Archivo testComponentes.js

```
describe('Primer test global', () => {  
  .  
  .  
  .  
    it('Visitando Opiniones', () => {  
      cy.get('[data-cy="opinionesClick"]').click();  
      cy.get('[data-cy="noOpiniones"]').should('not.exist');  
      cy.get('[data-cy="opinionesVisible"]').should('be.visible');  
      cy.get('[data-cy="nombre0"]').should('contain', 'Maria');  
      cy.pause();  
    });  
  .  
  .  
  .  
})
```

Requerimiento 12 (Opcional)

- **Paso 1:** Para la realización de este requerimiento opcional, se implementó la librería “vuex-persistedstate”, que permite mantener los estados en la localStorage del navegador mientras el usuario no los borre. La documentación se encuentra disponible en: <https://www.npmjs.com/package/vuex-persistedstate>. El uso e implementación es muy sencillo y se encuentra muy bien detallado.
- **Paso 2:** En la terminal de tu computador, y dentro de la carpeta raíz del proyecto, se debe instalar primeramente la dependencia mediante el comando: “npm install --save vuex-persistedstate” o “yarn add -D vuex-persistedstate”.
- **Paso 3:** Después de instalada, se debe importar en el archivo store.js, luego, implementar dentro de la instancia de Vuex.

Archivo store.js

```
import Vue from 'vue';
import Vuex from 'vuex';
import createPersistedState from 'vuex-persistedstate';

Vue.use(Vuex);

export default new Vuex.Store({
  state: {
    opiniones: [],
    juegos: [],
  },
  plugins: [createPersistedState()],
  getters: {
    enviandoJuegos(state){
      return state.juegos;
    },
    enviandoOpiniones(state){
      return state.opiniones;
    }
  },
  .
  .
  .
}
```