

Contenido

Requerimientos de la prueba:	2
Composición del proyecto	2
Estructura del proyecto.....	3
Carpeta src.....	3
Carpeta assets.....	3
Carpeta components	3
Menú.vue	3
Carpeta Router	4
index.js.....	4
Carpeta static.....	4
user.js.....	4
Carpeta views.....	4
HomeView.vue	4
Dashboard.vue.....	6

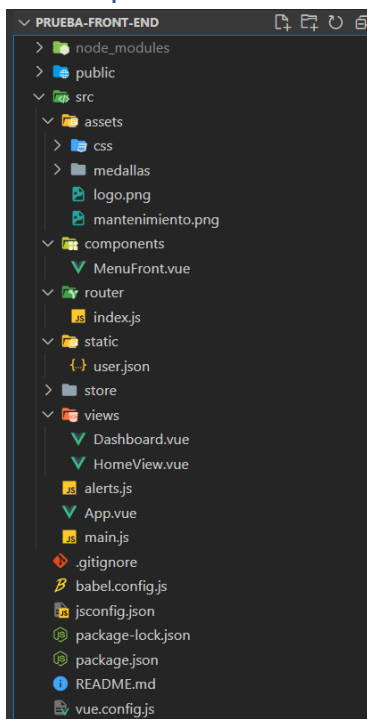
Prueba Desarrollador Front-End

Requerimientos de la prueba:

Diseñar un login que al ingresar un usuario pueda validar dos tipos de usuario: Administrador y Coordinador.

1. Al ingresar se debe mostrar una barra superior que tenga a la derecha la información de quien ingresó (Nombre, Tipo de usuario) y a la izquierda un ícono de la aplicación (Cualquier icono puede colocarse).
2. Si el tipo de usuario es Coordinador, debe mostrar una imagen en el resto de la pantalla que identifique que la página está en mantenimiento.
3. Si el tipo de usuario es Administrador, debe mostrar dos pestañas:
 1. En la primera se debe hacer una petición GET a la siguiente endpoint: <https://raw.githubusercontent.com/ag-grid/ag-grid/master/grid-packages/ag-grid-docs/src/olympicWinners.json>; posteriormente se debe crear una tabla que contenga las siguientes columnas: Atleta, Edad, País, Año, Fecha, Deporte, Oro, Plata, Bronce, Total.
 2. En la segunda pestaña se debe mostrar tres categorías informativas distribuidas uniformemente (Oro, plata, y bronce) cada uno con un icono y la suma total correspondiente a la categoría. Al darle click a alguna categoría debe desplegar una ventana emergente que muestre una descripción (cualquier texto) de la categoría.
4. Finalmente, un botón de logout que me permita retornar al login para probar otro usuario.

Composición del proyecto



Este proyecto tiene como finalidad el resolver la prueba solicitada para la vacante de Desarrollador Frontend donde se requiere persona con dominio sólido de las tecnologías web como HTML, CSS, JAVASCRIPT, TYPESCRIPT, VUEJS, consumo de APIs, manejo de respuestas de control de errores adicionalmente se deberán aplicar las buenas prácticas de desarrollo UI y UX, debe tener conocimiento en SEO, Google Analytics, Google Console, Figma, herramientas de los navegadores, inspeccionar, debuguear y simular distintos tipos de conexiones.

El proyecto está compuesto por la estructura que se puede observar en la imagen a la izquierda.

El proyecto fue diseñado de acuerdo con los requerimientos solicitados y cumple al pie de la letra.

Estructura del proyecto

La ejecución del proyecto usa librerías las cuales son necesarias para su correcto funcionamiento, dichas librerías son las siguientes:

1. Sweetalert2
2. Bootstrap 5
3. Axios
4. datatables.net
5. datatables.net-vue3
6. datatables.net-bs5
7. datatables.net-buttons-bs5
8. datatables.net-responsive-bs
9. jszip
10. pdfmake

Carpeta src

Esta carpeta contiene toda la información de la aplicación y a su vez está compuesta por subcarpetas las cuales poseen cada una archivos con funciones distintas.

Carpeta assets

Esta carpeta está compuesta por un archivo con el nombre **logo.png** una imagen con nombre **mantenimiento.png** y una subcarpeta de nombre **css** la cual contiene un archivo de nombre **cssPersonal.css** donde se encuentran estilos personalizados para el proyecto y una carpeta de nombre **medallas** la cual contiene las 3 imágenes de oro, plata y bronce.

Carpeta components

En esta carpeta se guardan los componentes que se requieren para la ejecución de ciertas funcionalidades en el aplicativo.

Dentro de esta carpeta se creó el componente **Menu.vue**.

Menú.vue

Este componente está compuesto por un menú que se va a renderizar en todas las vistas en las cuales sea invocado y que mostrara en la parte izquierda la imagen **logo.png** y a la derecha el nombre del usuario que se loguea, el rol que tiene asignado y el botón para cerrar la sesión abierta actualmente.

El componente menú para poder mostrar el nombre y el rol del usuario logueado usa las siguientes variables:

1. nombre:
2. rol

Para poder mostrar la información de estas variables se está consultando al **localStorage** que es donde se almacenan temporalmente los datos del usuario.

Carpeta Router

Esta carpeta contiene el archivo **index.js**.

index.js

Este archivo encarga del ruteo de las diferentes vistas de la aplicación para navegar de acuerdo hacia donde se quiera desplazar el usuario, la ruta principal se encarga de ir siempre a la vista **HomeView** que es la que contiene el formulario para que el usuario pueda loguearse.

Carpeta static

Esta carpeta contiene el archivo **user.json**.

user.js

Este archivo está compuesto por un json el cual contiene 4 usuarios para realizar pruebas de inicio de sesión en la aplicación.

Usuario: jesus.sarmiento@prueba.com Rol: **Administrador**

Contraseña: Prueba.2023

Usuario: jair.salgado@prueba.com Rol: **Coordinador**

Contraseña: Prueba.2024

Usuario: emilio.cortes@prueba.com Rol: **Administrador**

Contraseña: Prueba.2025

Usuario: bladimir.hernandez@prueba.com Rol: **Coordinador**

Contraseña: Prueba.2026

Carpeta views

Esta carpeta está compuesta por dos vistas, la vista **Dashboard.vue** y **HomeView.vue** las cuales son las necesarias para la funcionalidad del proyecto.

HomeView.vue



The screenshot shows a login form with the SmartQuick logo at the top. Below the logo, the text 'Ingrese su usuario y contraseña' is displayed. There are two input fields: 'Ingrese su correo' and 'Ingrese su contraseña'. Below these fields is a blue button labeled 'Iniciar Sesión'. At the bottom, there is a copyright notice '© 2023-2023'.

Esta vista es la vista principal y contiene el formulario para iniciar sesión el formulario está compuesto por dos campos: correo y contraseña, cuando el usuario se loguea de manera incorrecta salta una alerta la cuál es mostrada por medio de una ventana emergente estilizada por medio de **sweetalert2** informando que el correo o la contraseña son incorrectos, de lo contrario salta una alerta informando que se inicio sesión correctamente y enviando el usuario hacia el dashboard.

Los datos del usuario cuando ingresa correctamente son almacenados en el **localStorage** del navegador

temporalmente, esto debido a que no se puede conectar a ninguna API externa para el inicio de sesión y se requiere mantener los datos para que se pueda conservar la información y después ser consultada como en este caso por el componente Menú para mostrar los datos del usuario logueado, los datos hardcodeados en el localStorage son los siguientes:

1. correo
2. rol
3. nombre

Cuando el usuario ingresa sus datos se ejecuta un método de nombre **login()** dentro de este método se puede observar una constante de nombre **mydata** la cual obtiene los datos obtenidos del **user.json** y los parsea en json para asegurar de que estén en dicho formato, después se recorre dentro de un for el cual se encarga de recorrer el json y extraer los datos para después ser comparados por un if donde se obtiene un json que recibe los datos ingresados por el usuario en los campos correo y contraseña y compara que dentro los datos ingresados correspondan al mismo usuario, después de funcionar la condicional correctamente valida que sea menor que la variable valid y la detiene con un break para no seguir ejecutando el for, después esos datos pasan a una condicional nueva la cual valida que si valid es igual a i es porque el usuario y la contraseña son correctos y almacena los datos en el localStorage y muestra una alerta informando que se logueo correctamente de lo contrario dice que el usuario y la contraseña son incorrectos.

```

methods: {
  login() {
    // const urlActual = window.location.href;
    let json = {
      "correo": this.correo,
      "contrasena": this.contrasena
    };

    let valid = -1;
    const mydata = JSON.parse(JSON.stringify(datos));
    // console.log(mydata)

    for (var i = 0; i < mydata.length; i++) {
      if ((json['correo'] == mydata[i].email) && (json['contrasena'] == mydata[i].password)) {
        valid = i;
        break;
      }
    }

    if (valid != -1) {
      show_alert('success', 'Excelente! <br>Se ha logueado correctamente')
      localStorage.setItem('nombre', mydata[i].nombre);
      localStorage.setItem('correo', mydata[i].email);
      localStorage.setItem('rol', mydata[i].rol);
      this.$router.push('/dashboard');
    } else {
      show_alert('error', 'Oops... <br>Usuario o contraseña incorrecto')
    }
  },
}

```

Dashboard.vue

Esta vista se encarga de recibir los datos ingresados por el usuario y renderizar la información de acuerdo al rol del usuario logueado, si el usuario logueado es un coordinador mostrara la siguiente vista:



Como se puede observar en la imagen se muestra el nombre del usuario logueado, su rol y el botón para cerrar la sesión abierta actualmente.

De acuerdo con el requerimiento cuando el usuario logueado es un Coordinador se debe mostrar una imagen grande informando que el sitio se encuentra en mantenimiento.

Para que este apartado sea renderizado se realizó lo siguiente:

```

mounted() {
  if (localStorage.getItem('correo') && localStorage.getItem('rol') == 'Administrador') {
    this.getAtletas();
    this.rol = localStorage.getItem('rol');
  } else if (localStorage.getItem('correo') && localStorage.getItem('rol') == 'Coordinador') {
    this.rol = localStorage.getItem('rol');
  } else {
    this.$router.push('/')
  }
},

```

Dentro del mounted() de vue se valida consultando el localStorage obteniendo el item 'correo' y el 'rol' y se compara si es administrador o coordinador, si es coordinador se almacena el rol en la variable rol definida dentro de la data para luego ser validada con un v-else-if el cuál compara que la variable rol sea igual a 'Coordinador', como se puede observar:

```
<div v-else-if="rol == 'Coordinador'" class="text-center">
  <div class="card shadow">
    <div class="card-body">
      
    </div>
  </div>
</div>
<div v-else class="">
  <div class="card shadow">
    <div class="card-body">
      <p>Error, no se ha encontrado información</p>
    </div>
  </div>
</div>
```

De lo contrario si es Administrador se renderizará la siguiente información:

SmartQuick Desarrollado Por Jesús Sarmiento Jesús Sarmiento Administrador Cerrar Sesión

Listado de Atletas Categoria de Medallas

Excel PDF Imprimir Copiar Texto

Ver 10 entradas

Buscar

#	Atleta	Edad	Pais	Año	Fecha	Deporte	Oro	Plata	Bronce	Total
1	Michael Phelps	23	United States	2008	24/08/2008	Swimming	8	0	0	8
2	Michael Phelps	19	United States	2004	29/08/2004	Swimming	6	0	2	8
3	Michael Phelps	27	United States	2012	12/08/2012	Swimming	4	2	0	6
4	Natalie Coughlin	25	United States	2008	24/08/2008	Swimming	1	2	3	6
5	Aleksey Nemov	24	Russia	2000	01/10/2000	Gymnastics	2	1	3	6
6	Alicia Coutts	24	Australia	2012	12/08/2012	Swimming	1	3	1	5
7	Missy Franklin	17	United States	2012	12/08/2012	Swimming	4	0	1	5
8	Ryan Lochte	27	United States	2012	12/08/2012	Swimming	2	2	1	5
9	Allison Schmitt	22	United States	2012	12/08/2012	Swimming	3	1	1	5
10	Natalie Coughlin	21	United States	2004	29/08/2004	Swimming	2	2	1	5

Como se puede observar se muestra el nombre del usuario el cargo y el cierre de sesión, en la parte de abajo se pueden observar las dos pestañas solicitadas una con el Listado de todos los atletas y la otra con la Categoría de las medallas,

Para mostrar la información se hace uso de la librería de DataTables con las otras librerías necesarias las cuales permiten mostrar unos botones que son configurados para exportar la información a un archivo Excel, Pdf o imprimirla o bien ya sea copiarla y pasarla a un txt.

Estos botones se muestran configurando el DataTable de la siguiente manera:

```
<DataTable :data="atletas" :columns="columns"
  class="table table-striped table-bordered table-hover display" :options="{
    responsive: true, autoWidth: false, dom: 'Blfrtip', pageLength: 10, language: {
      sLengthMenu: 'Ver _MENU_ entradas', show: 'Ver', search: 'Buscar', zeroRecords: 'No hay registros para mostrar',
      sInfoEmpty: 'Mostrando _START_ de _END_ de _TOTAL_ registros',
      info: 'Mostrando registros _START_ a _END_ de _TOTAL_ registros', infoFiltered: '(Filtrados de _MAX_ registros.)',
      paginate: { first: 'Primero', previous: 'Anterior', next: 'Siguiente', last: 'Último' }
    }, buttons: botones
  }">
  <thead class="overflow-scroll">
    <tr>
      <th>#</th>
      <th>Atleta</th>
      <th>Edad</th>
      <th>País</th>
      <th>Año</th>
      <th>Fecha</th>
      <th>Deporte</th>
      <th>Oro</th>
      <th>Plata</th>
      <th>Bronce</th>
      <th>Total</th>
    </tr>
  </thead>
</DataTable>
```

Para mostrar el listado de los atletas se configuro de la siguiente manera:

```
data() {
  // console.log(window.location.href);
  return {
    atletas: null,
    rol: "",
    oro: "Esta es la ventana emergente de Oro",
    plata: "Esta es la ventana emergente de Plata",
    bronce: "Esta es la ventana emergente de Bronce",
    columns: [
      {
        data: null, render: function (data, type, row, meta) { return `${meta.row + 1}` }
      },
      { data: 'athlete' },
      { data: 'age' },
      { data: 'country' },
      { data: 'year' },
      { data: 'date' },
      { data: 'sport' },
      { data: 'gold' },
      { data: 'silver' },
      { data: 'bronze' },
      { data: 'total' },
    ],
  },
}
```

Como se puede observar se creo una variable atletas la cual se declara nula porque la información se le pasara después y se creó un arreglo de columnas el cuál contendrá los campos que se quieren mostrar de los datos obtenidos de la API.

Los datos obtenidos de la API se obtuvieron de la siguiente manera:

Se creo un meto de nombre getAtletas() el cuál por medio de un async await se hace consulta a la API y se parsean los datos en la variable atletas para mostrar la información en el arreglo de columnas que se renderiza con el datatable.


```

methods: {
  // getAtletas() {
  //   const respuesta = axios.get('https://raw.githubusercontent.com/ag-grid/ag-grid/master/grid-packages/ag-grid-docs/src/olympicWinners.json').then(
  //     response => {
  //       this.atletas = response.data
  //     },
  //   );
  //   const datosObtenidos = JSON.parse(JSON.stringify(respuesta));
  //   console.log(datosObtenidos)
  // },
  async getAtletas() {
    const respuesta = await fetch('https://raw.githubusercontent.com/ag-grid/ag-grid/master/grid-packages/ag-grid-docs/src/olympicWinners.json')
    this.atletas = await respuesta.json();

    const datosObtenidos = this.atletas;




    class sumaMedallas extends Array {
      sum(key) {
        return this.reduce((a, b) => a + (b[key] || 0), 0);
      }
    }
    const resultadoObtenido = new sumaMedallas(...datosObtenidos);

    this.sumaGold = resultadoObtenido.sum('gold');
    this.sumaSilver = resultadoObtenido.sum('silver');
    this.sumaBronze = resultadoObtenido.sum('bronze');
  },
  alertaMedallas(oro, plata, bronce){
    descripcionMedallas(oro, plata, bronce);
  }
}

```

Para la pestaña de la categoría medallas se obtiene lo siguiente:

[Listado de Atletas](#)
[Categoría de Medallas](#)

#	Medalla	Cantidad
1	 Oro	3143
2	 Plata	3131
3	 Bronce	3255

Como se puede observar se muestran los nombres de las medallas los iconos y la cantidad de medallas correspondientes, para ello se realizó lo siguiente:

```

async getAtletas() {
  const respuesta = await fetch('https://raw.githubusercontent.com/ag-grid/ag-grid/master/grid-packages/ag-grid-docs/src/olympicWinners.json')

  this.atletas = await respuesta.json();

  const datosObtenidos = this.atletas;

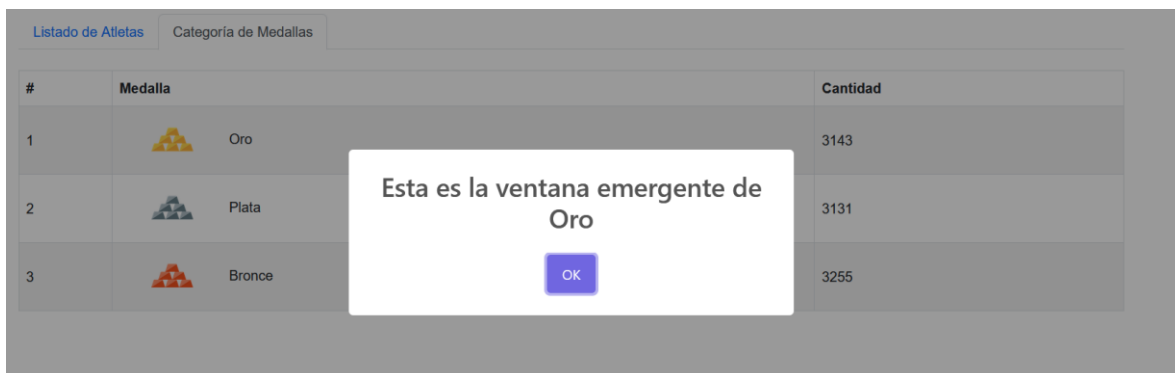
  class sumaMedallas extends Array {
    sum(key) {
      return this.reduce((a, b) => a + (b[key] || 0), 0);
    }
  }
  const resultadoObtenido = new sumaMedallas(...datosObtenidos);

  this.sumaGold = resultadoObtenido.sum('gold');
  this.sumaSilver = resultadoObtenido.sum('silver');
  this.sumaBronze = resultadoObtenido.sum('bronze');
},
alertaMedallas(oro, plata, bronce){
  descripcionMedallas(oro, plata, bronce);
}

```

Luego de obtener los datos de la API y parsearlos en un json dentro de la variable atletas son pasadas a una constante de nombre datosObtenidos y estos para saber la cantidad total de cada medalla se suman creando una clase de nombre sumaMedallas que se extiende de una función reservada de nombre Array para aplicarle la función reduce que es la encargada de contar todos los datos de una misma key y para ellos se crearon tres variables sumaGold, sumaSilver, sumaBronze, estas variables se les pasa la función sum que es para aplicar un filtro e indicar la key que se quiere sumar.

De acuerdo con lo solicitado se configuro para que cuando se presiona el icono de cada medalla se despliegue una ventana emergente como la mostrada a continuación:



Para ello cada que se presiona el botón correspondiente este ejecuta un método y le pasa la variable que le corresponde.

```
<div class="tab-pane fade" id="medallas" role="tabpanel" aria-labelledby="profile-tab">
  <br>
  <div class="table-responsive">
    <table class="table table-striped table-bordered table-hover align-middle display">
      <thead class="overflow-scroll">
        <tr>
          <th>#</th>
          <th>Medalla</th>
          <th>Cantidad</th>
        </tr>
      </thead>
      <tbody>
        <tr>
          <td>1</td>
          <td><a class="btn" v-on:click="alertaMedallas(oro)"></a> Oro</td>
          <td>{{ sumaGold }}</td>
        </tr>
        <tr>
          <td>2</td>
          <td><a class="btn" v-on:click="alertaMedallas(plata)"></a> Plata</td>
          <td>{{ sumaSilver }}</td>
        </tr>
        <tr>
          <td>3</td>
          <td><a class="btn" v-on:click="alertaMedallas(bronce)"></a> Bronce</td>
          <td>{{ sumaBronze }}</td>
        </tr>
      </tbody>
    </table>
  </div>
</div>
```

Este método es el siguiente:

```
alertaMedallas(oro, plata, bronce){
  descripcionMedallas(oro, plata, bronce);
}
```

Aquí el método espera cualquiera de las 3 variables para después ejecutar la función descipcionMedallas() que hace parte del archivo alerts.js el cuál está compuesto de la siguiente manera:

```
Dashboard.vue | alerts.js | X
src > alerts.js > ...
1  /** Se procede ocn la creación del apartado que contendrá las alertas y mensajes emergentes que se mostraran dependiendo de la función ejecutada */
2
3  import Swal from 'sweetalert2';
4
5  export function show_alert(icono, title) {
6    let timerInterval;
7    Swal.fire({
8      icon: icono,
9      title: title,
10     html: 'La notificación se cerrara in <b></b> milliseconds.',
11     timer: 2000,
12     timerProgressBar: true,
13     didOpen: () => {
14       Swal.showLoading()
15       const b = Swal.getHtmlContainer().querySelector('b')
16       timerInterval = setInterval(() => {
17         b.textContent = Swal.getTimerLeft()
18       }, 100)
19     },
20     willClose: () => {
21       clearInterval(timerInterval)
22     },
23     customClass: { confirmButton: 'btn btn-primary', popup: 'animated zoomIn' },
24   }).then((result) => {
25     /* Read more about handling dismissals below */
26     if (result.dismiss === Swal.DismissReason.timer) {
27       // console.log('I was closed by the timer')
28     }
29   })
30 }
31
32 export function descipcionMedallas(alertaMedalla) {
33   Swal.fire({
34     title: alertaMedalla,
35     showClass: {
36       popup: 'animate__animated animate__fadeInDown'
37     },
38     hideClass: {
39       popup: 'animate__animated animate__fadeOutUp'
40     }
41   })
42 }
```