



UNIVERSIDAD TÉCNICA ESTATAL DE QUEVEDO

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN Y

DISEÑO DIGITAL

SOFTWARE (REDISEÑO)

Aplicaciones web [5TO NIVEL] - A - SOFT-R

INFORME GRUPAL

Bedón Viteri Keyla Betzabé

Farinango Guandinango Freddy Vladimir

Villamarin Cuenca Iván Andrés

Zambrano Yong Angel Daniel

QUEVEDO LOS RIOS



Agosto 15

Introducción

En el presente trabajo se demuestra el uso de Vue.js y su integración con un backend en Spring Boot y PostgreSQL para el desarrollo de aplicaciones web modernas. Se abordan los conceptos esenciales de Vue.js, su arquitectura, ventajas, limitaciones y buenas prácticas, junto con una comparación entre los enfoques Server-Side Rendering (SSR) con Thymeleaf y Client-Side Rendering (CSR) con Vue.

Como caso práctico, se desarrolla el módulo Inventario de Equipos, implementando una API REST para operaciones CRUD y consultas específicas, y una interfaz en Vue que consume estos datos de forma reactiva con filtrado, ordenamiento y paginación. Finalmente, se integra el frontend dentro del proyecto Spring Boot, evidenciando cómo ambas tecnologías pueden coexistir para ofrecer soluciones interactivas, escalables y de fácil mantenimiento.

1. ¿Qué es Vue.js?

Vue.js es un framework progresivo de JavaScript que sirve para construir interfaces de usuario. Se le dice “progresivo” porque puedes usarlo para algo pequeño, como agregar una función dentro de una página ya hecha, o también para crear una aplicación completa que funcione en una sola página (SPA). Lo bueno es que no tienes que empezar con algo complicado, puedes ir poco a poco y agregar más funciones conforme lo necesitas [1].

Se centra en tres aspectos importantes. Primero, la reactividad, que significa que cuando los datos cambian, la vista se actualiza sola sin que tengas que hacerlo manualmente[2]. Segundo, el uso de componentes, que son piezas que se pueden reutilizar, como botones, formularios, tablas o tarjetas, para no tener que crearlos una y otra vez [1]. Y tercero, la simplicidad, porque tiene una forma de trabajar más amigable y fácil de aprender que otros frameworks, lo que ayuda a que más personas puedan comenzar sin tanta dificultad[3].

2. Historia y evolución breve

- Vue.js apareció en el año 2014 gracias a Evan You, quien buscaba crear una herramienta más ligera y sencilla que otras de ese momento, como AngularJS, pero sin perder lo útil que tenían. Desde el inicio, su objetivo fue que cualquier persona pudiera aprenderlo sin tanta dificultad, pero que a la vez sirviera para proyectos grandes [4] .
- Entre 2016 y 2020 llegó Vue 2, que marcó un gran paso porque estableció claramente el modelo de componentes, es decir, la forma de organizar las partes de una aplicación. También se fue formando un ecosistema más completo, con herramientas y extensiones que ayudaban a los desarrolladores a trabajar de forma más ordenada [5].
- En 2020 llegó Vue 3, que sigue siendo la versión actual. Fue prácticamente una reescritura interna para que todo funcionara más rápido y mejor. Además, trajo la Composition API, que da más libertad a la hora de programar, mejor compatibilidad con TypeScript para manejar datos con más seguridad y soporte oficial para Vite, una herramienta que acelera la creación y carga de proyectos [5].

3. Arquitectura y piezas clave

Vue.js se organiza en varias partes que trabajan juntas para que una aplicación funcione de manera ordenada[6], [7].

1. La primera es la capa de vista, que es lo que el usuario ve y con lo que interactúa. Aquí se usan estructuras como HTML y unas instrucciones llamadas directivas (por ejemplo, v-model, v-for o v-if) que sirven para conectar los datos con lo que aparece en pantalla de forma automática.
2. Luego está la capa de estado, donde se guardan y manejan los datos y la lógica de cada componente. Cuando la aplicación crece, estos datos pueden pasar a un lugar central llamado “store” (por ejemplo, Pinia) para que todo esté más organizado.
3. También están los servicios, que son las funciones que permiten comunicarse con un servidor o una API. Por ejemplo, se pueden usar herramientas como *fetch* o *axios* para enviar y recibir información desde tu backend (en tu caso, hecho con Spring Boot).
4. Otra parte importante es el enrutamiento, que se hace con Vue Router. Esto permite que, dentro de una sola aplicación, se puedan tener distintas páginas o rutas sin tener que recargar todo.

Finalmente, está la parte de build, que en Vue 3 normalmente se hace con Vite. Esto ayuda a desarrollar más rápido gracias a funciones como la recarga en caliente (hot-reload) y a preparar todo para que funcione bien en producción.

4. Conceptos fundamentales

Los conceptos básicos de Vue dan las herramientas necesarias para crear aplicaciones organizadas, fáciles de entender y que reaccionen de forma automática a los cambios en los datos [8], [9].

- Vue.js funciona con ideas clave que es importante conocer para poder usarlo bien. Una de ellas es la instancia de aplicación, que es como el punto de inicio de todo. Se crea con `createApp(App).mount('#app')`, lo que básicamente indica a Vue dónde y cómo debe “poner vida” a la página.
- Otra pieza importante son los componentes. Normalmente tienen archivos con extensión .vue y se dividen en tres partes: `<template>` (la estructura o diseño),

<script> (la lógica) y <style> (los estilos). Esto hace que todo esté junto y sea más fácil de entender.

- La **reactividad** es lo que permite que cuando los datos cambien, la interfaz se actualice sola. Esto se logra con herramientas como `ref`, `reactive`, `computed` y los “watchers” (observadores).
- También están las **props y eventos**, que sirven para que un componente padre y uno hijo puedan enviarse información entre sí. Por ejemplo, el padre puede pasarle datos al hijo con props, y el hijo puede avisar al padre que algo pasó usando eventos.
- Las **directivas** son instrucciones especiales que se colocan en el HTML para hacer tareas rápidas. Por ejemplo: `v-model` para enlazar datos con un input, `v-bind` para poner atributos, `v-on` para escuchar eventos como un clic, y `v-if` o `v-for` para mostrar o repetir elementos.
- Por último, la **Composition API** es una forma más moderna de organizar la lógica. Usa funciones como `setup`, `ref`, `computed` u `onMounted` para que el código sea más modular y fácil de mantener, en lugar de usar la forma más antigua llamada Options API.

5. Ecosistema esencial

El ecosistema de Vue reúne herramientas que facilitan el desarrollo, desde iniciar el proyecto y manejar páginas, hasta guardar datos y dar un mejor diseño, haciendo que todo el proceso sea más rápido y organizado[10], [11].

1. Vue.js cuenta con varias herramientas que lo acompañan y que ayudan a trabajar más rápido y de forma ordenada. Una de ellas es **Vite**, que permite que el proyecto se inicie casi al instante y que la compilación sea rápida y eficiente, lo que ahorra tiempo mientras se desarrolla.
2. Otra herramienta es **Vue Router**, que sirve para manejar las diferentes páginas o rutas dentro de una aplicación, pero sin necesidad de recargar toda la página cada vez que el usuario navega.
3. Para manejar la información que usan varios componentes está **Pinia**, que es un sistema de almacenamiento central donde se guardan datos y funciones que

pueden ser usados desde cualquier parte del proyecto. Es más sencillo de entender que el antiguo Vuex.

4. En cuanto a la comunicación con servidores, se usa mucho **Axios**, que permite conectarse con APIs y obtener o enviar datos de manera sencilla. Esto es muy útil para trabajar con información real que viene desde una base de datos o un backend.
5. También existen **kits de interfaz de usuario** como Element Plus, Vuetify, Naive UI o utilidades como Tailwind CSS. Estos ofrecen diseños ya listos o clases de estilo que ayudan a que la aplicación se vea bien sin tener que crear todo desde cero.

6. ¿Cuándo usar Vue en tu proyecto?

Vue es ideal cuando quieres agregar interactividad, trabajar separado del backend o avanzar rápido con un equipo que necesita una herramienta fácil de aprender y usar[12], [13].

- Vue.js es una buena opción cuando necesitas que tu página o aplicación sea más interactiva. Por ejemplo, si quieres que las tablas puedan buscar, filtrar o mostrar datos por páginas sin recargar todo, o si necesitas formularios que cambien y respondan en tiempo real mientras el usuario escribe o selecciona opciones.
- También es útil si quieres separar la parte visual del backend. Esto significa que puedes tener una aplicación completa que funcione en una sola página (SPA) o incluso mejorar poco a poco páginas hechas con Thymeleaf, agregando funciones interactivas sin tener que rehacer todo desde cero.
- Por último, Vue es una buena elección si buscas rapidez en el desarrollo y una curva de aprendizaje fácil, especialmente en equipos de estudiantes o personas que no tienen mucha experiencia previa. Esto hace que todos puedan trabajar sin sentir que la herramienta es demasiado complicada.

7. Ventajas y limitaciones

Vue.js tiene varias cosas positivas que lo hacen atractivo. Es sencillo para empezar, lo que permite que en poco tiempo ya estés creando algo funcional. También es muy productivo porque su forma de trabajar ahorra pasos y facilita las tareas. Su sistema de reactividad es muy potente, y los componentes reutilizables permiten ahorrar trabajo al no tener que repetir código. Además, cuenta con un ecosistema completo con herramientas como el router, el manejo de estado y Vite, que hacen más fácil el desarrollo. En su versión 3, Vue ofrece un rendimiento excelente, lo que significa que las aplicaciones son rápidas y responden bien[14].

Sin embargo, también tiene sus limitaciones. Si el equipo ya domina otra herramienta como React o Angular, el cambio puede ser difícil y llevar tiempo de adaptación. Y aunque Vue puede manejar proyectos grandes, es necesario ser muy ordenado en la arquitectura, organizando bien las carpetas, los stores, los servicios y, de ser posible, usando TypeScript para que el código sea más claro y seguro[15].

8. Buenas prácticas (aplicables a su proyecto)

Al trabajar con Vue, es fundamental seguir buenas prácticas para mantener el proyecto ordenado y fácil de gestionar, como modularizar la aplicación en componentes pequeños (TablaEquipos, FiltroEquipos, FormEquipo) para facilitar su modificación, centralizar las llamadas a la API en un único archivo de servicios (api/equiposService.js) para evitar repeticiones y simplificar cambios, gestionar datos como sesión o usuario mediante Pinia para acceso global, y aplicar validación en formularios estableciendo campos obligatorios y longitudes adecuadas, asegurando así la calidad y consistencia de la información enviada[16].

9) Pruebas, despliegue y mantenimiento

En un proyecto con Vue, es importante pensar en cómo probarlo, publicarlo y cuidarlo después. Para las pruebas de la interfaz se pueden usar herramientas como Vitest junto con Testing Library, aunque para un deber o trabajo universitario esto podría ser opcional. Sirve para asegurarse de que los botones, formularios y otras funciones respondan como deberían[17].

- Cuando el proyecto está listo, se genera una compilación con el comando `npm run build`. Esto crea una carpeta llamada `/dist`, que contiene todos los archivos optimizados y listos para ser publicados.

- El despliegue puede hacerse de dos formas. La primera es integrada, copiando el contenido de /dist a la carpeta /static de tu proyecto en Spring Boot, para que todo se sirva junto. La segunda es separada, usando un servidor como Nginx para mostrar la aplicación, mientras la API sigue en Spring Boot, recordando activar CORS para que se comuniquen sin problemas.
- En cuanto al mantenimiento, es buena práctica revisar la consola del navegador y la pestaña de red para detectar errores. También es importante manejar bien las fallas al llamar a la API, mostrando mensajes claros y amables para que el usuario sepa qué hacer.

Bibliografia

- [1] B. Nelson, “Getting to Know Vue.js: Learn to Build Single Page Applications in Vue from Scratch,” *Getting to Know Vue.js: Learn to Build Single Page Applications in Vue from Scratch*, pp. 1–265, Jan. 2018, doi: 10.1007/978-1-4842-3781-6.
- [2] SOLOMON. ESEME, “ARCHITECTING VUE. JS 3 ENTERPRISE-READY WEB APPLICATIONS; BUILD AND DELIVER SCALABLE AND HIGH-PERFORMANCE, ENTERPRISE-READY APPLICATIONS,” 2023.
- [3] JOHN. AU-YEUNG, “VUE.JS 3 BY EXAMPLE : build eight real-world applications from the ground up using vue 3,... vuex, and vuetify,” 2021.
- [4] J. G. Arévalo, L. Viecco, and L. Arévalo, “Methodology to define an integration process between frameworks SCRUM, Django REST framework y Vue.js, implemented for software development, from quality management approach and agility,” *IOP Conf Ser Mater Sci Eng*, vol. 844, no. 1, p. 012022, May 2020, doi: 10.1088/1757-899X/844/1/012022.
- [5] C. F. Manning, “Erik Hanchett Foreword by Chris Fritz”.
- [6] P. David. Garaguso and Olaf. Zander, “Vue.js 3 design patterns and best practices build enterprise-ready, modular Vue.js applications along with best practice with Pinia and Nuxt,” 2023.
- [7] B. Qin, “University experiment management system based on Django and Vue.js,” *Proceedings - 2021 2nd International Conference on Big Data and Informatization Education, ICBDE 2021*, pp. 453–456, Apr. 2021, doi: 10.1109/ICBDE52740.2021.00109.
- [8] Joran. Quinten, “BUILDING REAL-WORLD WEB APPLICATIONS WITH VUE.JS 3 : build a portfolio of Vue.js and TypeScript web applications to advance your career in web development,” 2024.
- [9] S. Riyadi, N. Sakinah, and Asroni, “Implementation of Vue.Js Framework for Front-End Development: A Study Case of Employee Information System,” *Proceedings - ICE3IS 2024: 4th International Conference on Electronic and Electrical Engineering and Intelligent System: Leading-Edge Technologies for Sustainable Societies*, pp. 105–109, 2024, doi: 10.1109/ICE3IS62977.2024.10775391.
- [10] I. Revenchuk and V. Steshko, “Архітектурні рішення і методи оптимізації для підвищення продуктивності до- датків на Node.js та Vue.js,” *Біоніка інтелекту*, vol. 1, no. 98, pp. 64–69, Dec. 2022, doi: 10.30837/BI.2022.1(98).08.

- [11] J. Song, M. Zhang, and H. Xie, "Design and Implementation of a Vue.js-Based College Teaching System," vol. 14, no. 13, p. 59, 2019, doi: 10.3991/ijet.v14i13.10709.
- [12] O. C. Novac, D. E. Madar, C. M. Novac, G. Bujdoso, M. Oproescu, and T. Gal, "Comparative study of some applications made in the Angular and Vue.js frameworks," *2021 16th International Conference on Engineering of Modern Electric Systems, EMES 2021 - Proceedings*, Jun. 2021, doi: 10.1109/EMES52337.2021.9484150.
- [13] Nanik. Tolaram and Nick. Glynn, "Full-stack web development with Go : build your web applications quickly using the Go programming language and Vue.js," 2023.
- [14] K. Bielak, B. Borek, and M. Plechawska-Wójcik, "Web application performance analysis using Angular, React and Vue.js frameworks," *Journal of Computer Sciences Institute*, vol. 23, pp. 77–83, Jun. 2022, doi: 10.35784/JCSI.2827.
- [15] J. Sianandar and I. B. Kerthyayana Manuaba, "Performance Analysis of Hooks Functionality in React and Vue Frameworks," *Proceedings of 2022 International Conference on Information Management and Technology, ICIMTech 2022*, pp. 139–143, 2022, doi: 10.1109/ICIMTECH55957.2022.9915183.
- [16] T. Maulida *et al.*, "Visualization of Front-End Data Logger Internet of Things Technology using Vue.Js Framework," *Proceeding - 6th International Conference on Information Technology, Information Systems and Electrical Engineering: Applying Data Sciences and Artificial Intelligence Technologies for Environmental Sustainability, ICITISEE 2022*, pp. 693–698, 2022, doi: 10.1109/ICITISEE57756.2022.10057919.
- [17] P. D. Dimitar G Velev, "PROCEEDINGS OF THE 7 TH INTERNATIONAL CONFERENCE ON APPLICATION OF INFORMATION AND COMMUNICATION TECHNOLOGY AND STATISTICS IN ECONOMY AND EDUCATION ICAICTSEE-2017 EDITOR: PUBLISHING COMPLEX-UNWE, SOFIA, BULGARIA".

Manual del framework — guía paso a paso para partes del proyecto

Vista “Equipos” con Thymeleaf (SSR) vs Vue.js (CSR)

Mediante esta práctica se podrá notar las diferencias prácticas que existe entre un enfoque del lado del servidor (SSR) usando Thymeleaf y un enfoque del lado del cliente (CSR) usando Vue.js, aplicadas a la pantalla “Equipos”. Se incluye la implementación y fragmentos de código existente.

1. Enfoques de renderizado

-Thymeleaf (SSR)

El HTML se construye en el servidor y se envía completo al navegador.

Las interacciones (filtrar, paginar, etc.) suelen implicar enviar el formulario y recargar la página con un nuevo HTML.

Ventaja: simplicidad en flujos tradicionales basados en formularios.

-Vue.js (CSR)

El HTML se construye y actualiza en el navegador; los datos se solicitan a una API en formato JSON.

Las interacciones son reactivas: cambios de filtros/orden/paginación no recargan toda la página; se actualiza solo el DOM necesario.

Ventaja: experiencia de usuario más fluida y responsiva.

2. Arquitectura actual del módulo “Equipos”

El controlador expone la vista SSR y una vista alternativa para Vue:

```
@Controller
@RequestMapping("/dashboard/admin/equipos")
public class EquipoController {
```

```

@Autowired private EquipoService equipoService;
@Autowired private LaboratorioRepository laboratorioRepository;

@GetMapping
public String listar(@RequestParam(required = false) String q,
                    @RequestParam(required = false) Integer labId,
                    @RequestParam(required = false) String estado,
                    Model model) {

    model.addAttribute("equipos", equipoService.listar(q, labId,
estado));
    model.addAttribute("Labs", equipoService.laboratorios());
    model.addAttribute("q", q);
    model.addAttribute("LabId", labId);
    model.addAttribute("estado", estado);

    model.addAttribute("equipoForm", new Equipo());
    return "dashboard/equipos";
}

@PostMapping("/crear")
public String crear(@ModelAttribute("equipoForm") @Valid Equipo
equipo,
                    BindingResult br,
                    RedirectAttributes ra) {
    if (br.hasErrors()) {
        ra.addFlashAttribute("error", "Datos inválidos. Revisa Los
campos.");
        return "redirect:/dashboard/admin/equipos";
    }

    // Resolver Laboratorio por ID
    if (equipo.getLaboratorio() != null &&
equipo.getLaboratorio().getIdLaboratorio() != null) {
        Laboratorio lab = laboratorioRepository
            .findById(equipo.getLaboratorio().getIdLaboratorio())
            .orElse(null);
        equipo.setLaboratorio(lab);
    }

    try {
        equipoService.guardar(equipo);
        ra.addFlashAttribute("ok", "Equipo creado correctamente.");
    } catch (IllegalArgumentException e) {
        ra.addFlashAttribute("error", e.getMessage());
    }
    return "redirect:/dashboard/admin/equipos";
}

```

```

@PostMapping("/editar/{id}")
public String editar(@PathVariable Integer id,
                    @ModelAttribute("equipoForm") @Valid Equipo
equipo,
                    BindingResult br,
                    RedirectAttributes ra) {
    if (br.hasErrors()) {
        ra.addFlashAttribute("error", "Datos inválidos. Revisa los
campos.");
        return "redirect:/dashboard/admin/equipos";
    }
    Equipo existente = equipoService.buscarPorId(id);
    if (existente == null) {
        ra.addFlashAttribute("error", "El equipo no existe.");
        return "redirect:/dashboard/admin/equipos";
    }

    // Resolver laboratorio por ID
    Laboratorio lab = null;
    if (equipo.getLaboratorio() != null &&
equipo.getLaboratorio().getIdLaboratorio() != null) {
        lab =
laboratorioRepository.findById(equipo.getLaboratorio().getIdLaboratorio())
).orElse(null);
    }

    existente.setCodigoEquipo(equipo.getCodigoEquipo());
    existente.setTipoEquipo(equipo.getTipoEquipo());
    existente.setMarca(equipo.getMarca());
    existente.setModelo(equipo.getModelo());
    existente.setEstado(equipo.getEstado());
    existente.setLaboratorio(lab);

    try {
        equipoService.guardar(existente);
        ra.addFlashAttribute("ok", "Equipo actualizado.");
    } catch (IllegalArgumentException e) {
        ra.addFlashAttribute("error", e.getMessage());
    }
    return "redirect:/dashboard/admin/equipos";
}

@PostMapping("/eliminar/{id}")
public String eliminar(@PathVariable Integer id, RedirectAttributes ra) {
    try {
        equipoService.eliminar(id);
        ra.addFlashAttribute("ok", "Equipo eliminado.");
    }
}

```

```

    } catch (IllegalStateException e) {
        ra.addFlashAttribute("error", e.getMessage());
    } catch (Exception e) {
        ra.addFlashAttribute("error", "No se pudo eliminar el equipo.");
    }
    return "redirect:/dashboard/admin/equipos";
}

@GetMapping("/editar/{id}")
public String editarVista(@PathVariable Integer id,
    @RequestParam(required = false) String q,
    @RequestParam(required = false) Integer labId,
    @RequestParam(required = false) String estado,
    Model model,
    RedirectAttributes ra) {
    var equipo = equipoService.buscarPorId(id);
    if (equipo == null) {
        ra.addFlashAttribute("error", "El equipo no existe.");
        return "redirect:/dashboard/admin/equipos";
    }

    // Listas y filtros (para que la vista tenga todo)
    model.addAttribute("equipos", equipoService.listar(q, labId,
estado));
    model.addAttribute("labs", equipoService.laboratorios());
    model.addAttribute("q", q);
    model.addAttribute("labId", labId);
    model.addAttribute("estado", estado);

    // Modo edición
    model.addAttribute("equipoForm", equipo);
    model.addAttribute("editMode", true);
    model.addAttribute("editId", id);

    return "dashboard/equipos";
}

@GetMapping("/vue")
public String vistaVue() {
    return "dashboard/equipos-vue";
}
}

```

Consideración: se dispone de dos rutas para comparar en vivo.

SSR: /dashboard/admin/equipos

CSR: /dashboard/admin/equipos/vue

3. Flujo con Thymeleaf (SSR)

-Filtros y envío (GET)

Los filtros se envían al servidor; esta arma un nuevo HTML con el resultado:

```
<form class="anuncios" th:action="@{/dashboard/admin/equipos}"
method="get" style="gap:8px;align-items:end;flex-wrap:wrap">
  <div>
    <label>Buscar</label>
    <input name="q" type="text" th:value="{q}" placeholder="código
/ tipo / marca / modelo" style="padding:8px;border-radius:8px;border:1px
solid #ddd;">
  </div>
  <div>
    <label>Laboratorio</label>
    <select name="labId" style="padding:8px;border-
radius:8px;border:1px solid #ddd;min-width:200px">
      <option value="" th:selected="{labId ==
null}">Todos</option>
      <option th:each="l : {labs}"
        th:value="{l.idLaboratorio}"
        th:text="{l.nombreLaboratorio}"
        th:selected="{labId != null and labId ==
l.idLaboratorio}">
      </option>
    </select>
  </div>
  <div>
    <label>Estado</label>
    <select name="estado" style="padding:8px;border-
radius:8px;border:1px solid #ddd;">
      <option value="" th:selected="{estado == null or estado ==
''}">Todos</option>
      <option th:value="'activo'"          th:selected="{estado ==
'activo'}">Activo</option>
      <option th:value="'inactivo'"        th:selected="{estado ==
'inactivo'}">Inactivo</option>
      <option th:value="'mantenimiento'" th:selected="{estado ==
'mantenimiento'}">Mantenimiento</option>
    </select>
  </div>
```

```

<button class="filtro-btn" type="submit">Filtrar</button>
<a class="btn" th:href="@{/dashboard/admin/equipos}">Limpiar</a>
</form>

```

-Render de la tabla en servidor

El servidor ya pasa la lista equipos y Thymeleaf itera:

```

<tbody>
    <tr th:each="e: ${equipos}">
        <td style="padding:10px;border-bottom:1px solid #f0f0f0"
th:text="${e.codigoEquipo}"></td>
        <td style="padding:10px;border-bottom:1px solid #f0f0f0"
th:text="${e.tipoEquipo}"></td>
        <td style="padding:10px;border-bottom:1px solid #f0f0f0"
th:text="${e.marca}"></td>
        <td style="padding:10px;border-bottom:1px solid #f0f0f0"
th:text="${e.modelo}"></td>
        <td style="padding:10px;border-bottom:1px solid #f0f0f0"
th:text="${e.estado}"></td>
        <td style="padding:10px;border-bottom:1px solid #f0f0f0"
th:text="${e.laboratorio != null ? e.laboratorio.nombreLaboratorio :
''}"></td>
        <td style="padding:10px;border-bottom:1px solid #f0f0f0">
            <!-- Editar: ahora GET para precargar el form -->
            <a class="btn"
th:href="@{'/dashboard/admin/equipos/editar/' +
${e.idEquipo}}">Editar</a>

            <form th:action="@{'/dashboard/admin/equipos/eliminar/' +
${e.idEquipo}}" method="post" style="display:inline-block"
onsubmit="return confirm('¿Eliminar equipo?');">
                <button class="logout" type="submit" style="padding:8px
12px;border-radius:8px">Eliminar</button>
            </form>
        </td>
    </tr>
    <tr th:if="${#lists.isEmpty(equipos)}">
        <td colspan="7" style="padding:14px;color:#777;">No hay
equipos para mostrar.</td>
    </tr>
</tbody>

```

4. Flujo con Vue (CSR) en equipos-vue.html

-Reactividad y carga de datos

La vista CSR gestiona estado, filtros y peticiones desde el navegador:

```
<div id="app">
  <!-- Filtros -->
  <div class="row g-2 mb-3">
    <div class="col-sm-6 col-md-4">
      <input class="form-control" v-model="q" placeholder="Buscar
(código, tipo, marca, modelo)" />
    </div>
    <div class="col-sm-4 col-md-3">
      <input class="form-control" v-model="estado" placeholder="Estado
(opcional)" />
    </div>
    <div class="col-sm-4 col-md-3">
      <input class="form-control" v-model.number="labId"
placeholder="ID de laboratorio (opcional)" />
    </div>
    <div class="col-sm-4 col-md-2 d-grid">
      <button class="btn btn-primary" @click="cargar">Buscar</button>
    </div>
  </div>
</script>
<script>
  const { createApp, ref, computed, onMounted, watch } = Vue;

  createApp({
    setup() {
      const q = ref("");
      const labId = ref(null);
      const estado = ref("");
      const orden = ref("codigoEquipo");

      const equipos = ref([]);
      const cargando = ref(false);
      const error = ref("");

      // Paginación
      const page = ref(1);
      const pageSize = ref(10);

      const cargar = async () => {
        cargando.value = true;
        error.value = "";
        try {
          // Llama a tu API REST (debes tener EquipoApiController)
          const params = new URLSearchParams();
          if (q.value) params.append("q", q.value);
          if (labId.value) params.append("labId", labId.value);
        }
      }
    }
  }).mount("#app")
</script>
```

```

        if (estado.value) params.append("estado", estado.value);

        const r = await
fetch(`/api/admin/equipos?${params.toString()}`);
        if (!r.ok) throw new Error("No se pudo cargar la lista");
        equipos.value = await r.json();
        page.value = 1; // reset página al recargar
    } catch (e) {
        error.value = e.message;
    } finally {
        cargando.value = false;
    }
};

onMounted(cargar);

```

-Orden y paginación en cliente

El orden y la paginación se calculan sin pedir nada al servidor:

```

<table class="table table-striped align-middle">
  <thead>
    <tr>
      <th :class="{ 'active-col': orden=== 'codigoEquipo' }"
@click="orden='codigoEquipo'">Código</th>
      <th :class="{ 'active-col': orden=== 'tipoEquipo' }"
@click="orden='tipoEquipo'">Tipo</th>
      <th :class="{ 'active-col': orden=== 'marca' }"
@click="orden='marca'">Marca</th>
      <th :class="{ 'active-col': orden=== 'modelo' }"
@click="orden='modelo'">Modelo</th>
      <th :class="{ 'active-col': orden=== 'estado' }"
@click="orden='estado'">Estado</th>
    </tr>
  </thead>

```

```

// Orden + conteo + paginación
const ordenados = computed(() => {
  return equipos.value.slice().sort((a, b) => {
    const A = ("" + (a[orden.value] ?? "")).toLowerCase();
    const B = ("" + (b[orden.value] ?? "")).toLowerCase();
    return A.localeCompare(B);
  });
});

```

```

const totalFiltrados = computed(() => ordenados.value.length);

const totalPages = computed(() => {
  return Math.max(1, Math.ceil(ordenados.value.length /
    pageSize.value));
});

watch([pageSize, ordenados], () => { page.value = 1; });

const paginaActual = computed(() => {
  const start = (page.value - 1) * pageSize.value;
  return ordenados.value.slice(start, start + pageSize.value);
});

```

Conclusión CSR: filtrar → fetch a la API; ordenar/paginar → sin recarga, cálculo local y actualización inmediata del DOM.

5. “Contrato” de la API

API es como 4 puertas para manejar equipos desde Vue:

- Ver (listar con filtros)
- Crear
- Actualizar
- Eliminar

```

const params = new URLSearchParams();
if (q.value) params.append("q", q.value);
if (labId.value) params.append("labId", labId.value);
if (estado.value) params.append("estado", estado.value);

const r = await
fetch(`/api/admin/equipos?${params.toString()}`);

```

-Ver (GET)

devuelve la lista de equipos en JSON, aplicando filtros si los envías.

Ejemplo de URL:

/api/admin/equipos?q=hp&estado=activo&labId=3

Ejemplo de respuesta (JSON):

```
[
  {
    "idEquipo": 12,
    "codigoEquipo": "HP-001",
    "tipoEquipo": "PC",
    "marca": "HP",
    "modelo": "ProDesk",
    "estado": "activo",
    "laboratorio": { "idLaboratorio": 3, "nombreLaboratorio": "Lab Redes" }
  }
]
```

Vue recibe ese JSON, lo guarda en `equipos.value` y pinta la tabla sin recargar.

-Crear (POST)

agrega un equipo nuevo.

```
await fetch('/api/admin/equipos', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    codigoEquipo: "PC-123",
    tipoEquipo: "PC",
    marca: "Dell",
    modelo: "OptiPlex",
    estado: "activo",
    laboratorio: { idLaboratorio: 3 } // ➡ así
  })
});
```

-Actualizar (PUT)

cambia datos de un equipo existente.

```
await fetch('/api/admin/equipos/12', {
  method: 'PUT',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    codigoEquipo: "HP-001",
    tipoEquipo: "PC",
    marca: "HP",
    modelo: "ProDesk G6",
    estado: "mantenimiento",
    laboratorio: { idLaboratorio: 4 }
  })
});
```

-Eliminar (DELETE)

borra un equipo por su id. En Vue, tras borrar, simplemente recargas la lista con tu cargar().

```
const eliminar = async (id) => {  
  if (!confirm('¿Eliminar equipo?')) return;  
  const r = await fetch(`/api/admin/equipos/${id}`, { method: 'DELETE' });  
};  
if (!r.ok) { error.value = 'No se pudo eliminar'; return; }  
await cargar();
```

6. Manual del framework — Guía paso a paso (Vue.js)

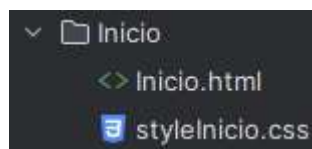
Propósito del módulo

Implementar un listado filtrable de ayudantías/asignaturas en una página existente, usando Vue 3:

- Filtro por nombre o docente (búsqueda reactiva).
- Filtro por día.
- Botón de recarga que vuelve a consultar al backend.
- Estados de Cargando... y Sin resultados.

Prerrequisitos

- Página base con header + sidebar y hoja de estilos styleInicio.css.
- Backend con endpoint GET /api/ayudantias (ver contrato en §6).
- Conexión a Internet (para cargar Vue por CDN).



Insertar el contenedor de Vue

En index.html, dentro de <div class="dashboard">, después del <aside class="sidebar">:

```
<li><a href="#"><i class="fas fa-calendar-alt"></i> Calendario</a></li>
<li><a href="#"><i class="fas fa-comments"></i> Foros</a></li>
<li><a href="#"><i class="fas fa-file-alt"></i> Documentos</a></li>
<li><a href="#"><i class="fas fa-cog"></i> Configuración</a></li>
<li><a href="#"><i class="fas fa-sign-out-alt"></i> Cerrar Sesión</a></li>
</ul>
</aside>
<div id="app">
  <ayudantias-list></ayudantias-list>
</div>
</div>
```

Cargar Vue (CDN) y el script del componente

Al final de index.html, antes de </body>:

```
<script src="https://unpkg.com/vue@3/dist/vue.global.prod.js"></script>
<script>
  const { createApp, ref, computed, onMounted } = Vue;

  const AyudantiasList = {
    template: /*html*/
    <section class="panel">
      <h2 style="margin-bottom:.5rem;">
        <i class="fas fa-chalkboard-teacher"></i> Ayudantias / Asignaturas
      </h2>

      <div style="display:flex; gap:.5rem; flex-wrap:wrap; margin-bottom:.75rem;">
        <input v-model="q" placeholder="Filtrar por nombre o docente..."
          style="padding:.5rem; border:1px solid #e3e6ea; border-radius:6px; flex:1; min-width:220px;">
        <select v-model="dia"
          style="padding:.5rem; border:1px solid #e3e6ea; border-radius:6px;">
          <option value="">Todos los días</option>
          <option>Lunes</option><option>Martes</option><option>Miércoles</option>
          <option>Jueves</option><option>Viernes</option><option>Sábado</option>
        </select>
        <button @click="cargar" :disabled="loading">

```

```

        style="padding:.5rem .8rem; border:none; border-radius:6px; background:var(--secondary); color:#fff;">
        {{ loading ? 'Cargando...' : 'Recargar' }}
    </button>
</div>

<p v-if="loading">Cargando datos...</p>
<p v-else-if="filtradas.length === 0">Sin resultados.</p>

<ul v-else style="list-style:none; padding:0; display:grid; gap:.5rem;">
    <li v-for="a in filtradas" :key="a.id"
        style="background:#fff; border:1px solid #f0f0f0; border-radius:8px; padding:.75rem;">
        <strong>{{ a.nombre }}</strong><br>
        <small>
            <i class="fas fa-user-tie"></i> {{ a.docente }} ·
            <i class="fas fa-calendar-day"></i> {{ a.dia }} ·
            <i class="fas fa-clock"></i> {{ a.horario }}
        </small>
    </li>
</ul>
</section>

```

```

</ul>
</section>
',
    setup() {
        const lista = ref([]);
        const q = ref('');
        const dia = ref('');
        const loading = ref(false);

        const normal = s => (s||'').toLowerCase().normalize('NFD').replace(/[\p{Diacritic}]/gu, '');

        const filtradas = computed(() =>
            lista.value.filter(a =>
                (!q.value || normal(a.nombre).includes(normal(q.value))) || normal(a.docente).includes(normal(q.value))
                (!dia.value || a.dia === dia.value)
            )
        );
    }
);

```

```

async function cargar() {
    loading.value = true;
    try {
        const r = await fetch('/api/ayudantias');
        if (!r.ok) throw new Error('HTTP ' + r.status);
        lista.value = await r.json();
    } catch (e) {
        lista.value = [
            { id:1, nombre:'Estructuras de Datos', docente:'Ing. Ramírez', dia:'Lunes', horario:'08:00' },
            { id:2, nombre:'Cálculo II', docente:'MSc. Pacheco', dia:'Martes', horario:'18:00' },
            { id:3, nombre:'Base de Datos', docente:'Ing. Torres', dia:'Miércoles', horario:'14:00' },
            { id:4, nombre:'Programación Web', docente:'Ing. López', dia:'Jueves', horario:'09:00' },
            { id:5, nombre:'IA Aplicada', docente:'Dra. Fernández', dia:'Viernes', horario:'16:00' },
        ];
    } finally {
        loading.value = false;
    }
}

```

```

        onMounted(cargar);
        return { q, dia, loading, filtradas, cargar };
    };
    createApp({ components:{ AyudantiasList } }).mount('#app');
</script>

```

7. Conclusión

El enfoque SSR con Thymeleaf resulta adecuado para flujos simples basados en formularios, mientras que el enfoque CSR con Vue aporta una experiencia más ágil en pantallas con alta interacción (filtros, orden, paginación, acciones). La coexistencia de ambas vistas permite demostrar comparativamente las diferencias en tiempo percibido y usabilidad sin modificar la lógica de negocio existente.

Inventario de Equipos (Vue + Spring Boot + PostgreSQL)

1) Backend: preparar la API REST

1.1 Entidad JPA Equipo (mapea a la tabla equipo)

Src/main/java/com/uteq/SCLI/model/Equipo.java

La clase Equipo es una entidad JPA que mapea la tabla equipo, almacenando datos como código, tipo, marca, modelo, estado y el laboratorio al que pertenece, con un identificador generado automáticamente.

```
package com.uteq.SCLI.model;

import jakarta.persistence.*;

@Entity
@Table(name = "equipo")
public class Equipo {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id_equipo")
    private Integer id;

    @Column(name = "codigo_equipo", nullable = false, unique = true)
    private String codigoEquipo;

    @Column(name = "tipo_equipo", nullable = false)
    private String tipoEquipo;
```



```

@Column(name = "marca", nullable = false)
private String marca;

@Column(name = "modelo")
private String modelo;

@Column(name = "estado")
private String estado;

@Column(name = "id_laboratorio", nullable = false)
private Integer idLaboratorio;

// Getters y Setters
public Integer getId() { return id; }
public void setId(Integer id) { this.id = id; }

public String getCodigoEquipo() { return codigoEquipo; }
public void setCodigoEquipo(String codigoEquipo) {
this.codigoEquipo = codigoEquipo; }

public String getTipoEquipo() { return tipoEquipo; }
public void setTipoEquipo(String tipoEquipo) {
this.tipoEquipo = tipoEquipo; }

public String getMarca() { return marca; }
public void setMarca(String marca) { this.marca = marca; }

public String getModelo() { return modelo; }
public void setModelo(String modelo) { this.modelo = modelo;
}

public String getEstado() { return estado; }
public void setEstado(String estado) { this.estado = estado;
}

public Integer getIdLaboratorio() { return idLaboratorio; }
public void setIdLaboratorio(Integer idLaboratorio) {
this.idLaboratorio = idLaboratorio; }
}

```

1.2 Repositorio

Src/main/java/com/uteq/SCLI/repository/EquipoRepository.java

La interfaz EquipoRepository extiende JpaRepository para manejar operaciones CRUD sobre la entidad Equipo y añade consultas personalizadas, tanto en JPQL como en SQL nativo, para obtener equipos filtrados por laboratorio o mediante una función almacenada en la base de datos.

```
// src/main/java/com/uteq/SCLI/repository/EquipoRepository.java
package com.uteq.SCLI.repository;

import com.uteq.SCLI.model.Equipo;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository; // <-- ESTE
import es el que faltaba

import java.util.List;

@Repository
public interface EquipoRepository extends JpaRepository<Equipo,
Integer> {

    // JPQL: por id de laboratorio (tabla mapeada a la entidad
    Equipo)
    @Query("SELECT e FROM Equipo e WHERE e.idLaboratorio =
:labId")
    List<Equipo> findByIdLaboratorio(@Param("labId") int labId);

    // FUNCTION/SPA nativa (ajusta el nombre si tu función se
    llama distinto)
    @Query(value = "SELECT * FROM
sp_equipos_por_laboratorio(:lab)", nativeQuery = true)
    List<Equipo> equiposPorLaboratorio(@Param("lab") Integer
lab);
}
```

1.3 Servicio (con alias para compatibilidad)

Src/main/java/com/uteq/SCLI/service/EquipoService.java

La clase EquipoService es un servicio de Spring que centraliza la lógica de negocio para la entidad Equipo. Proporciona métodos CRUD básicos y consultas personalizadas para obtener equipos filtrados por laboratorio, ya sea mediante JPQL o una función almacenada, delegando las operaciones al EquipoRepository.

```
import com.uteq.SCLI.model.Equipo;
import com.uteq.SCLI.repository.EquipoRepository;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class EquipoService {

    private final EquipoRepository repo;

    // Inyección por constructor
    public EquipoService(EquipoRepository repo) {
        this.repo = repo;
    }

    // ----- CRUD -----
    public List<Equipo> listar() { return repo.findAll(); }

    public Equipo crear(Equipo equipo) { return
repo.save(equipo); }

    public Equipo actualizar(Integer id, Equipo equipo) {
        equipo.setId(id);
        return repo.save(equipo);
    }

    public void eliminar(Integer id) { repo.deleteById(id); }

    // ----- Consultas por laboratorio -----

    // JPQL
    public List<Equipo> obtenerPorLaboratorio(int labId) {
        return repo.findByIdLaboratorio(labId);
    }
}
```

```

    }

    // FUNCTION/SP nativa
    public List<Equipo> obtenerSPPorLaboratorio(Integer lab) {
        return repo.equiposPorLaboratorio(lab);
    }

    // ♦ Alias para que tu controller actual no rompa
    public List<Equipo> porLaboratorio(Integer lab) {
        return repo.equiposPorLaboratorio(lab);
    }
}

```

1.4 Controlador REST

Src/main/java/com/uteq/SCLI/controller/EquipoController.java

Controlador REST que expone la API /api/equipos con operaciones CRUD sobre Equipo (GET, POST, PUT, DELETE) y un endpoint extra GET /api/equipos/laboratorio/{lab} para listar equipos por id de laboratorio, delegando la lógica al EquipoService.

```

package com.uteq.SCLI.controller;

import com.uteq.SCLI.model.Equipo;
import com.uteq.SCLI.service.EquipoService;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/equipos")
public class EquipoController {

    private final EquipoService service;

    public EquipoController(EquipoService service) {
        this.service = service;
    }

    @GetMapping
    public List<Equipo> listar() {

```

```

        return service.listar();
    }

    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public Equipo crear(@RequestBody Equipo equipo) {
        return service.crear(equipo);
    }

    @PutMapping("/{id}")
    public Equipo actualizar(@PathVariable Integer id,
    @RequestBody Equipo equipo) {
        return service.actualizar(id, equipo);
    }

    @DeleteMapping("/{id}")
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void eliminar(@PathVariable Integer id) {
        service.eliminar(id);
    }

    @GetMapping("/laboratorio/{lab}")
    public List<Equipo> porLaboratorio(@PathVariable Integer
lab){
        return service.porLaboratorio(lab);
    }

}

```

3) Frontend aparte: Vue + Vite

3.1 Crear proyecto

```

npm create vite@latest frontend -- --template vue
cd frontend
npm i
npm i axios

```

3.2 Proxy y alias en Vite

frontend/vite.config.js

```

1 import { defineConfig } from 'vite'
2 import vue from '@vitejs/plugin-vue'
3 import { fileURLToPath, URL } from 'node:url'
4
5 export default defineConfig({
6   plugins: [vue()],
7   resolve: {
8     alias: { '@': fileURLToPath(new URL('./src', import.meta.url)) }
9   },
10  server: {
11    port: 5173,
12    proxy: { '/api': { target: 'http://localhost:8081', changeOrigin: true } }
13  }
14 })
15

```

3.3 Cliente HTTP + servicio

frontend/src/services/http.js

```

1
2 // frontend/src/services/http.js
3 import axios from 'axios'
4
5 // Base común para todas las llamadas al backend
6 const http :AxiosInstance = axios.create({
7   baseURL: '/api', // gracias al proxy de Vite -> http://localhost:8
8   timeout: 10000 // opcional: 10s
9 })
10
11 export default http Show usages · new *
12

```

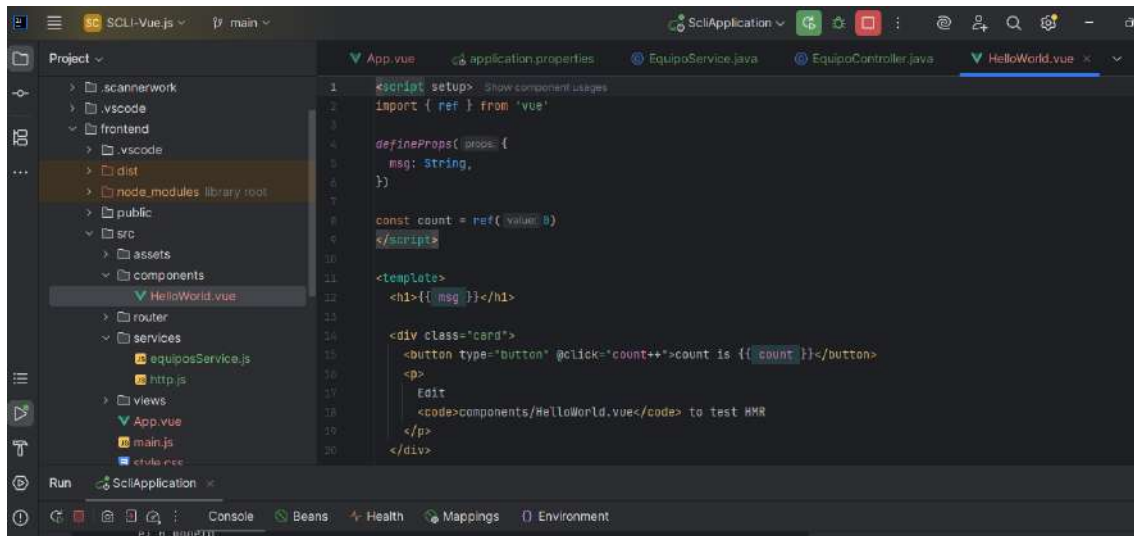
```

1 // frontend/src/services/equiposService.js
2 import http from './http'
3
4 // Solo GET por ahora (listar)
5 export const listarEquipos = () => http.get('/equipos') Show usages · new *
6
7 // Dejamos preparados los demás para después:
8 export const crearEquipo = (data) => http.post('/equipos', data) Show usages · new *
9 export const actualizarEquipo = (id, data) => http.put('/equipos/${id}', data) Show usages · new *
10 export const eliminarEquipo = (id) => http.delete('/equipos/${id}') Show usages · new *
11
12 // Consulta adicional por laboratorio (cuando la expongas en el backend)
13 export const equiposPorLab = (lab) => http.get('/equipos/laboratorio/${lab}') Show usages · new *
14
15 export const getEquiposPorLab = async (labId) => {
16   const res :Response = await fetch(`${api}/equipos/laboratorio/${labId}`);
17   return res.json();
18 };
19

```

3.4 Vista (tabla + formulario + filtro)

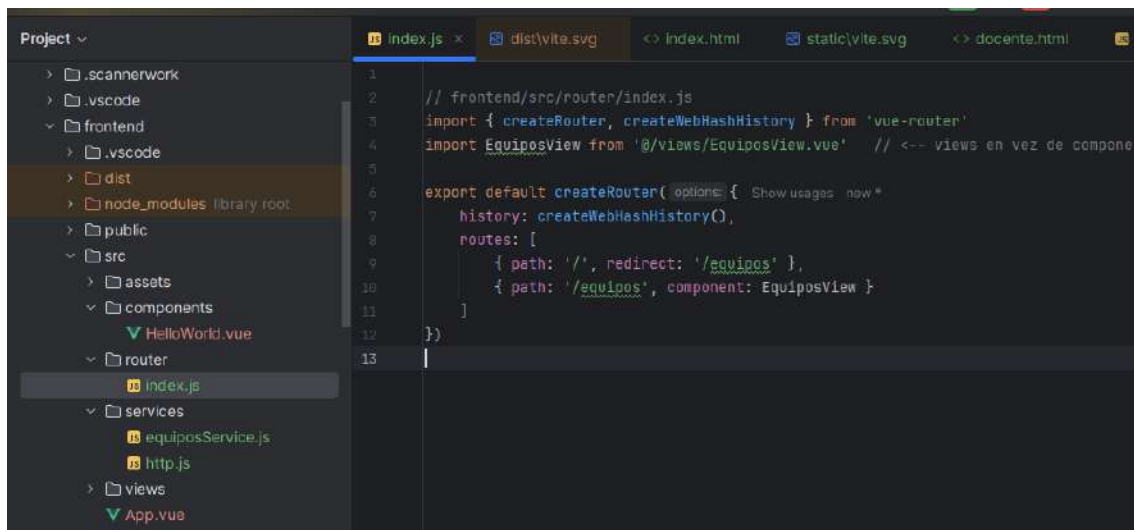
frontend/src/components/EquiposView.vue



```
1 <script setup>
2 import { ref } from 'vue'
3
4 defineProps({
5   msg: String,
6 })
7
8 const count = ref(0)
9
10 </script>
11
12 <template>
13   <h1>{{ msg }}</h1>
14
15   <div class="card">
16     <button type="button" @click="count++>count is {{ count }}</button>
17   </div>
18
19   <p>
20     Edit
21     <code>components/HelloWorld.vue</code> to test HMR
22   </p>
23 </template>
```

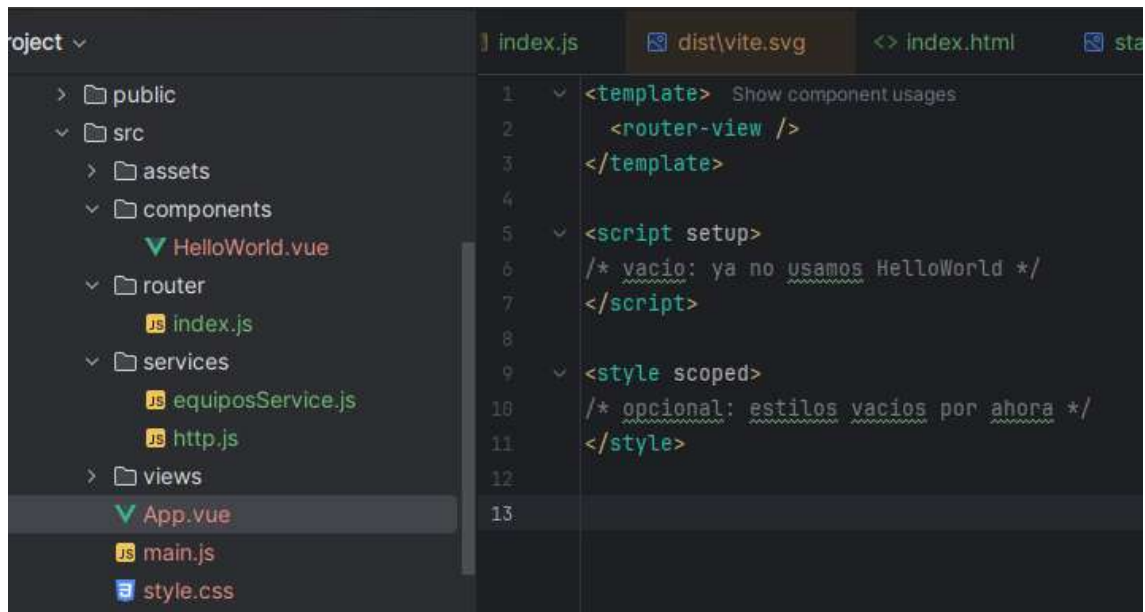
3.5 Router + App

frontend/src/router/index.js (modo **history** en dev; en prod usaremos **hash**)

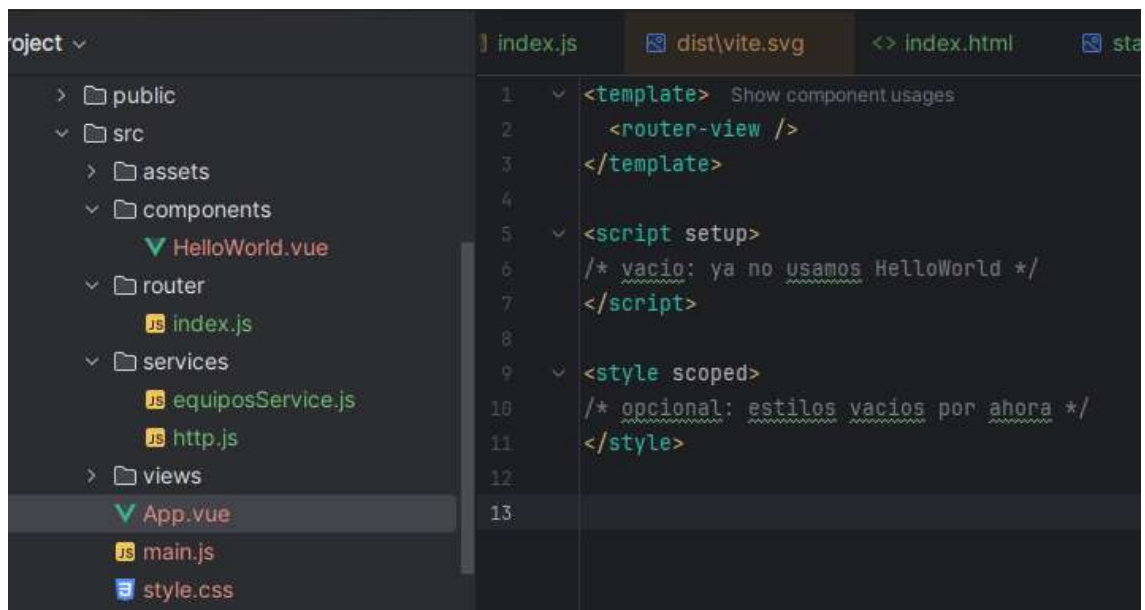


```
1 // frontend/src/router/index.js
2 import { createRouter, createWebHashHistory } from 'vue-router'
3 import EquiposView from '@/views/EquiposView.vue' // <-- views en vez de componen
4
5 export default createRouter({
6   history: createWebHashHistory(),
7   routes: [
8     { path: '/', redirect: '/equipos' },
9     { path: '/equipos', component: EquiposView }
10   ]
11 })
```

Frontend/src/App.vue.js



frontend/src/main.js



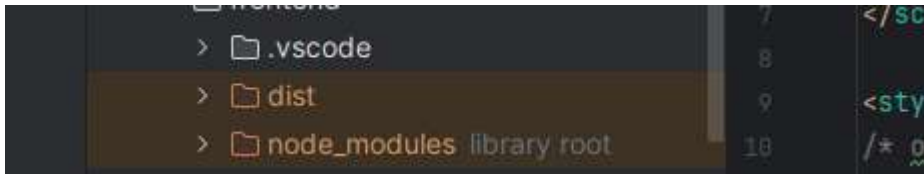
5) Integrar el frontend dentro de Spring Boot (para la demo)

5.1 Build del front

5.2 Copiar el contenido de dist/ a static/

Copia lo de adentro de frontend/dist/ a:

src/main/resources/static/



5.3 SPA routing simple (hash mode)

Para evitar 404 en rutas internas, cambia el router a **hash** y recompila:

```
rc/router/index.js
teRouter, createWebHashHistory ]
sView from '@/views/EquiposView.
t createRouter( options: { Show usa
createWebHashHistory(),
```

5.4 Botón en el panel (Thymeleaf) para abrir la SPA

En templates/dashboard/docente.html:

```
<a class="card-opcion" href="/index.html#/equipos">Inventario de
Equipos</a>
```

5.5 Probar integrado

- Se levanta Levanta Spring Boot (8081).
- <http://localhost:8081/index.html#/equipos>.

Inventario de Equipos

Filtrar por laboratorio:

ID	Código	Tipo	Marca	Modelo	Estado	Laboratorio	Acciones
2	C2	PC	HP	Model QFM	activo	2	<input type="button" value="Editar"/> <input type="button" value="Eliminar"/>
3	C3	PC	HP	Model IHT	activo	3	<input type="button" value="Editar"/> <input type="button" value="Eliminar"/>
4	C4	PC	HP	Model QEX	activo	4	<input type="button" value="Editar"/> <input type="button" value="Eliminar"/>
5	C5	PC	HP	Model XSL	activo	5	<input type="button" value="Editar"/> <input type="button" value="Eliminar"/>
1	C1	PC	HP	Model TXN	activo	3	<input type="button" value="Editar"/> <input type="button" value="Eliminar"/>