Informe Comparativo: Implementación de Ejercicios Angular vs. Vue.js

Jesus Machado C.I: 28.553.080
Luis Augusto Sandoval C.I: 26.781.082
Cesar Campo C.I: 28.022.781
Yhonka Machado C.I: 27.718.080

13 de abril de 2025

Índice

1. Introducción

Este informe presenta un análisis comparativo detallado entre dos implementaciones de los ejercicios vistos en clase. Una versión fue desarrollada utilizando el framework **Vue.js** (código original proporcionado) y la otra utilizando **Angular** (nueva implementación).

El objetivo es destacar las diferencias clave en el enfoque de desarrollo, centrándonos en aspectos específicos como el manejo de vistas, el estado y los "hooks"(ciclo de vida y estado en Angular), la gestión de datos (fetching/almacenamiento) y la reactividad. Además, se incluye una comparación general entre Angular y Vue.js.

La implementación de Angular se realizó utilizando **Angular CLI**, **TypeScript** como lenguaje principal, componentes **standalone** y **Bootstrap** para los estilos base, mientras que la versión original utiliza **Vue 3** con **Composition API**, **Vite** como herramienta de construcción y **Vuetify** como librería de componentes UI.

2. Análisis Comparativo Detallado (Angular vs. Vue.js)

A continuación, se detallan las diferencias observadas en los aspectos solicitados, contrastando la implementación en Angular con la base de Vue.js.

2.1. Manejo de Vistas

2.1.1. Angular

En Angular, la gestión de vistas se basa en una arquitectura de **Componentes**. Cada componente típicamente separa su lógica, plantilla y estilos en archivos distintos:

- **Lógica del Componente (`.ts`):** Una clase TypeScript decorada con `@Component`, donde se define el estado (propiedades de la clase) y el comportamiento (métodos). Ejemplo: `books.component.ts`.
- **Plantilla (`.html`):** Un archivo HTML que define la estructura visual del componente. Utiliza una sintaxis específica de Angular con directivas estructurales como `*ngFor` (para iterar listas) y `*ngIf` (para renderizado condicional), y data binding (‘[]‘ para propiedades, ‘()‘ para eventos, ‘[()]‘ para two-way binding con `ngModel`). Ejemplo: `books.component.html`.
- **Estilos (`.scss` o `.css`):** Archivo para definir los estilos específicos del componente (encapsulados por defecto). Ejemplo: `books.component.scss`.

El enrutamiento se gestiona mediante el módulo `RouterModule`, configurando las rutas en un archivo (`app.routes.ts`) y utilizando el componente `<router-outlet>` para renderizar el componente activo.

Ejemplo (Template Binding en `books.component.html`):

```
<tr *ngFor="let book of (books | async)">
  <td>{{ book.id }}</td>
  <td>{{ book.title }}</td>
  <td>
    <span *ngFor="let author of book.authors; let i = index">
      {{ author.name }}<span *ngIf="i < book.authors.length - 1">, </span>
    </span>
  </td>
  <td>{{ book.download_count }}</td>
  <td>
    <!-- Event Binding con (click) -->
    <button class="btn btn-sm btn-info" (click)="editBook(book)">Editar</button>
    <!-- Two-way Binding con [(ngModel)] en el formulario -->
    <input type="text" [(ngModel)]="newBook.title" name="title">
```

```
  </td>
</tr>
```

2.1.2. Vue.js (Código Original)

Vue.js utiliza **Single File Components (SFC)** con extensión `.vue`. Estos archivos agrupan la plantilla, la lógica y los estilos en un solo lugar:

- **Plantilla (`<template>`):** HTML enriquecido con directivas de Vue como `v-for`, `v-if`, y sintaxis de binding (':' o 'v-bind:' para props, '@' o 'v-on:' para eventos, 'v-model' para two-way binding).
- **Lógica (`<script setup lang="ts»`):** Utiliza la Composition API, donde la lógica se organiza mediante funciones componibles. El estado y los métodos se definen directamente dentro del `setup`.
- **Estilos (`<style scoped>`):** CSS/SCSS/Less que se aplican únicamente al componente actual si se usa el atributo 'scoped'.

El enrutamiento se configura con `vue-router` (archivo `router/index.ts`) y se usa el componente `<RouterView>` para mostrar la vista correspondiente. La implementación original usa **Vuetify** para los componentes UI, lo que abstrae mucho HTML y CSS.

2.1.3. Diferencias Clave

- **Estructura de Archivos:** Angular separa lógica, plantilla y estilos por defecto; Vue los agrupa en SFCs.
- **Sintaxis de Plantillas:** Diferentes directivas ('*ngFor' vs. 'v-for', '*ngIf' vs. 'v-if') y sintaxis de binding ('[]'/'()' vs. ':'/'@'). Angular es más explícito en la separación de bindings.
- **Organización de la Lógica:** Clases TypeScript con decoradores en Angular vs. Composition API funcional en '<script setup>' de Vue 3.
- **UI Components:** Angular usó Bootstrap manualmente, Vue usó la librería Vuetify integrada.

2.2. Uso de Hooks y State

2.2.1. Angular

Angular no utiliza el término "Hooks"de la misma manera que React o Vue. Los conceptos equivalentes son:

- **Ciclo de Vida (Lifecycle Hooks):** Son métodos específicos que se pueden implementar en la clase del componente (e.g., 'ngOnInit', 'ngOnDestroy', 'ngOnChanges'). Se definen como métodos de la clase que implementan interfaces predefinidas (e.g., 'OnInit').
- **Estado (State):** El estado del componente se gestiona principalmente mediante **propiedades** de la clase TypeScript. Para estados más complejos o compartidos entre componentes, se utilizan **Servicios** inyectables y **RxJS** (e.g., 'BehaviorSubject' o 'Subject') para manejar flujos de datos reactivos.
- **Comunicación:** '@Input()' para pasar datos del padre al hijo, '@Output()' junto con 'EventEmitter' para emitir eventos del hijo al padre.

Ejemplo (Estado y Ciclo de Vida en `books.component.ts`):

```typescript
import { Component, OnInit } from '@angular/core';
import { BehaviorSubject } from 'rxjs'; // Para estado reactivo
import { Book } from './book.model';
// ... otras importaciones

@Component({ /* Decorador */ })
export class BooksComponent implements OnInit { // Implementa interfaz de ciclo de vida

  // Estado gestionado como propiedades de clase
  books: BehaviorSubject<Book[]> = new BehaviorSubject<Book[]>([]); // Estado reactivo con RxJS
  selectedBook: Book | null = null; // Estado simple
  newBook: Partial<Book> = { /*...*/ }; // Estado para el formulario
  authorInput: string = '';

  constructor(private http: HttpClient) { } // Inyeccion de dependencias

  // Metodo del ciclo de vida OnInit
  ngOnInit(): void {
    this.loadBooks();
  }

  loadBooks(): void {
    this.http.get<any>('/json/fantasy-books.json').subscribe(data => {
      this.books.next(data.results); // Actualiza el BehaviorSubject
    });
  }
  // ... otros metodos (addBook, editBook, etc.)
}
```

2.2.2. Vue.js (Código Original)

Vue 3 con Composition API maneja estado y ciclo de vida así:

- **Estado Reactivo:** Utiliza funciones como ‘ref()‘ para tipos primitivos y ‘reactive()‘ para objetos, creando referencias reactivas. Cualquier cambio en estas referencias actualiza automáticamente la vista.
- **Ciclo de Vida (Lifecycle Hooks):** Son funciones importadas directamente (e.g., ‘onMounted‘, ‘onUpdated‘, ‘onUnmounted‘) y llamadas dentro del bloque ‘<script setup>‘.
- **Comunicación:** Props definidos con ‘defineProps()‘ y eventos emitidos con ‘defineEmits()‘.

Ejemplo conceptual (Estado y Ciclo de vida en Vue):

```typescript
import { ref, onMounted } from 'vue';
import axios from 'axios'; // Dependencia externa para fetching

// <script setup lang="ts">
const books = ref([]); // Estado reactivo con ref()
const isLoading = ref(true);

const fetchBooks = async () => {
  try {
    const response = await axios.get('/json/fantasy-books.json');
    books.value = response.data.results; // Actualiza el valor de ref
  } catch (error) {
    console.error(error);
  } finally {
    isLoading.value = false;
  }
```

```
};

// Hook de ciclo de vida
onMounted(() => {
  fetchBooks();
});
// </script>
```

2.2.3. Diferencias Clave

- **Concepto de "Hooks":** Angular usa métodos de ciclo de vida en clases; Vue 3 usa funciones de ciclo de vida importables en ‘setup‘.
- **Manejo del Estado Reactivo:** Angular requiere el uso explícito de RxJS (‘BehaviorSubject‘, ‘async‘ pipe) o confiar en la detección de cambios de Zone.js para la reactividad; Vue tiene reactividad fina integrada con ‘ref‘/‘reactive‘.
- **Estructura:** Estado como propiedades de clase en Angular vs. variables reactivas declaradas en ‘setup‘ en Vue.

2.3. Gestión del Almacenamiento (Fetching y State Management)

2.3.1. Angular

- **Fetching HTTP:** Se utiliza el módulo **`HttpClient`** incorporado, que retorna **Observables** de RxJS. Esto permite manejar respuestas asíncronas de forma reactiva, encadenar operaciones, cancelar peticiones, etc. El módulo se provee mediante inyección de dependencias (`provideHttpClient()`) y se inyecta en los componentes o servicios.
- **Gestión de Estado (Componente):** Como se mencionó, estado simple en propiedades de clase.
- **Gestión de Estado (Aplicación/Compartido):** Se fomenta el uso de **Servicios** inyectables para encapsular la lógica de negocio y el estado compartido. Estos servicios pueden usar RxJS (‘BehaviorSubject‘) para proporcionar datos reactivos a múltiples componentes. Para aplicaciones muy grandes, existen librerías como NgRx (inspirada en Redux).
- **Almacenamiento Local:** Acceso directo a ‘localStorage‘ o ‘sessionStorage‘ mediante las APIs del navegador.

El uso del **pipe `async`** en las plantillas es una práctica común para suscribirse (y desuscribirse automáticamente) a Observables.

Ejemplo (Uso de `HttpClient` y `async` pipe):

```
// En books.component.ts
loadBooks(): void {
  // HttpClient retorna un Observable
  this.http.get<any>('/json/fantasy-books.json').subscribe(data => {
    this.books.next(data.results); // Actualiza el BehaviorSubject
  });
}

// En books.component.html
<tr *ngFor="let book of (books | async)">
  <td>{{ book.title }}</td>
</tr>
```

2.3.2. Vue.js (Código Original)

- **Fetching HTTP:** No tiene un cliente HTTP incorporado. Se suelen usar librerías externas como **`axios`** (como indica el `package.json` original) o la API nativa `fetch`. Generalmente se trabaja con **Promesas** y `async/await`.
- **Gestión de Estado (Componente):** Uso de 'ref' y 'reactive'.
- **Gestión de Estado (Aplicación/Compartido):** Librerías como **Pinia** (recomendada actualmente) o Vuex. También se pueden usar 'provide'/'inject' o simples Componibles ('composables') para compartir lógica y estado.
- **Almacenamiento Local:** Acceso directo a las APIs del navegador.

2.3.3. Diferencias Clave

- **Cliente HTTP:** Angular tiene 'HttpClient' integrado (basado en RxJS); Vue depende de librerías externas (e.g., 'axios', basado en Promesas).
- **Manejo Asíncrono:** RxJS (Observables) es central en Angular; Promesas ('async/await') son más comunes en Vue (aunque se puede usar RxJS).
- **Estado Global:** Servicios con RxJS o NgRx en Angular vs. Pinia/Vuex en Vue. Angular tiene un fuerte énfasis en la Inyección de Dependencias para los servicios.
- **Integración:** El 'async' pipe de Angular simplifica el manejo de Observables en las plantillas.

2.4. Manejo de la Reactividad

2.4.1. Angular

La reactividad en Angular se basa principalmente en **Zone.js**. Esta librería "parchea"(monkey-patches) las APIs asíncronas del navegador (como 'setTimeout', 'addEventListener', 'Promise.then', XHR). Cuando una de estas operaciones finaliza, Zone.js notifica a Angular, que dispara un ciclo de **detección de cambios**.

- **Detección de Cambios:** Angular recorre el árbol de componentes desde la raíz hacia las hojas, comparando los valores actuales de las propiedades vinculadas en la plantilla con sus valores anteriores. Si hay diferencias, actualiza el DOM.
- **Optimización:** Se puede usar 'ChangeDetectionStrategy.OnPush' en los componentes para limitar cuándo se ejecuta la detección de cambios (solo cuando cambian los '@Input' o se dispara un evento explícitamente desde el componente o sus hijos, o cuando un Observable al que se está suscrito con 'async' pipe emite un valor).
- **RxJS:** Aunque no es el mecanismo *fundamental* de detección de cambios del DOM, RxJS es la herramienta principal para *gestionar flujos de datos reactivos* de manera explícita dentro de la lógica de la aplicación.

La reactividad no es tan .ªutomágica.ª nivel de variables individuales como en Vue; depende de que Angular detecte que algo *podría* haber cambiado.

2.4.2. Vue.js (Código Original)

Vue 3 utiliza un sistema de reactividad basado en **Proxies** de JavaScript.

- **Seguimiento Fino:** Cuando se accede a una propiedad de un objeto reactivo (‘ref‘ o ‘reactive‘) dentro de una función de renderizado o un efecto (‘computed‘, ‘watch‘), Vue registra esa dependencia.
- **Actualización Automática:** Cuando esa propiedad cambia, Vue sabe exactamente qué componentes o efectos dependen de ella y los vuelve a ejecutar de forma eficiente.
- **Transparencia:** El desarrollador trabaja con variables JavaScript normales (envueltas en ‘ref‘ o ‘reactive‘), y el sistema se encarga de la reactividad de forma transparente.

Este sistema es generalmente más eficiente en términos de no necesitar revisar todo el árbol de componentes si solo una pequeña parte del estado cambió.

2.4.3. Diferencias Clave

- **Mecanismo Subyacente:** Zone.js + Detección de Cambios (revisión del árbol) en Angular vs. Proxies + Seguimiento de Dependencias (actualización precisa) en Vue 3.
- **Granularidad:** La reactividad de Vue es más fina y automática a nivel de variable individual. La de Angular es más a nivel de ciclo de detección (aunque optimizable).
- **Dependencias Externas:** Angular depende de Zone.js (aunque hay iniciativas para hacerlo opcional); Vue no tiene esta dependencia externa para su reactividad central.
- **Manejo Explícito:** Angular fomenta el uso de RxJS para manejar explícitamente flujos de datos reactivos.

3. Comparación General: Angular vs. Vue.js

Desde la perspectiva del equipo que trabajó con Angular, aquí se presenta una comparación general con Vue.js, basada en la experiencia y las características observadas.

3.1. Facilidad de Aprendizaje

- **Vue.js:** Generalmente percibido como más fácil de aprender, especialmente para desarrolladores con experiencia en HTML, CSS y JavaScript básicos. Su sintaxis de plantilla es intuitiva y el framework es progresivo (se pueden adoptar características avanzadas gradualmente). La Composition API, aunque potente, introduce un paradigma ligeramente diferente al de la Options API tradicional.
- **Angular:** Presenta una curva de aprendizaje más pronunciada. Requiere una comprensión sólida de **TypeScript**, conceptos como **Inyección de Dependencias**, **Módulos** (aunque menos relevantes con componentes standalone), **Decoradores**, y especialmente **RxJS**. La estructura opinionada y la cantidad de conceptos a asimilar desde el principio pueden ser desafiantes para los principiantes. Sin embargo, esta estructura aporta beneficios en proyectos grandes.

3.2. Usabilidad (Experiencia de Desarrollo - DX)

- **Vue.js:** Ofrece gran flexibilidad (Options API vs. Composition API). Las SFCs son convenientes para muchos desarrolladores. Las herramientas como Vite proporcionan una experiencia de desarrollo muy rápida (HMR instantáneo). Es menos opinionado, lo que da más libertad pero puede llevar a inconsistencias si no se establecen convenciones.
- **Angular:** Es muy **opinionado**, lo que significa que impone una estructura y forma de trabajar. Esto puede ser visto como una ventaja en equipos grandes, ya que promueve la **consistencia**. El **Angular CLI** es extremadamente potente para generar código (componentes, servicios, módulos), realizar actualizaciones, linting, testing, etc. La integración profunda con TypeScript es una gran ventaja para la detección temprana de errores y el refactoring en aplicaciones complejas. RxJS, aunque complejo, es muy poderoso para manejar lógica asíncrona compleja. La introducción de componentes *standalone* ha simplificado parte de la configuración inicial.

3.3. Características Distintivas

- **Vue.js:**
  - **Simplicidad y Progresividad:** Fácil de empezar, se adapta desde proyectos pequeños a grandes.
  - **Flexibilidad:** Opciones para organizar el código (Options vs. Composition).
  - **Rendimiento:** Reactividad fina basada en Proxies muy eficiente.
  - **SFCs:** Combinación conveniente de plantilla, lógica y estilos.
- **Angular:**
  - **Framework Completo ("Batteries Included"):** Incluye soluciones integradas para routing, HTTP, gestión de estado básica (servicios), formularios, etc.
  - **TypeScript como Ciudadano de Primera Clase:** Fuerte tipado y herramientas avanzadas desde el inicio.
  - **Inyección de Dependencias (DI):** Sistema robusto para gestionar servicios y dependencias, facilitando el testing y la modularidad.
  - **RxJS:** Integración profunda para programación reactiva y manejo de flujos de datos asíncronos.
  - **Angular CLI:** Herramienta de línea de comandos muy completa que estandariza el desarrollo.
  - **Arquitectura Opinionada:** Ideal para grandes aplicaciones empresariales donde la estructura y consistencia son cruciales.
  - **Mecanismo de Detección de Cambios (Zone.js):** Aunque diferente a Vue, funciona robustamente para la mayoría de los casos.

4. Conclusión

La implementación del proyecto en Angular ha puesto de manifiesto las diferencias fundamentales en la filosofía y arquitectura respecto a Vue.js. Angular proporciona un entorno de desarrollo más estructurado y opinionado, con herramientas integradas como TypeScript, RxJS y una potente CLI, lo cual es beneficioso para aplicaciones complejas y equipos grandes. Su sistema de componentes, inyección de dependencias y manejo de estado a través de servicios con RxJS ofrecen un patrón claro para construir aplicaciones escalables.

Por otro lado, Vue.js (observado en el código original) destaca por su flexibilidad, facilidad de adopción y un sistema de reactividad más directo y transparente basado en Proxies. La elección entre uno y otro dependerá de las necesidades específicas del proyecto, el tamaño del equipo, la experiencia previa y las preferencias sobre la estructura y las herramientas. Ambos frameworks son capaces y modernos, pero abordan el desarrollo frontend desde perspectivas distintas.