Universidad de Carabobo

Facultad Experimental de Ciencias y Tecnología

Departamento de Computación

Unidad de Desarrollo Web

EXPLORANDO OTROS FRAMEWORKS:

REACT Y VUE

TUTOR:

Prof. Javier Escobar

ALUMNO:

Breto Kevin C.I.: 28.351.815

Diaz Gerardo C.I.: 30.388.971

López José C.I: 30.077.008

Mendoza Ulises C.I.: 27.518.340

BÁRBULA, ABRIL 2025

1. Introducción

Este informe detalla el análisis de una aplicación web desarrollada con Next.js y React. Se examinan las prácticas implementadas en el manejo de vistas, el uso de hooks y estado, la gestión del almacenamiento (limitada al alcance del código), así como también el manejo de la reactividad. Para cada aspecto analizado en React/Next.js, se realizará una comparación directa observando cómo se abordaría una funcionalidad equivalente en Vue.js, destacando las diferencias y similitudes entre estos dos. Se utilizarán ejemplos concretos extraídos del código fuente proporcionado (page.tsx, Table.tsx, BookForm.tsx) como base para la explicación y la comparación.

2. Análisis del Código Fuente y Comparación con Vue.js

2.1. Manejo de Vistas

React/Next.js: La UI se construye componiendo componentes funcionales escritos con JSX. El componente padre (page.tsx) organiza y renderiza componentes hijos (BookForm, Table), pasándoles datos y funciones mediante props.

Ejemplo (page.tsx):

```
// page.tsx

<main>

  {/* ... */}

  <BookForm onAdd={addBook} onUpdate={updateBook} ... />

  {/* ... */}

  <Table data={tableData} onDelete={deleteBook} ... />

</main>
```

Comparación con Vue.js: Vue también es un framework basado en componentes. La estructura sería similar, con un componente padre (ej: HomePage.vue) que importa y utiliza componentes hijos (<BookForm>, <DataTable>) dentro de su sección <template>. La sintaxis de la plantilla es más cercana a HTML estándar en lugar de JSX.

```
<!-- HomePage.vue (Template) -->
<template>
  <main>
    <!-- ... -->
    <BookForm @add="addBook" @update="updateBook" :update-data="updateData" ...
/>
    <!-- ... -->
    <DataTable :data="tableData" @delete="deleteBook" ... />
  </main>
</template>
```

Dando como resultado una composición de componentes muy similar en ambos

React/Next.js: El renderizado condicional se logra usando expresiones JavaScript (como operadores ternarios o &&) dentro de JSX.

Ejemplo (BookForm.tsx - Botones):

```
// BookForm.tsx
<div>
    {!isUpdate ? ( <button>Agregar</button> ) : ( <div> /* Botones
Actualizar/Cancelar */ </div> )}
</div>
```

Comparación con Vue.js: Vue utiliza directivas estructurales directamente en la plantilla HTML, lo cual muchos encuentran más declarativo para este propósito. Se usaría v-if, v-else-if, y v-else.

```
<!-- BookForm.vue (Template) -->
<div>
  <button v-if="!isUpdate">Agregar</button>
  <div v-else>
    <button>Actualizar</button>
    <button @click="handleCancel">Cancelar</button>
  </div>
</div>
```

Ambos logran el mismo resultado, pero la sintaxis difiere: JS en JSX vs. directivas en plantilla.

React/Next.js: La iteración sobre listas de datos se realiza comúnmente con el método .map() de JavaScript dentro de JSX, generando un array de elementos JSX. Se requiere una key única para cada elemento.

Ejemplo (Table.tsx - Filas):

```
// Table.tsx
<tbody>
  {data.map((row: Data) => (
    <tr key={row.id}>
      {/* ... celdas ... */}
    </tr>
  ))}
</tbody>
```

Comparación con Vue.js: Vue proporciona la directiva v-for, que se considera muy idiomática y clara para renderizar listas. También requiere una :key.

```
<!-- DataTable.vue (Template) -->

<tbody>

  <tr v-for="row in data" :key="row.id">

    <!-- Acceso directo a propiedades: {{ row.name }} -->

    <td v-for="column in columns" :key="column.key">{{ row[column.key] }}</td>

    <!-- ... celdas ... -->

  </tr>

</tbody>
```

Nuevamente, el concepto es idéntico (iterar y asignar clave), pero la implementación sintáctica varía (JS .map() vs. directiva v-for).

2.2. Uso de Hooks y State

React/Next.js: El estado local se maneja principalmente con el hook useState, que devuelve el valor del estado y una función para actualizarlo. Llamar a la función de actualización dispara un re-renderizado del componente.

Ejemplo (page.tsx):

```
// page.tsx

const [tableData, setTableData] = useState<Data[]>(books);

const deleteBook = (id: number) => {

  setTableData(currentData => currentData.filter(item => item.id !== id)); //
Actualiza -> Re-render

};
```

Comparación con Vue.js: Vue maneja el estado reactivo de forma diferente:

Composition API: Se usan ref() para valores primitivos/simples y reactive() para objetos. Las mutaciones a la propiedad .value de un ref o a las propiedades de un objeto reactive son rastreadas por Vue, y las actualizaciones del DOM ocurren automáticamente donde sea necesario.

```
// HomePage.vue (Script setup)

import { ref } from 'vue';

const tableData = ref([...books]); // [...] para crear copia si books viene de
fuera

const deleteBook = (id) => {

  tableData.value = tableData.value.filter(item => item.id !== id); // Mutación
directa -> Reactividad

};
```

Options API: El estado se declara en el objeto devuelto por la función data(). Vue hace que estas propiedades sean reactivas. Las mutaciones directas (ej: this.tableData.push(...) o this.tableData = ...) suelen ser suficientes.

La diferencia entre ambos: explícita vía setters en React vs. más implícita/automática basada en proxies/seguimiento de dependencias en Vue**.**

React/Next.js: Los efectos secundarios (como reaccionar a cambios en props o estado) se gestionan con el hook useEffect. Se especifican dependencias para controlar cuándo se ejecuta el efecto.

Ejemplo (BookForm.tsx): Reacciona al cambio de la prop updateData para poblar el formulario

```tsx
// BookForm.tsx

useEffect(() => {

    if (updateData) {

      setFormData(updateData);

      setIsUpdate(true);

    } else { /* reset form */ }

  }, [updateData]); // Se ejecuta si updateData cambia
```

Comparación con Vue.js: Vue ofrece varias formas de manejar efectos secundarios y reacciones a cambios:

Composition API: watch() es el equivalente más directo a useEffect con dependencias. Observa una fuente específica (prop, ref, reactive, getter) y ejecuta una callback cuando cambia. watchEffect() es similar pero rastrea automáticamente sus dependencias internas y se re-ejecuta si cambian.

```vue
// BookForm.vue (Script setup)

import { watch, reactive, ref } from 'vue';

const props = defineProps(['updateData']); // Define prop

const formData = reactive({...defaultForm});

const isUpdate = ref(false);

watch(() => props.updateData, (newVal) => { // Observa la prop directamente

  if (newVal) {

    Object.assign(formData, newVal); // Actualiza el objeto reactivo

    isUpdate.value = true;

  } else { /* reset form */ }

}, { immediate: true }); // immediate: true para ejecutar al inicio si es necesario
```

Options API: Se utiliza la opción watch en la definición del componente. Las propiedades computed también son muy importantes para derivar estado y a menudo reducen la necesidad de watchers.

Vue separa más explícitamente la observación (watch) de los cálculos derivados (computed), mientras que useEffect en React puede usarse para ambos patrones

2.3. Gestión del Almacenamiento (Local y Comunicación entre Componentes)

React/Next.js: En este código, la gestión se basa en estado local (useState en BookForm) y el patrón "levantar el estado" (page.tsx gestiona tableData y pasa datos/funciones a Table y BookForm vía props).

Ejemplo: page.tsx define addBook y lo pasa como onAdd a BookForm. BookForm llama a props.onAdd(formData) al enviar.

```
// page.tsx -> Pasa la función

<BookForm onAdd={addBook} ... />

// BookForm.tsx -> Llama la función recibida por props

const handleSubmit = (e: FormEvent) => { /* ... */ if (onAdd) onAdd(formData);
/* ... */ };
```

Comparación con Vue.js: El patrón de "levantar el estado" es idéntico conceptualmente. La diferencia está en la comunicación hacia arriba (del hijo al padre). En lugar de pasar funciones callback como props, Vue utiliza un sistema de emisión de eventos. El hijo emite un evento (emit('eventName', payload)) y el padre escucha ese evento en la plantilla (@eventName="handler").

```
<!-- HomePage.vue (Template escuchando el evento) -->

<BookForm @add-book="handleAddNewBook" ... />
```

```
// BookForm.vue (Script setup emitiendo el evento)

import { defineEmits } from 'vue';

const emit = defineEmits(['add-book']); // Declara los eventos que puede emitir

const handleSubmit = () => { /* ... */ emit('add-book', formData); /* ... */ };
```

El flujo de datos (props hacia abajo) es igual. La comunicación hacia arriba (eventos en Vue vs. callbacks en React) es la principal diferencia mecánica, aunque logran el mismo objetivo.

2.4. Manejo de la Reactividad

React/Next.js: La reactividad se basa en la inmutabilidad y la detección de cambios de estado (vía llamadas a setters de useState). Cuando el estado cambia, React re-renderiza el componente y sus hijos, compara el Virtual DOM y aplica los cambios al DOM real. El flujo de datos es unidireccional (props abajo, callbacks/eventos arriba).

Ejemplo Flujo Eliminar: Click en Table -> llama onDelete (prop) -> ejecuta deleteBook en page.tsx -> setTableData con nuevo array -> React re-renderiza page y Table con nuevos datos.

Comparación con Vue.js: Vue utiliza un sistema de reactividad basado en seguimiento de dependencias (generalmente mediante Proxies de JavaScript). Cuando se accede a una propiedad reactiva (ref, reactive, data) durante el renderizado o en un computed/watch, Vue registra esa dependencia. Cuando la propiedad se modifica, Vue sabe exactamente qué partes de la aplicación (componentes, watchers, etc.) dependen de ella y las actualiza de forma granular.

Ejemplo Flujo Eliminar Vue: Click en DataTable -> Emite evento delete con id -> HomePage escucha @delete, ejecuta deleteBook -> deleteBook muta tableData.value (Composition API) o this.tableData (Options API) -> El sistema de reactividad de Vue detecta el cambio y actualiza automáticamente sólo la parte del DOM correspondiente a la lista en DataTable, sin necesariamente re-renderizar todo el componente HomePage.

3. Conclusión

La aplicación React/Next.js analizada utiliza los componentes funcionales, JSX, y los hooks (useState, useEffect) para crear una interfaz interactiva. Su manejo del estado y la comunicación sigue el patrón común de "levantar el estado". Mientras que la comparación integrada con Vue.js revela que, si bien ambos comparten principios fundamentales como la componentización y el flujo de datos unidireccional, difieren notablemente en la sintaxis (JSX vs. Plantillas HTML + Directivas), el manejo del estado y la reactividad (Hooks explícitos vs. Sistema de seguimiento de dependencias más implícito), y los mecanismos de comunicación hijo-padre (Callbacks vía props vs. Emisión de eventos).

La elección entre ellos a menudo se reduce a la preferencia por una sintaxis u otra, la curva de aprendizaje percibida, y el ecosistema específico que rodea a cada framework. El código proporcionado sirve como un buen ejemplo de una implementación básica pero clara de los conceptos de React en una aplicación CRUD simple.

4. Bibliografía

- React. (s/f). React.dev. Recuperado el 10 de abril de 2025, de https://react.dev/
- *Vue.Js*. (s/f). Vuejs.org. Recuperado el 10 de abril de 2025, de https://vuejs.org/