# Práctica 7.1. Ejercicios Vue vol. 3

Realizar los siguientes ejercicios usando Vue 3 con el Composition API, el lenguaje TS y la sintaxis de <script setup>

## 1. Utilización de herramientas de consultas.

### 1.1 Postman.

a. Instala postman, si todavía no lo tienes y realiza una petición GET a la API REST Countries. Accede al endpoint /v3.1/name/{name} y prueba a pedir varios países
b. Simula la creación de un Recurso (POST Request). Utiliza https://beeceptor.com/ (este servicio permite crear endpoints temporales para probar peticiones POST) y define un endpoint en el servicio de mock API
c. Autenticación con API Keys. Utiliza Postman para hacer una petición a la API autenticada

### 1.2 RapidAPI.

a. **Instala RapidAPI client** en vscode. Date de **alta en RapidAPI**. Crea una nueva petición HTTP en RapidAPI Client a una API que tenga endpoints: https://[nombre-api].rapidapi.com/[endpoint]. Configura la Autenticación con API Key.
b. **Creación de Datos** (CREATE - POST) y Manejo del Cuerpo de la Petición: elige una API en RapidAPI que permita crear recursos vía POST. Busca APIs de "Tareas", "Listas de Tareas", "Notas" o similares
c. **Actualización de Datos** (UPDATE - PUT/PATCH) y Parámetros en la URL. Realizar peticiones PUT o PATCH para actualizar recursos existentes
d. **Eliminación de Datos (DELETE)**. Realizar peticiones DELETE para eliminar recursos
e. **Autenticación Avanzada con OAuth 2.0**. Busca una API en RapidAPI que utilice OAuth 2.0, y configura OAuth 2.0 en RapidAPI Client. Obtener un Token de Acceso y úsalo para realizar una petición.

## 2. Uso de la biblioteca Axios.

a. **Crea una aplicación web en Vue que haga uso de axios**. Instala axios.
b. **Petición GET.** Elige una API y realiza desde la aplicación una petición GET
c. **Petición POST.** Enviar datos con POST y recibe confirmación del servidor. Realiza peticiones POST con Axios para enviar datos al servidor, utilizando el formato JSON en el cuerpo de la petición
d. **Configuración de Peticiones**: Headers Personalizados y Parámetros de Consulta. Usa una API que permita Headers Personalizados o Parámetros de Consulta: REST Countries o JSONPlaceholder son suficientes para este ejercicio.
e. **Interceptors para Loggear Peticiones y Respuestas** (o para Autenticación). Define Interceptores para Peticiones y Respuestas: Los interceptores se definen a nivel de la instancia de axios (o directamente en axios si no estás usando instancias)
f. **Manejo Avanzado de Errores y Cancelación de Peticiones**. Simula diferentes tipos de errores, con los siguientes escenarios:
    - **Error de Red (Network Error):** Intenta hacer una petición a una URL inexistente o desconecta temporalmente la red de tu ordenador para forzar un error de red
    - **Error 404 (Not Found) u otro Error de Estado HTTP:** Haz una petición a un endpoint que sabes que devuelve un error 404 (por ejemplo, a un recurso inexistente en JSONPlaceholder, como https://jsonplaceholder.typicode.com/posts/9999). O puedes usar REST Countries y pedir un país que no existe.
    - **Timeout:** Configura un timeout muy corto en la configuración de Axios y haz una petición a una API que sabes que tarda un poco en responder, para que la petición exceda el timeout.
    - **Error en el Lado del Cliente (ej. error en el `then()`):** Simplemente lanza un error dentro de la función .then() para ver cómo .catch() lo captura

g. **Manejo de errores.** Maneja Errores en el Bloque .catch(): En el bloque .catch() de tus peticiones Axios, inspecciona el objeto error para identificar el tipo de error y manejarlo de forma apropiada. Axios proporciona información útil en el objeto error

**3. Generador de imágenes con IA:** Usa RapidApi y date de alta en **Flux** u otra **API de generación de imágenes con IA**. Crea una app en Vue para generar imágenes a partir de un prompt. Usa Axios como biblioteca para realizar la petición a la API
Despliega la aplicación en alguna web parecida a Vercel, Netlify, Heroku, GitHub Pages, etc y pon la ruta aquí

**4. API simulada de Biblioteca Universitaria con Beeceptor.** Realiza una aplicación en Vue que haga uso de una API creada con Beeceptor que simula una API de una biblioteca universitaria, que te permita interactuar con libros, préstamos y un perfil de estudiante
Para ello sigue los pasos:

a) Visita https://beeceptor.com y crea un nuevo endpoint (por ejemplo, "biblioteca-universitaria-api")
b) Configura los siguientes endpoints: GET /libros
respuesta:

```
[
 {"id": 1, "titulo": "Introducción a la Programación", "autor": "Ada Lovelace", "disponible": true},
 {"id": 2, "titulo": "Física Cuántica para Principiantes", "autor": "Marie Curie", "disponible": false},
 {"id": 3, "titulo": "El Arte de la Procrastinación", "autor": "John Perry", "disponible": true}
]
```

c) GET /libros/:id

```
"id": 1,
"titulo": "Introducción a la Programación",
"autor": "Ada Lovelace",
"disponible": true,
"descripcion": "Un libro fundamental para comenzar en el mundo de la programación",
"fechaPublicacion": "2022-01-15"
```

```
}
```

d) POST /prestamos

```
{"mensaje": "Préstamo realizado con éxito", "fechaDevolucion": "2025-03-05"}
```

e) GET /perfil

```
{
  "id": "12345",
  "nombre": "María García",
  "carrera": "Ingeniería Informática",
  "librosPrestados": [
    {"id": 2, "titulo": "Física Cuántica para Principiantes", "fechaDevolucion": "2025-03-01"}
  ]
}
```

La App debe:

1. Mostrar una lista de libros disponibles al cargar
2. Permitir ver detalles de un libro al hacer clic en él
3. Implementar un sistema de "préstamo" de libros
4. Mostrar el perfil del estudiante con sus libros prestados
5. Las peticiones realizadas deben tener gestión de errores.

**5. Monitoreo de carga con axios:** Crea una pequeña aplicación en Vue 3 que permita probar la característica: “Monitoreo de carga: Axios permite seguir el progreso de las solicitudes y descargas, algo que fetch no ofrece de forma nativa”

Para ello debes realizar una petición get, usando Axios, a un fichero grande e ir mostrando el progreso de carga

En la petición no olvides poner:

```
responseType: 'blob', // Importante para descargar archivos binarios
```

y crear un enlace para descargar el archivo:

```
const url = window.URL.createObjectURL(new Blob([respuesta.data]));
```

**6. Explorador de Música con Spotify:** Crea una aplicación Vue 3, con axios, que use la API de Spotify (la puedes encontrar en RapidApi), pero debes hacerlo con https://developer.spotify.com/documentation/web-api/tutorials/getting-started y usar su sistema de autenticación

La aplicación permitiría a los usuarios buscar artistas, álbumes o canciones en Spotify y luego crear listas de reproducción personalizadas. Además, podrían explorar las listas de reproducción de otros usuarios o las listas temáticas de Spotify.

**Funcionalidades**

- **Autenticación con Spotify**: Los usuarios podrían iniciar sesión con sus cuentas de Spotify para acceder a sus datos y crear sus propias listas de reproducción. Implementar el flujo de autenticación de OAuth 2.0 para obtener tokens de acceso y permisos para acceder a los datos de los usuarios
- **Búsqueda**: Una interfaz de búsqueda intuitiva para encontrar artistas, álbumes o canciones.
- **Visualización de resultados**: Mostrar los resultados de búsqueda de forma organizada, con imágenes de portada y detalles relevantes.
- **Creación y edición de listas de reproducción**: Permitir a los usuarios crear nuevas listas, agregar canciones, eliminar canciones y cambiar el nombre de las listas.
- **Explorar listas de reproducción**: Explorar listas de reproducción públicas de otros usuarios o listas temáticas de Spotify.
- **Reproducción de música**: Integrar un reproductor de música (podría ser el reproductor web de Spotify o uno personalizado) para escuchar las canciones directamente desde la aplicación.

Despliega la aplicación en alguna web parecida a Vercel, Netlify, Heroku, GitHub Pages, etc y pon la ruta aquí

## 7. Pruebas y documentación

### 7.1 Pruebas.

Partiendo de la aplicación que interactúa con una API simulada de biblioteca universitaria (creada con Beeceptor, ejercicio 4), el objetivo de este ejercicio es **diseñar e implementar un conjunto de pruebas unitarias y de componentes exhaustivas utilizando Vitest**. Estas pruebas deben asegurar la correcta funcionalidad de la aplicación, abarcando la gestión de la visualización de libros, los detalles de los libros, el sistema de préstamos, la información del perfil del estudiante y la gestión de errores en las peticiones a la API.

El endpoint de Beeceptor simulado para la API de la biblioteca universitaria es: `biblioteca-universitaria-api` (o el nombre que se haya configurado)

a) Realiza las pruebas Unitarias usando Vitest
b) Ejecuta un test de cobertura. (Puedes usar el proveedor v8 o istanbul)

**Tabla de Pruebas:**

| Característica/Componente a Probar | Caso de Prueba | Tipo de Prueba | Resultado Esperado |
|---|---|---|---|
| **Listado de Libros** | Cargar componente y verificar que se llama a la API para obtener libros. | Componente | La API es llamada al montar el componente. |
| **Listado de Libros** | Verificar que la lista de libros se renderiza correctamente con datos de la API. | Componente | Se muestra una lista de libros con título, autor y estado de disponibilidad. |
| **Listado de Libros** | Probar que se muestra un mensaje de "Cargando..." mientras se obtienen los datos. | Componente | Mientras se cargan los datos, se muestra un indicador de carga. |

| | | | |
|---|---|---|---|
| **Listado de Libros** | Simular un error en la API al obtener libros. | Componente | Se muestra un mensaje de error amigable al usuario. |
| **Detalles del Libro** | Navegar al detalle de un libro al hacer clic desde la lista. | Componente | Al hacer clic en un libro, la aplicación navega a la ruta de detalle del libro. |
| **Detalles del Libro** | Verificar que se muestra la información completa del libro en la página de detalle. | Componente | Se muestran título, autor, descripción, fecha de publicación y estado de disponibilidad del libro. |
| **Detalles del Libro** | Simular un error al obtener los detalles de un libro específico. | Componente | Se muestra un mensaje de error amigable al usuario si no se pueden cargar los detalles del libro. |
| **Detalles del Libro** | Probar la navegación a un detalle de libro con un ID inválido. | Componente | Se gestiona correctamente el ID inválido, mostrando un mensaje adecuado (ej. "Libro no encontrado"). |
| **Préstamo de Libros** | Simular un préstamo exitoso de un libro. | Integración | Al intentar prestar un libro, se llama a la API, y si es exitoso, se muestra un mensaje de éxito. |
| **Préstamo de Libros** | Verificar el mensaje de éxito tras un préstamo exitoso. | Integración | Después de un préstamo exitoso, se muestra un mensaje confirmando el préstamo y la fecha de devolución. |
| **Préstamo de Libros** | Simular un error en la API durante el proceso de préstamo. | Integración | Si la API devuelve un error al intentar prestar un libro, se muestra un mensaje de error al usuario. |
| **Perfil del Estudiante** | Cargar componente de perfil y verificar llamada a la API para obtener datos. | Componente | Al montar el componente de perfil, se realiza una petición a la API para obtener los datos del perfil. |

| | | | |
|---|---|---|---|
| **Perfil del Estudiante** | Verificar que se muestra la información del perfil del estudiante correctamente. | Componente | Se muestra el nombre, carrera y lista de libros prestados del estudiante. |
| **Perfil del Estudiante** | Verificar que se listan correctamente los libros prestados en el perfil. | Componente | Se muestra una lista de libros prestados con título y fecha de devolución para cada libro. |
| **Perfil del Estudiante** | Simular un error al obtener los datos del perfil del estudiante desde la API. | Componente | Se muestra un mensaje de error amigable al usuario si no se puede cargar el perfil del estudiante. |
| **Gestión de Errores Generales** | Simular un error de red al hacer cualquier petición a la API. | Integración | La aplicación debe manejar errores de red de forma global y mostrar un mensaje genérico de error de conexión. |
| **Gestión de Errores Generales** | Probar la gestión de diferentes códigos de error HTTP (404, 500) en las peticiones. | Integración | La aplicación debe interpretar los códigos de error HTTP y mostrar mensajes de error específicos o genéricos. |

# 2.- Información de interés

**RECURSOS NECESARIOS:**

- Ordenador personal con, al menos, 4 Gigabyte de memoria RAM.
- Conexión a Internet.
- Procesador de textos.
- Navegador web.
- Entorno de desarrollo web.
- Software para comprimir los archivos de la Tarea.

**WEBS DE INTERÉS:**

- Visual Studio Code
- Herramientas del desarrollador (firefox)
- NodeJs
- EsLint
- GitHub
- GitHub Pages

# 3.- Evaluación. Criterios de evaluación implicados.

**RA7: Desarrolla aplicaciones Web dinámicas, reconociendo y aplicando mecanismos de comunicación asíncrona entre cliente y servidor.**

Los criterios de evaluación implicados del **RA7** son los siguientes:

RA7.a) Se han evaluado las ventajas e inconvenientes de utilizar mecanismos de comunicación asíncrona entre cliente y servidor Web.

RA7.b) Se han analizado los mecanismos disponibles para el establecimiento de la comunicación asíncrona.

RA7.c) Se han utilizado los objetos relacionados.

RA7.d) Se han identificado sus propiedades y sus métodos.

RA7.e) Se ha utilizado comunicación asíncrona en la actualización dinámica del documento Web.

RA7.f) Se han utilizado distintos formatos en el envío y recepción de información.

RA7.g) Se han programado aplicaciones Web asíncronas de forma que funcionen en diferentes navegadores.

RA7.h) Se han clasificado y analizado librerías y frameworks que faciliten la incorporación de las tecnologías de actualización dinámica a la programación de páginas Web.

RA7.i) Se han creado y depurado programas que utilicen estas librerías y frameworks.

| Rúbrica de la tarea | | | |
|---|---|---|---|
| **Criterios de Evaluación (CE)** | **Ejercicio implicado** | **Calificación CE** | **Retroalimentación** |
| 7.a, 7.b, 7.c, 7.d, 7.f, 7.h | EJ1, EJ2 | | |
| 7.e, 7.f, 7.g, 7.h | EJ3, EJ4, EJ5, EJ6 | | |
| 7.i | EJ4, EJ7 | | |