

Programación Web Avanzada

PEC 3: La importancia del estado

En la PEC 3 vamos a abordar los conceptos principales de la programación reactiva, como es el concepto de estado y su gestión mediante eventos. Para lograrlo se va a continuar con el desarrollo del proyecto de nuestro gestor de series, al que se le añadirá el comportamiento necesario para poder crear y mostrar elementos de manera dinámica haciendo uso del concepto de estado.

Conocimientos, Habilidades y Competencias

En esta PEC se desarrollan los siguientes conocimientos (K), habilidades (S) y Competencias (C) del Grado en Multimedia 2023:

- **K1.** Identificar y diferenciar los elementos de la teoría de la multimedia para analizar, conceptualizar y diseñar productos interactivos multimedia.
- **S3.** Utilizar de manera adecuada los lenguajes de programación, las herramientas de desarrollo y las tecnologías disponibles para el análisis, el diseño y la implementación de aplicaciones multimedia.
- **S7.** Analizar y sintetizar información para evaluar soluciones tecnológicas y elaborar propuestas de proyectos multimedia teniendo en cuenta los recursos, las alternativas disponibles y las condiciones de mercado.

Objetivos

Los objetivos concretos de esta PEC son:

- Familiarizarse con el funcionamiento de Vue.js
- Conocer la gestión del estado en Vue.js
- Conocer conceptos básicos de componentes en Vue

Introducción

En esta PEC trabajaremos fundamentalmente las técnicas de programación reactiva y, en particular, las veremos en el contexto del framework Vue. La programación reactiva se considera un paradigma de programación donde trabajamos fundamentalmente con flujos de datos y eventos de propagación del cambio. Debido a su naturaleza, la programación reactiva es un subparadigma englobado por el paradigma declarativo, ya que el desarrollador se centra más en los flujos de datos y sus propiedades, y no en el flujo de ejecución. En contraposición, encontramos el paradigma imperativo, donde el desarrollador indica los pasos a ejecutar para cambiar las estructuras de datos.

💡 El término “reactive” está directamente ligado al “Reactive Manifesto”, cuya última versión se publicó en 2014, y donde se definen las principales características de los sistemas reactivos. Si quieres profundizar más en este tema, puedes consultar la definición del manifiesto [aquí](#).

Para comprender los conceptos básicos de la programación reactiva, imagina que tenemos el siguiente código en JavaScript puro:

```
let A0 = 1
let A1 = 2
function add() {
  let sum = A0 + A1;
  A1 = 5;
  return sum;
}
console.log(add()); //Output: 3
```

En la primera línea de la función `add()` realizamos un sumatorio entre las variables `A0` y `A1`. En principio, como la función devuelve la variable `sum`, debería retornar el valor 3. Sin embargo, en la segunda línea de la función modificamos la variable `A1` a 5. **¿Cuál es el valor que devuelve la función `add()`?** En realidad, sigue siendo 3, ya que la variable `sum` se ha creado previamente al cambio de la variable `A1`. Estos son los **principios de la programación imperativa**.

Sin embargo, veamos ahora cómo actúa la programación reactiva con un ejemplo en Vue. Imaginemos que tenemos las siguientes variables en el `data()`:

```
data: () => ({
  A0: 1,
  A1: 2,
}),
```

Ahora definimos un método que realiza la misma operación que el ejemplo anterior: sumar los valores de las variables `A0` y `A1`. El hecho de utilizar `this.A0` y `this.A1` da una pista de cómo será el comportamiento reactivo de este método.

```
add() {
  let sum = this.A0 + this.A1;
  this.A1 = 5;
```

```
    return sum;
}
```


Y, finalmente, utilizamos el método `add()` en el template:

```
<template>
  <div id="app">
    {{add()}}
  </div>
</template>
```

En este caso, **cuando se muestre el valor de `add()`, veremos el valor 6, ya que se ha aplicado el cambio de la variable `A1` en la segunda línea del método `add()`**. Esto es debido a la aplicación de los **principios de la programación reactiva**.

En sistemas reactivos, como Vue, existe un **sistema interno que reacciona a los cambios del entorno**. En resumen, Vue hace que todas las variables declaradas en `data()` dispongan de un sistema de *proxies* o *getters/setters* que se encargan de escuchar el entorno para hacer cambios si es oportuno. En cambio, si definimos una variable local, ésta perderá la reactividad, ya que nunca acciona el sistema de *get/set* o *proxy* que Vue utiliza para ello. De esta manera, el código del primer ejemplo (la función `add()`) nunca será reactivo, ya que las variables que se utilizan no disponen de *proxies* o *getters/* y *setters* (de hecho, no están definidas en `data()`)

No obstante, cuando las operaciones son más complejas, Vue dispone de métodos más sofisticados para escuchar cambios en el DOM, como son las propiedades computadas (**computed properties**) y los observadores (**watchers**), como veremos en los materiales.

 Si quieres practicar y profundizar más en cómo se lleva a la práctica la programación reactiva, en [este video](#) por Jordan Joxwiak, desarrollador de Google, podrás ver un seminario muy interesante. El seminario es bastante extenso (dura aproximadamente una hora), pero repasa conceptos y muestra ejemplos que pueden ayudarte a profundizar. Su visualización es opcional y no es necesaria para la elaboración de esta PEC.

Recursos de lectura

Tal y como hicimos en la PEC anterior, a continuación os indicamos los recursos de lectura que consideramos importantes que reviséis y estudiéis para realizar esta PEC. También os indicamos las secciones (o capítulos concretos) que deberíais leer con atención para la correcta ejecución de esta PEC. Para cada una de las partes recomendadas incluimos un comentario para motivar nuestra recomendación. Por supuesto, si queréis profundizar, podéis revisar y leer las secciones no recomendadas en este listado de cada recurso.

 **Libro: Azaustre Rodríguez, C. Conceptos y técnicas avanzadas de programación web. FUOC, 2024**

1.4 . Eventos y reactividad

Exploraremos brevemente el concepto de eventos y reactividad en JavaScript y algunos de sus frameworks.

4. Mantenimiento y gestión del estado

Este capítulo explica el concepto de gestión del estado en el desarrollo web, que se refiere al manejo de los datos que determinan cómo se ve y se comporta una aplicación en cualquier momento dado. Nos centraremos únicamente en los capítulos 4.1, 4.2, 4.3, 4.4 y 4.7 por ahora.


Vue docs

Essentials. <https://vuejs.org/guide/essentials/application.html>

En esta serie de capítulos se hace un pequeño repaso a las características básicas de Vue, su estructura y los principios básicos para usar este framework. En esta PEC nos centraremos en los apartados *Computed Properties*, *Event Handling* y *Form Input Bindings*, por lo que prestaremos especial atención a ellos.

Components In-depth: <https://vuejs.org/guide/components/registration.html>

Esta serie de capítulos se centra de un modo más específico en las particularidades de los componentes en Vue respecto a otros frameworks de programación reactiva. Es importante prestar especial atención al capítulo Events, ya que el núcleo de esta PEC se basa en los principios de este capítulo.

 *Recuerda que Vue 3 cuenta con dos tipos de API: Options API y Composition API. El uso de cada API implica un estilo de programación diferente y el uso de métodos diferentes. En esta PEC haremos uso de la Options API y de su documentación.*

 **Video: Chris Dixon. [In-Depth Vue 3 for Beginners](#). O'Reilly, 2021**

Chapter 6. Forms, events and modifiers

En los videos de este capítulo se hace un resumen de los formularios, sus particularidades en Vue, y cómo obtener y publicar sus datos a lo largo de una aplicación.

Chapter 5. Countdown Project

En este capítulo se ejemplifica mediante un pequeño proyecto aspectos básicos del framework. Son especialmente útiles las menciones a propiedades computadas y *watchers*.

Formato de entrega

Debe entregarse **un archivo zip con el contenido de la carpeta src del proyecto y un archivo pdf con las respuestas a las preguntas formuladas**. El documento pdf deberá contar como mínimo con el nombre del alumno en el encabezado.

Puntuación

- La puntuación de cada pregunta está detallada en su enunciado, **sobre un total de 10 puntos**.
- Las preguntas puntúan un **máximo de 2.5 puntos**. La puntuación está detallada en cada pregunta.
- El desarrollo de los componentes puntúan un **máximo de 7.5 puntos**. En la corrección se tendrá en cuenta:
 - Creación de eventos (10%)
 - Captura de eventos (10%)
 - Creación correcta de ítems y gestión de sus propiedades (20%)
 - Gestión del borrado de ítems (10%)
 - Búsqueda y filtrado (20%)
 - Desarrollo y funcionamiento global de la aplicación (20%)
 - Calidad general del código (10%)

Consultas

En caso de que tengáis que consultar algo mediante el foro o por correo electrónico, copiad vuestro código a una plataforma online y enviad el link para evitar problemas con el correo de la UOC (ya que elimina los ficheros *.js* y *.vue* para evitar inyectar código malicioso así como la indentación del código). Os recomendamos utilizar cualquiera de estas dos opciones:

- [Codepen](#) (para sencillos snippets de código)
- [CodeSandBox](#) (para ejercicios más complejos)

Nota: En caso de publicar algún código en el foro, deberán ser consultas genéricas y no directamente soluciones a los ejercicios. Hacer accesible soluciones de ejercicios a otros compañeros, aunque pueda no tener una intención directa, se considerará copia y se penalizará académicamente a nivel de asignatura.


Propiedad intelectual y plagio

La Normativa académica de la UOC dispone que el proceso de evaluación se cimenta en el trabajo personal del estudiante y presupone la autenticidad de la autoría y la originalidad de los ejercicios realizados.

En esta PEC no está permitido el uso de herramientas de Inteligencia Artificial. En el plan de estudios y en [la web sobre integridad académica y plagio de la UOC](#) encontrarás más información sobre las faltas y las consecuencias del plagio. Recuerda que si utilizas material externo siempre tienes que hacer referencia a las fuentes, también puedes leer <https://biblioteca.uoc.edu/es/pagina/Como-citar/>

El estudiante será calificado con un suspenso (D/0) si se detecta falta de originalidad en la autoría de alguna prueba de evaluación continua (PEC) o final (PEF), sea porque haya utilizado material o dispositivos no autorizados, sea porque ha copiado textualmente de internet, o ha copiado apuntes, de PEC, de materiales, manuales o artículos (sin la cita correspondiente) o de otro estudiante, o por cualquier otra conducta irregular.

Preguntas

 Las respuestas a preguntas que requieran consultar material externo deben incluir referencias a la fuente. Puedes leer la sección “Propiedad intelectual y plagio” para obtener más detalles.

Ejercicio 1. (0.5 puntos)

Vue implementa un mecanismo de transformación de notaciones para los nombres de eventos. ¿Qué tipo de notaciones se recomienda usar para emitir eventos y para escucharlos en el componente padre? Pon un ejemplo ilustrativo.

Ejercicio 2. (0.5 puntos)

¿Para qué sirve el elemento “emits” o “defineEmits” en el ámbito de un componente? ¿Qué dos estructuras pueden utilizarse para detallar sus integrantes? Pon un ejemplo.

Ejercicio 3. (0.5 puntos)

Dado el siguiente código en un componente de Vue:

```
import { reactive } from 'vue';

const user = reactive({
  name: 'Ana',
  age: 30
});

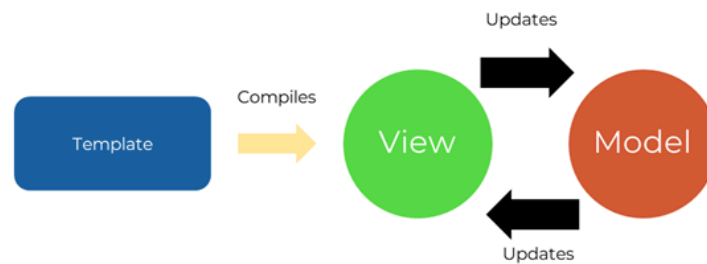
function updateAge(newAge) {
  user.age = newAge;
}

updateAge(40);
```

¿Es correcto el uso de “reactive” para este caso? ¿Qué ocurriría si se usara “ref”? ¿Cuál es la diferencia entre ambos?

Ejercicio 4. (0.5 puntos)

Explica el concepto que se muestra en el siguiente diagrama, centrándote en las flechas de color negro.



Ejercicio 5. (0.5 puntos)

- A. ¿Qué es una propiedad computada? Pon dos ejemplos.
- B. ¿Qué diferencia a una propiedad computada de una propiedad "watched"? Ilústralo con un ejemplo.

Parte práctica

En esta PEC abordaremos el estado y la gestión de eventos de los componentes creados en la anterior PEC. A modo de resumen, la funcionalidad que implementaremos será la siguiente: **Los datos de nuestra aplicación residirán en el componente App**, por lo que el formulario debe emitir un evento con la información del ítem creado para que sea capturado por el componente App y se pueda actualizar la lista de elementos, en lugar de usar valores de ejemplo.

Borrado de elementos

Al hacer clic en el botón de eliminar de cada tarjeta de serie, debe emitirse un evento desde el componente ShowCard hacia la raíz de donde están los datos, lo cual permita realizar el borrado efectivo del ítem en cuestión. Es importante incluir en el evento la información más eficiente y relevante posible.

Creación de elementos

Ahora, el componente CardForm se encargará de recoger los datos que rellene el usuario para transformarlo en un ítem de tipo “show”, para añadirlo al listado de series.

Debe tenerse en cuenta lo siguiente al darle al botón de crear:

- Comprobar que los campos *título*, *descripción*, *URL de la imagen*, *tags* y *fecha* no están vacíos. Si alguno de ellos está vacío deberá mostrar un mensaje de error y no emitirá el ítem.
- Separar el contenido del campo de *tags* por comas, y agregarlos a un Array con tantos elementos como campos se extraigan.
- Crear un objeto con la información almacenada en el formulario y en el formato correcto esperado para cada propiedad.
- Emitir el evento que permita añadir el nuevo ítem al listado.
- Borrar los campos del formulario.

Sobre el ID del elemento, éste debe ser aleatorio para evitar duplicados. Para hacer esto, se recomienda utilizar el método `randomUUID()` de la interfaz [Crypto](#), disponible en el objeto “window”.

Visibilidad del formulario

A partir de ahora, el formulario para añadir elementos CardForm estará oculto por defecto. El botón de “Add new show” situado en el componente FilterBar hará que se muestre.

Una vez que CardForm esté visible, el botón “Add new show” debería estar oculto, ya que no tendría sentido mostrarlo.

El botón situado arriba a la derecha del formulario debería ocultarlo, y volver al estado original.

Búsqueda y filtrado de datos

Después de haber creado el esqueleto en la PEC anterior, ahora vamos a hacer que funcione el sistema de búsqueda y filtrado de datos.

Debe tenerse en cuenta lo siguiente:

- El componente FilterBar se encargará de recoger los inputs del usuario, pero será el componente App el que realice toda la lógica de negocio. Asimismo, App traspasará a CardBoard los elementos ya filtrados. Valora si es necesario crear métodos, computed o watchers para optimizar el código, así como el uso de props, emits o v-model para la comunicación entre componentes.
- El filtrado debe realizarse inmediatamente cada vez que el usuario realice una acción, ya sea escribir en el campo de búsqueda o modificar los parámetros de filtrado.
- El campo de búsqueda deberá tener en cuenta el nombre, la descripción y las etiquetas a la hora de filtrar los datos.
- El selector de etiquetas deberá recoger y mostrar todas las etiquetas del array de series de, sin duplicar.
- El selector de ordenación deberá ser por nombre, fecha de lanzamiento o valoración.
- El botón de “Clear” deberá reiniciar todos los filtros y devolver el listado original de series.