Universidad de Jaén

Tema 5 Frameworks MVC en el cliente: VueJS

Desarrollo de Aplicaciones Web

José Ramón Balsas Almagro
Departamento de Informática
Universidad de Jaén
jrbalsas@ujaen.es

v 0.3

Objetivos

- Conocer algunos ***Frameworks*** **MVC de programación en el cliente**
- Conocer las **características generales** de *Frameworks* para aplicaciones web centradas en el cliente
- Comprender el concepto de **arquitectura basada en componentes** en una aplicación **VueJS**
- Saber manejar las **funcionalidades básicas** de VueJS
- Saber construir una **aplicación CRUD** usando VueJS

Frameworks Javascript MVC en cliente

- Principalmente definen patrones de diseño en la construcción de aplicaciones web aplicando conceptos de Ingeniería del Software
- Disponen de o permiten usar bibliotecas de componentes enriquecidos
- **Algunos ejemplos:**

Tipo	Descripción
BackboneJS	2010. Framework de desarrollo MVC para webapps. MIT license.
~~AngularJS~~ Angular	2009. Framework MVC para desarrollo de webapps. Mantenido por Google. MIT License. Rediseñado para trabajar con Typescript
ReactJS	2013. Biblioteca de renderizado eficiente y componentes RIA (virtual DOM). Facebook. BSD License.
EmberJS	2011. Framework MVW para desarrollo de webapps. Utiliza MVC. MIT License.
Knockout	2010. Framework MVVM ligero. MIT License.
VueJS	2014. MVVM. MIT License.
SvelteJS	2016. MVVM. MIT License. Genera código JS para ejecutar en cliente a partir de html, js y css
MeteorJS	2012. End-to-End mobile/web applications in Javascript. MIT License.

Elementos en Frameworks MVC

- Varían según el *framework*
  - Modulares o reutilizando bibliotecas de terceros para elementos específicos
- **Algunos elementos habituales**
  - **Modelo**
    - Enlazado dinámico vista-modelo (data binding): Cambios automáticos en modelo/vista
    - Inyección de dependencias
  - **Vista**
    - Plantillas/manipulación DOM. Aplicaciones en una página, *SPA*
    - Componentes: definición y bibliotecas
    - Validación en cliente
  - **Controlador,** soporte para mejoras ES6/7 o meta-lenguajes e.g. Typescript
  - **Módulos auxiliares**
    - **Enrutador**, ejecución en cliente según URL
    - **Gestión de estado**
    - **Gestión de conexiones asíncronas**
    - **Generación de contenidos en servidor (SSR)**
  - **Utilidades de desarrollo**
    - **TDD,** *integración con bibliotecas de prueba contínua*
    - Generadores de código, analizadores de código (*Linters*), empaquetadores *(Bundlers*), …

VueJS

https://vuejs.org

- **Características**
  - Biblioteca para construir interfaces de usuario. Evan You, 2014 . MIT License.
  - Reutiliza conceptos exitosos de Angular y React
  - Basada en uso de HTML, CSS y Javascript
  - Renderizado declarativo basado en componentes
    - Uso de HTML como motor de plantillas
    - Estado basado en Javascript
    - Actualizaciones reactivas de vistas en respuesta a modificaciones del estado
- **Usos**
  - Simplificación de programación en cliente, e.g. modificación del dom, gestión eventos…
  - Organización de vistas mediante componentes, e.g. layout
  - *Simple-Page Applications* (SPA)
  - Generación de contenidos en servidor (SSR), e.g. Nuxt
  - Generación de contenidos estáticos (SSG), e.g. VitePress, Nuxt
  - Aplicaciones híbridas en escritorio, e.g. Electron, o móviles, e.g. Ionic Vue, Quasar

Ecosistema Vue

- Proyectos que complementan la biblioteca básica (core) dotándola de características de framework completo:
- **Herramientas de desarrollo:**
  - Generadores de código: Vue CLI, Vite
  - Extensiones para IDEs, e.g. VSCode, IntelliJ, Eclipse, …
  - Plugin en navegadores: *Vue Devtools* https://devtools.vuejs.org/
  - Soporte para otros metalenguajes: Typescript, JSX, SCSS, …
  - *Vue loader* para soporte de componentes *.vue (*Single-File Components, SFC*)
  - Plugin *ESLint* para análisis estático de estilo y errores
  - Utilidades TDD (*Test-Drive Development*)
  - Vue PlayGround para pruebas on-line https://sfc.vuejs.org/
- **Vue Router** para desarrollo de SPA
- **Pinia, Vuex** para gestión centralizada de estado de la aplicación
- **VuePress** para generación de contenidos en servidor
- …

Vistas

- Definición básica usando **HTML5**
  - Enfoque declarativo frente a creación dinámica con javascript
- Elementos adicionales a modo de plantilla.
  - Puede utilizar fragmentos html5 <template id="nombre"> Ejemplo interactivo
- Vue procesa los elementos adicionales del documento para generar el DOM de la vista
- **Elementos de plantilla:**
  - **Directivas,** elementos sintácticos, normalmente atributos. Actualizan dinámicamente el DOM frente a cambios en la expresión
    - e.g <p **v-if**="expresión"></p>
  - **Interpolaciones**: *{{expresión js}}*, se sustituyen por el resultado de la expresión o el valor de una variable del estado del componente e.g. {{cliente.nombre}}
  - **Componentes**, permiten crear nuevas etiquetas html y sus funcionalidades asociadas, e.g. < RelojComponent />

Inicializando una aplicación Vue

- **Pasos**
  - Crear el componente raíz
  - Asociarlo a un elemento del DOM
- La carga de la biblioteca se puede hacer de forma clásica (objeto global) o mediante módulos ES6 (elementos importados)
  - Existen versiones para producción (prod) y desarrollo (con comprobaciones adicionales)

```
<script  src="https://unpkg.com/vue@3">
</script>
<script defer>
 const App = {
   template: `<h1>Hello World</h1>`
 }
 //Interacción mediante Objeto global
 Vue.createApp(App).mount("#app")
</script>

<div id="#app"></div>
```

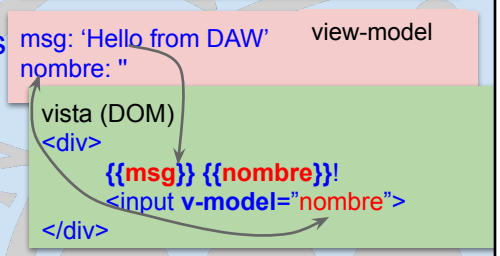
```
<script type="module">
 //Interacción mediante API de funciones
import {createApp} from
    "https://unpkg.com/vue@3/dist/vue.esm-browser.js"
 const App = {
     template: `<h1>Hello World</h1>` }
 createApp(App).mount("#app")
</script>

<div id="#app"></div>
```

Modelo de Vista

- El modelo de vista (estado) de un componente se define mediante variables vinculadas (*binding*) a su representación en el DOM
- Se declaran en un *objeto devuelto* por el método **data()**
- Accesibles desde la vista por su nombre o desde los métodos del componente usando this
- Son **variables reactivas:**
  - Implementan un ***patrón observador***
  - Cambios en su valor son notificados a la vista (*binding*)
  - Cambios en la vista son reflejados en *las variables (double binding)*. e.g. <input>

```
const App = {
 data() {
   return {
     msg: "Hello from DAW",
     name: ""
   }
 },
 template: "..."
}
```



- Ejemplo básico aplicación JS Vue

Algunas directivas predefinidas

- **v-text=”**expresión**”**
  - Asocia el texto de una etiqueta al resultado de una expresión *(single binding)*.

    <p **v-text=**”nombre”> o {{nombre}}
- **v-bind:atributo="expresión"**
  - Asocia un valor a una propiedad del elemento <a v-bind:href="var_url">enlace
  - Puede simplificarse mediante **":"** e.g. <a :href="var_url">enlace
- **v-model=**”modelVar”
  - Asocia (*double binding*) un control de formulario a una propiedad del modelo
- **v-if=**"condición" , **v-show=**"condición"
  - incluye/muestra un elemento si una condición es cierta
  - Puede ir acompañada de otra etiqueta con directiva **v-else** o **v-else-if**
- **v-for=**”elem **in** array”
  - crea una copia de una plantilla por cada elemento de un array
  - Permite iterar en un rango de valores e.g. " i in 100"
  - Si la colección puede mutar es conveniente asignar un identificador único para cada ítem con v-bind:key="valor único"

Ejemplo de modificación de vista

```
const App = {
 data() {
   return {
     clientes: [
       {id: 1, nombre: 'Pepe Alcántara', dni: '11111111-A', socio: false},
       {id: 2, nombre: 'Maria López', dni: '33333333-C', socio: true},
       {id: 3, nombre: 'Carlos Sánchez', dni: '22222222-B', socio: false},
     ],
   }
 },
 template: "#listado"
}
```

Ejemplo:
https://jsfiddle.net/jrbalsas/hfvsw1jn/

```
<template id="listado">
 <table>
 <tr>
   <td>Nº</td><td>Nombre</td><td>DNI</td><td>Socio</td>
 </tr>
 <tr v-for="c in clientes" v-bind:key="c.id">
   <td v-text="c.id"></td>
   <td>{{c.nombre}}</td><td>{{c.dni}}</td><td>{{c.socio}}</td>
 </tr>
</table>
<p>Total clientes: {{clientes.length}}</p>
</template>
```

Nº	Nombre	DNI	Socio
0	CARLOS	12345678T	true
1	MARÍA	87654321G	true
2	JUAN	12121212H	false

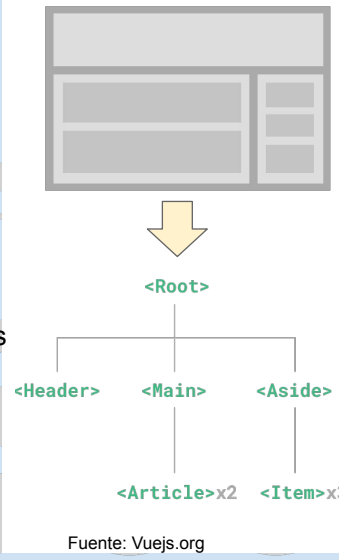
Total clientes: 3

Arquitectura basada en componentes

- En Vue, una aplicación se organiza en una jerarquía de objetos JS denominados **componentes**.
  - Disponen de una **vista** como un conjunto de elementos del DOM procesados como plantilla *(template) y estilos CSS asociados*
  - Almacenan su estado, **modelo de vista**, mediante *variables reactivas* y propiedades proporcionadas en su declaración
  - Disponen de métodos predefinidos y personalizados para gestionar su lógica de **control**
  - Utilizan **servicios** para acceder a funcionalidades no relacionadas con las vistas, que obtienen mediante *inyección de dependencias*
- **Ejemplo de uso:**

<ClubFooter text="Club de Tenis" /> o <club-footer text="Club de Tenis"/>

Importante: en documentos html utilizar kebab-case: club-footer. Dentro de plantillas de componentes se recomienda usar Pascal-Case: ClubFooter

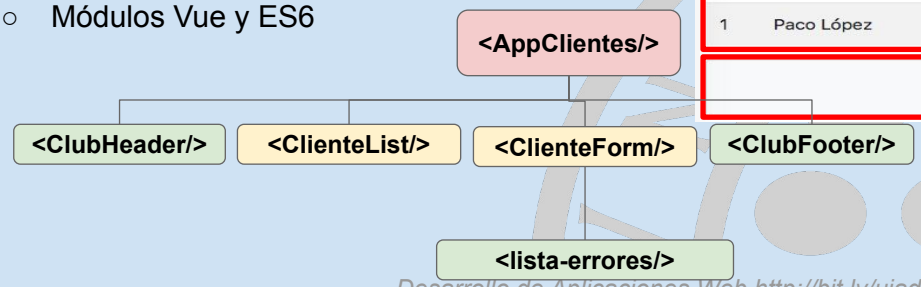


Fuente: Vuejs.org

Aplicación VueJS de ejemplo

Código Ejemplo: https://github.com/jrbalsas/dawClubVue

- **Servidor**
  - Servicio web REST basado en JAX-RS
- **Cliente**
  - SPA con único fichero index.html
  - Arquitectura basada en componentes
  - Implementación DAO en cliente basada en servicios fetch o array (pruebas)
  - Módulos Vue y ES6



Definición de componentes

Métodos de definición

- Objeto literal JS, normalmente en su propio módulos ES6 (.js). Se pueden procesar directamente
- *Single-File Component (SFC)*, módulo específico Vue (.vue). Requieren preprocesamiento previo

Modalidades de configuración (API Styles)

- **Options API,** versión tradicional v2 ← Utilizaremos este enfoque
  - Dispone de propiedades y métodos preestablecidos que definen el estado y comportamiento del componente, e.g. *data*, *methods*, *mounted*, props, etc.
  - Enfoque más orientado a objetos
  - Uso en aplicaciones más simples o para mejoras en cliente
  - El estado es reactivo por defecto
- **Compositions API,** versión v3+ (febrero, 2022)
  - Nueva versión. En proceso de adaptación: algunas bibliotecas de terceros no adaptadas
  - Útil con aplicaciones más complejas.
  - Permite construir componentes a partir de funciones con estado (*Composables*) que pueden reutilizarse en diferentes componentes
  - El programador debe definir expresamente la reactividad de las variables de estado

Ejemplo de Componente JS

Universidad de Jaén

```
const ClubFooter =  {
 props: [
   'text' //Component attribute (single binding)
 ],
 mounted() {
   console.log("Cargado componente: ClubFooter.js")
 },
 template: `
 <footer class="card text-end well bg-light">
   <div class="card-body">
     <p class="card-text">
     <span v-if="!text">Made with &#9829;in
DAW</span>
       {{ text }}
     </p>
   </div>
 </footer>
 `
}
```

Ejemplo: https://jsfiddle.net/jrbalsas/8m95bd0s

Debe definirse el ámbito de utilización:

- **Global:** accesibles desde cualquier componente:

```
app.component( 'ClubFooter', ClubFooter )
```

- **Local:** solo en los componentes que lo solicitan

```
components: {
  'ClubFooter': ClubFooter.
}
```

```
<template>
<!-- in html template use kebab-case naming -->
 <club-footer text="Hello from DAW"/>
</template>
```


Single-File Components, SFC

Universidad de Jaén

- Ficheros con extensión .vue
- Formato específico para definir la vista (template), lógica de control y estilo de un componente
- Requieren un pre-procesamiento específico (transpilación) a módulo JS. Utilizar herramientas como Vite o Vue CLI
- **Ventajas**
  - Sintaxis familiar
  - Cohesión
  - Simplicidad de uso frente a enfoque programático
  - Integración con IDE, e.g. auto-completado, comprobación de tipos…
  - Pueden integrarse como módulos ES6, i.e. utilizando import

```
//Componente.vue

<template>
   <!--Plantilla del componente-->
</template>
<script>
export default {
 // Definición del componente
}
</script>
<style>
   /* clases de estilo del componente*/
</style>
```

A partir de ahora los ejemplos los realizaremos con componentes SFC, aunque serían fácilmente convertibles a componentes JS

Ejemplo de Componente SFC

Universidad de Jaén

```
//ClubFooter.vue
<template>
 <footer class="card text-right well bg-light">
   <div class="card-body">
     <p class="card-text">
       <span v-if="!text">
           Made with &#9829; in DAW
       </span>
       {{ text }}
     </p>
   </div>
 </footer>
</template>

<script>
export default {
 props: [
   'text' //Component attribute (single binding)
 ],
 mounted() {
   console.log("Cargado componente ClubFooter.vue")
 },
}
</script>
```

Consideraciones de estilo

- En templates de otros componentes se utilizan con notación *PascalCase* ya que serán pre-procesados
- El módulo se nombra como la etiqueta del componente

```
<template>
<!-- in component templates use PascalCase -->
 <ClubFooter text="Hello from DAW"/>
</template>
```


Integración de componentes SFC

Universidad de Jaén

```
//AppClientes.vue
<template>
 <div>
   <ClubHeader text="Club de tenis"/>
         <ClientesList :datos="clientes"/>
   <ClubFooter/>
 </div>
</template>

<script>
//import ClubFooter from "./components/commons/ClubFooter" //JS
Component
import ClubFooter from "./components/commons/ClubFooter.vue"
import ClubHeader from "./components/commons/ClubHeader.vue";
import ClientesList from "./components/clientes/ClienteList.vue";

export default {

 components: { ClubHeader,ClubFooter, ClientesList },
 data() {
   return {
     clientes: [ ],
   }
 },
}
</script>
```

```
//ClubFooter.vue
<template>
 <footer class="card text-right well bg-light">
   <div class="card-body">
     <p class="card-text">
       <span v-if="!text">
           Made with &#9829; in DAW
       </span>
       {{ text }}
     </p>
   </div>
 </footer>
</template>

<script>
export default {
 props: [
   'text' //Component attribute (single binding)
 ],
 mounted() {
   console.log("Cargado componente ClubFooter.vue")
 },
}
</script>
```

Ejemplo interactivo

Estructura de componentes

- Su **estado** (*propiedades, atributos, propiedades calculadas*) y **comportamiento** (*métodos*) de la **lógica de control** se definen en el objeto configuración del componente.
- Su **vista** (*template*) puede acceder directamente a las variables de estado (modelo de vista) y métodos
- **Variables de estado del componente**
  - **Atributos:** definidos en objeto devuelto por función *data()*
  - **Propiedades**: información proporcionada al componente como atributos html. Definidos en propiedad *props*. **Ejemplo:**. <Componente propiedad1="valor" />
  - **Propiedades calculadas**: definidas como funciones en opción *computed*. Recalculan su valor automáticamente cuando las variables utilizadas cambian
- **Métodos del componente**
  - **Métodos propios:** funciones definidas en propiedad *methods*. Normalmente utilizadas como manejadores de eventos asociados a la vista
  - **Métodos predeterminados (*hooks*)**: se ejecutan en fases específicas del ciclo de vida del componente, e.g. *created(), mounted(), unmounted()*, etc.

Ejemplo de propiedades

```
//ListaErrores.vue
<template>
 <div v-if="Object.keys(msgs).length > 0">
   <ul>
     <li class="text-danger"
         v-for="(message,property, index) in msgs"
         :key="property">
       {{property}}: {{message}}
     </li>
   </ul>
 </div>
</template>

<script>
/**Visualiza un listado de errores de validación
  @notes use &lt;erroresValidacion :msgs="array_errores"/&gt;
*/
export default {
 props: [
   'msgs'  //Component attribute (single binding)
 ],
}
</script>
```

4

3

Componente para visualizar errores de bean validation, e.g.
[{"message":"La longitud debe estar entre 4 y 25 caracteres","name":"nombre"},{...}]

```
//ClientesForm.vue
<template>
       ...
       <ListaErrores :msgs="errorMsgs"/>
</template>
<script>
import ListaErrores from "...";

export default {
 ...
 components: {
   ListaErrores
 },
 data() {
   return {
     errorMsgs: { }
   }
 },
}
</script>
```

2

1

Variables calculadas y Observadores

- **Variables calculadas** (*computed*) +info
  - Se definen mediante una función asociada en la opción ***computed*** del componente que utiliza alguna/s de las variables reactivas de componente
  - **Objetivo:** evitar uso de expresiones complejas en vistas
  - Su **valor** se evalúa al resultado de la función asociada:
    - El valor se reutiliza (cacheado) si no ha cambio en las variables reactivas que utiliza
    - Solo se recalcula si cambia alguna de sus variables reactivas
- **Observadores** (*watcher*) +info
  - Funciones que se ejecutan cuando se modifica una propiedad reactiva
  - Definidas en la opción **watcher** del componente
  - A diferencia de las variables calculadas usadas para visualizar información, su uso es solo para realizar acciones asíncronas, e.g. peticiones ajax, …

```
props: ['clientes'],
data() {
 return {
   filtro: "",
 }
},

computed: {
 clientesFiltrados() {
   return this.clientes.filter(c => {
     return c.nombre.includes(this.filtro)
   })
 }
},
watch: {
 filtro(newFiltro, oldFiltro) {
   console.log(`Filtrando por:${newFiltro}
               ( anterior: ${oldFiltro})`)
 }
},
```


Ejemplo de variables calculadas

```
//ListaClientes.vue
<template>
   <h1>Listado de Clientes</h1>
   <p>
     <input v-model="filtro"                    2
            placeholder="filtrado por nombre">
   </p>
   <table>
   <tr>
     <td>Nº</td><td>Nombre</td>
    <td>DNI</td><td>Socio</td>
   </tr>
   <tr v-for="c in clientesFiltrados"           4
       :key="c.id"
       v-on:click="muestra(c)"
       title="Ver detalles">
     <td>{{c.id}}</td>
     <td>{{c.nombre}}</td>
     <td>{{c.dni}}</td>
     <td>{{c.socio}}</td>
   </tr>
 </table>
 <p>Total clientes: {clientesFiltrados.length}}</p>
</template>
```

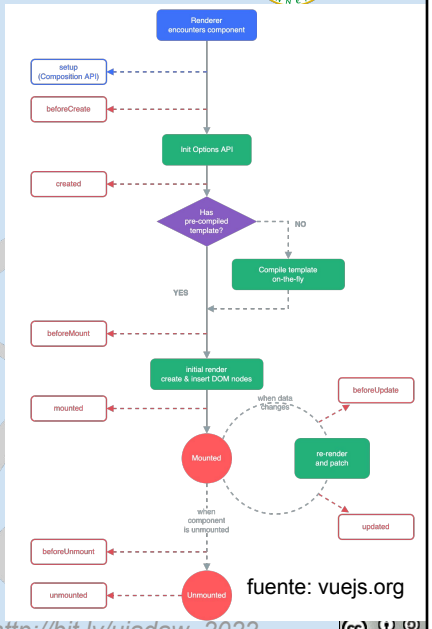
Uso: <ListaClientes :clientes="clientes"/> 1

Ejemplo interactivo

```
//... continue
<script>
export default {
 props: ['clientes'],   //component properties
 data() {               //reactive variable
   return {
     filtro: "",
   }
 },
 computed: {      // computed variable
3  clientesFiltrados() {
     console.log(`filtrando por ${this.filtro}`)
     return this.clientes.filter( c => {
             return c.nombre.includes(this.filtro)
             })
   }
 },
 watch: {               // observadores
3  filtro(newFiltro, oldFiltro) {
     console.log(`Filtrando por ${newFiltro}`)
   }
 },
 mounted() {
   console.log("Componente ListaClientes.vue visualizado")
 }
}
</script>
```

Ciclo de vida de un componente

- Vue determina los siguientes **estados de un componente:**
  - **Creado**: se ha detectado el componente y se ha configurado para comenzar a utilizarse, e.g. inicialización de propiedades, etc.
  - **Montado**: se ha procesado su apariencia visual (template) y se ha añadido al DOM (visualizado).
  - **Actualizado**: cuando se producen cambios en el estado se actualiza su apariencia visual
  - **Desmontado**: cuando el componente deja de utilizarse en el DOM
- Los componentes pueden definir manejadores, *Hooks*, que se ejecuten antes o después de producirse un cambio en alguno de sus estados: e.g. created(), mounted(), updated(), unmounted(), beforeCreated(),...



fuente: vuejs.org

Gestión de eventos

- Los **manejadores** se asocian con la directiva **v-on** (o acortador **@)** seguida del evento a capturar

```
<button v-on:click="manejador">Opción 1</button>
```

- Puede ser código JS (simple) en línea o un método del componente
- Por defecto los métodos reciben un **parámetro con el evento JS** generado
- Se les pueden proporcionar parámetros **parámetros personalizados** (si es necesario el evento nativo se puede pasar como $event)

```
<button @click="manejador('valor', $event)">Opción 2</button>
```

- **Modificadores de eventos:**
  - Permiten realizar operaciones habituales: *stop, prevent, …*

```
<form @submit.prevent="validarDatos" >    // ejecuta event.preventDefault() en método
```

  - Los eventos de teclado o ratón permiten especificar la tecla que los activa, e.g. teclado: *enter, tab, space, esc, ctrl, alt, up*… ratón: *left, right, middle*

```
<input @keyup.enter="buscaParcial" >
```

Más información sobre eventos

Ejemplo de manejo de eventos

- Ordenación de tabla por columnas

```
<!-- ClienteList.vue. →
<template>
 <h2>Listado de cliente </h2>

 <table class="col table table-striped table-hover">
   <thead>
   <tr title="Seleccionar encabezado para ordenar">
     <th v-on:click="orderBy('id')">ID
         <span v-if="order==='id'">&darr;</span></th>
     <th @click="orderBy('nombre')">Nombre
         <span v-if="order==='nombre'">&darr;</span></th>
     <th @click="order='dni'">DNI
         <span v-if="order==='dni'">&darr;</span></th>
     <th @click="order='socio'">Socio
         <span v-if="order==='socio'">&darr;</span></th>
   </tr>
   </thead>
   <tr class="c-pointer"
       v-for="c in clientesOrdenados" :key='c.id'>
     <td>{{ c.id }}</td><td>{{ c.nombre }}</td>
     <td>{{ c.dni }}</td><td>{{ c.socio }}</td>
   </tr>
 </table>
</template>
```

1

3

```
<script>
export default {
 props: [
   'datos', //Component attribute (single binding)
 ],
 data() {
   return {
     order: 'id'
   }
 },
 computed: {
   clientesOrdenados() {
     console.log("ordenando por " + this.order)
     //return sorted copy
     return this.datos.slice().sort((c1, c2) => {
       let result = 0;
       if (c1[this.order] > c2[this.order]) result = 1;
       if (c1[this.order] < c2[this.order]) result = -1;
       return result;
     });
   }
 },
 methods: {
   orderBy(newOrder) {
     this.order = newOrder;
   }
 },
}
</script>
```

2

Gestión de formularios

- Vue permite vincular controles de formularios con variables reactivas del componente

  <input **v-model**="variable_reactiva">
- El enlazado es en ambos sentidos (***double binding***): control←→variable
- Es equivalente a un enlazado simple y el evento de actualización correspondiente:

  <input v-bind:value="variable_reactiva"
  v-on:input="event => variable_reactiva = event.target.value">
- El valor inicial depende del atributo JS vinculado, NO de los atributos de las etiquetas (i.e. value)
- La directiva v-model permite **modificadores** (pueden anidarse):
  - .lazy cambia el evento input por defecto por uno de tipo change, e.g. al cambiar a otro control en vez de al pulsar una tecla
  - .number convierte el la cadena introducida a un valor numérico
  - .trim elimina espacios a ambos lados delt exto

Ejemplo básico de formulario

Universidad de Jaén

Ejemplo interactivo

```
<script >
import ListaClientes from "./ListaClientes.vue"

export default {
 components: { ListaClientes },
 data() {
   return {
     lastClienteId: 3,
     clientes: [
       { id: 1, nombre: 'Pepe Alcántara', dni: '11111111-A', socio: false },
       { id: 2, nombre: 'Maria López', dni: '33333333-C', socio: true },
       { id: 3, nombre: 'Carlos Sánchez', dni: '22222222-B', socio: false },
     ],
     cliente: {}
   }
 },
 methods: {
   altaCliente() {
     this.cliente.id = ++this.lastClienteId;
     this.clientes.push(this.cliente);
     this.cliente = {}
   }
 }
}
</script>
```

3

```
<template>
 <ListaClientes :clientes="clientes" />
 <h2>Añadir cliente</h2>
 <form @submit.prevent='altaCliente'>
   <p><input placeholder="nombre" v-model='cliente.nombre'></p>
   <p><input placeholder="dni" v-model='cliente.dni'> </p>
   <p>Socio<input type="checkbox" v-model='cliente.socio'></p>
   <p><input type="submit" value="Alta"></p>
 </form>
</template>
```

2

1

Validación en cliente

Universidad de Jaén

- No existe un mecanismo específico para definir restricciones en controles de formularios
- Existen bibliotecas especializadas de terceros, e.g. vee-validate, vuelidate…
- No obstante, es posible sistematizar el proceso mediante validaciones manuales y gestión dinámica de clases de estilo de los controles

Ejemplo

```
<template>
 <ListaClientes :clientes="clientes"/>
 <h2>Añadir cliente</h2>
 <form @submit.prevent='altaCliente'>
   <p><input placeholder="nombre"
             v-model='cliente.nombre'></p>
   <p><input placeholder="dni" v-model='cliente.dni'>
     <div v-if="errorMsgs.dni" >{{errorMsgs.dni}} </div></p>
   <p>Socio
       <input type="checkbox" v-model='cliente.socio'></p>
   <p><input type="submit" value="Alta"></p>
 </form>
</template>
```

1

3

Ejemplo interactivo

```
<script >
export default {
 data() {
   return {  //...
     errorMsgs: {}, //form validation errors
     cliente: { nombre:"", dni:"", socio: false}
   }
 },
 methods: {
  altaCliente() {
    if (this.validaCliente()) {
         this.cliente.id=++this.lastClienteId;
         this.clientes.push(this.cliente);
         this.cliente={ nombre:"", dni:"", socio: false};
    }
  },
  validaCliente() {
    this.errorMsgs={} //clean previous errors
    let valido=true;
    if ( /^\d{7,8}-?[A-Z]$/i.test(this.cliente.dni)===false ) {
      this.errorMsgs.dni = "No es un nif válido";
      valido=false;
    }
    // other validations
    return valido;
  }
 }
}
</script>
```

2

Propuesta de solución: El objeto *errorMsgs* contendrá una propiedad por cada control input donde se detecte el error. El contenido será la descripción del error

Desarrollo de Aplicaciones Web http://bit.ly/ujadaw. 2022

CC BY SA

Reglas de estilo en componentes

- **Reglas de estilo**
  - Los componentes pueden definir reglas de estilo a nivel global **<style>** o local **<style scoped>**
  - Los estilos locales (*scoped)* no pueden ser modificados por otros estilos globales con el mismo nombre (son únicos)
- **Propiedades dinámicas de estilo**
  - Vue facilita la asignación dinámica de clases de estilo (atributo *class*) o reglas de estilo (atributo *style*) a etiquetas html en componentes
- **Sintaxis:**
  - **:class**="{claseCSS1: expresion1, claseCSS1: expresión2}" //equivalente a v-bind:class
  - **:style**={propCSS1: expresion1, propCSS1: expresión2}"
- Las clases de estilo o propiedades con guiones deben ir entre comillas, e.g. 'font-size'
- **Ejemplos**:

```
<input :class="{'text-danger': errores.nombre }" />
<body :style="{ color: preferencias['color'] }">
```

+información

```
<template>
   <input placeholder="dni" v-model='cliente.dni'
          :class="{error: errorMsgs.dni}">
     <div v-if="errorMsgs.dni">{{errorMsgs.dni}}</div>
</template>
<style scoped>
 .error { color: red;  } /*has precedence*/
</style>
<style>
 .error { color: orange; }
</style>
```

Ejemplo interactivo

Inyección de dependencias

- **Motivación:**

  ¿Cómo proporcionar elementos a componentes en cualquier nivel de la jerarquía?

  - **Opción 1** ❌**:** Enlazado de propiedades (binding)
    - Problema: Solo es válido para pasar datos a un hijo directo pero se complica para profundidades superiores (*prop drilling*)
  - **Opción 2** ✅**:** Un componente pueden proporcionar (provide) elementos a cualquier hijo de sus jerarquía mediante inyección (*inject*)
- **Uso:**
  - Se definen las variables compartidas en el estado de los componentes
  - Las variables inyectadas pueden ser accedidas como locales en la vista o en el código del componente mediante this
  - Componente proveedor: **provide**: { variable1: dato1, variable2: dato2 }
  - Componente cliente: **inject**: ['variable1', 'variable2']

Ejemplo de componente formulario


```
<script> //AppClientes.vue
import {ClientesDAOfetch, ClientesDAOList} from
"./services/clientes.services.js";

//const clientesDAO = new ClientesDAOList();
const clientesDAO =  new
            ClientesDAOfetch('http:/.../api/clientes');
export default {
 provide: { clientesDAO }
}
</script>
```

1

```
<template>  //ClienteForm.vue
 <h2 v-if="cliente.id > 0">Edición de cliente </h2>
 <h2 v-if="cliente.id === 0">Alta de cliente </h2>
 <ListaErrores :msgs="errorMsgs" />

 <form novalidate name='frmClientes' class="well">
   <h3>ID: <span>{{ cliente.id }}</span></h3>

   <input v-model="cliente.nombre" />
   <input v-model="cliente.dni" />
   <input type="checkbox" v-model="cliente.socio" />
   <button @click.prevent="guardarCliente">
            Guardar</button>
 </form>
</template>
```

```
<script>
export default {
 props: ['cliente'],
 inject: ['clientesDAO'],
 data() {
   return {
     errorMsgs: {}
   }
 },
 methods: {
   async guardarCliente() {
     if (this.validarCliente()) {
       console.log("actualizando cliente")
       try {
         if (this.cliente.id === 0) {//create client
           await this.clientesDAO.crea(this.cliente)
         } else { //update client
           await this.clientesDAO.guarda(this.cliente)
         }
       } catch (err) {  s//server validation error
         err.forEach(error => {
           this.errorMsgs[error.name] = error.message;
         })
       }
     }
   },
 },
}
</script>
```

2

El componente gestiona altas y modificaciones en DAO

3

Gestión de eventos entre componentes

- **Motivación:** ¿Cómo proporciona información (datos/notificaciones) un componente hijo a su padre?
  - **¿Opción 1? ✗:** Si sus propiedades (*binding*) son de tipo objeto pasan por variable y se pueden modificar para ambos… pero esto supone un problema de acoplamiento importante
- **Solución:**
  - Generar **eventos** que sean manejados por el padre e incluso se les proporcione información adicional:
  - **Componente emisor** (hijo):
    i. Declara los eventos que emite mediante **emits:** ['evento1', 'evento2']
    ii. Recibe los manejadores mediante directivas v-on:evento1="manejador"
    iii. Genera los eventos: **this.$emit**('evento1', valor1, valor2)
  - **Componente receptor** (padre)
    i. Proporciona los manejadores <ComponenteHijo @evento1="manejador">

Ejemplo de eventos entre componentes (I)

Universidad de Jaén

```
 /* AppClientes.vue */
<template> <!- AppClientes.vue -->
 <ClientesList  v-if="cliente.id === undefined"
                :datos="clientes"
                @select-cliente="visualizaCliente" />
 <ClientesForm v-if="cliente.id !== undefined"
                :cliente="cliente"
                @update-cliente="cargaClientes"
                @delete-cliente="borraCliente" />
</template>
<script>
const clientesDAO = new ClientesDAOfetch(/*...*/);

export default {
 /*... */
 data() {
   return {
     clientes: [],
     cliente: {},
   }
 },
/*...*/
```

2

5

Acción: seleccionar un cliente del listado para su edición en el formulario

```
 methods: { /*...continua*/
   cargaClientes() {
     this.clientes = await clientesDAO.buscaTodos()
     this.cliente = {}
   },
   async visualizaCliente(clienteId) {
     console.log(`visualizando cliente ${clienteId}`)
     this.cliente = await clientesDAO.busca(clienteId)
   },
 }
}
</script>
```

4

```
<template> /*ClientesList.vue*/
 <h2>Listado de cliente </h2>
 <table >
   <tr class="c-pointer" v-for="c in datos" :key='c.id'
       @click="$emit('selectCliente',c.id)">
       </tr>
   </table>
</template>

<script> //ClientesLista.vue
export default {
 props: ['datos'],
 emits: ['selectCliente'] }
</script>
```

3

1

Ejemplo de eventos entre componentes (II)

Universidad de Jaén

```
 /* AppClientes.vue */
<template> <!- AppClientes.vue -->
 <ClientesList  v-if="cliente.id === undefined"
               :datos="clientes"
               @select-cliente="visualizaCliente" />
 <ClientesForm v-if="cliente.id !== undefined"
                :cliente="cliente"
                @update-cliente="cargaClientes"
                @delete-cliente="borraCliente" />
</template>
<script>
const clientesDAO = new ClientesDAOfetch(/*...*/);

export default {
 /*... */
 data() {
   return {
     clientes: [],
     cliente: {},
   }
 },
 methods: {
  cargaClientes() {...},
  async borraCliente(clienteId) {
   if (clienteId > 0) {
      await clientesDAO.borra(clienteId);
   this.cargaClientes();
  }   /*... */
```

2

Acción: Alta, edición o borrado del cliente seleccionado

4.1

4.2

```
<template> //ClientesForm.vue
 <form novalidate >   <!-- ... -->
   <button @click.prevent="guardarCliente">
           Guardar</button>
   <button v-if='cliente.id>0'
           @click.prevent="$emit('deleteCliente',cliente.id)">
           Borrar </button>
 </form>
</template>
<script>
export default {
  props: ['cliente'],
  inject: ['clientesDAO'],
  emits: ['updateCliente','deleteCliente'],
  methods: {
   async guardarCliente() {
     if (this.validarCliente()) {
       try {
         //...create/update cliente with DAO
         this.$emit('updateCliente')
       } catch (err) {
         //Show server-side errors
       }
     }
   }, validarCliente() { /*...*/ },
 }
}
</script>
```

3.2

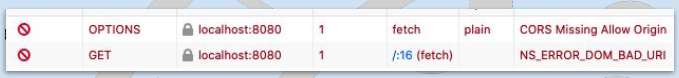
1

3.1

Cross-Origin Resource Sharing, CORS

- **Problema:** Si desplegamos una aplicación JS en un servidor/puerto diferente al del API REST, por defecto las peticiones AJAX no pueden alcanzarla por restricción CORS
- **Método de protección en el cliente**
  - El servidor envía encabezados al navegador para indicar qué peticiones AJAX debe autorizar al mismo

- **Basado en encabezados enviados por el servidor al navegador**
  - **El navegador pregunta** *(pre-fly OPTIONS request)* al sitio si puede hacérselas
  - **El servidor responde** indicando:
    - orígenes que admite
    - métodos (GET, POST,...) y encabezados que aceptará



  - **Si el servidor no acepta** → el navegador rechaza la petición
- **Encabezados devueltos por servidor con CORS activado:**
  - **Access-Control-Allow-Origin:** http://localhost:3030 // Usar * para cualquier origen (¡ojo!)
  - **Access-Control-Allow-Methods:** GET, POST, … // Métodos permitidos
  - **Access-Control-Allow-Headers:** Content-Type ,... //Encabezados que se admiten

Configuración CORS en JAX-RS

- **Métodos de generación de encabezados CORS en JAX-RS**
  - **Localmente, en métodos de controlador REST**
    - Utilizar método header() del objeto *Response*
  - **A nivel global del API REST**
    - Filtro global para generar encabezados con anotación @Provider
    - Se ejecutará para todas las peticiones

```
@Provider
public class ClubCORSAPIFilter implements ContainerResponseFilter {

   @Override
   public void filter(ContainerRequestContext requestContext, ContainerResponseContext response) {
     response.getHeaders().putSingle("Access-Control-Allow-Origin", "http://localhost:3000");
     response.getHeaders().putSingle("Access-Control-Allow-Methods", "OPTIONS, GET, POST, PUT, DELETE");
     response.getHeaders().putSingle("Access-Control-Allow-Headers", "Content-Type");
   }
}
```

Importante: el **origen *** en peticiones CORS solo debe permitirse para APIs con información pública (sólo consultas)…

Empaquetado de una aplicación Vue

- **Objetivo:** procesamiento de sus contenidos para generar una versión optimizada, descargable y ejecutable directamente en el navegador
- **Operaciones habituales**
  - Descarga de dependencias de terceros
  - Análisis estático (*linter*) para detectar errores
  - Procesamiento de ficheros SFC (.vue) para convertirlos en código JS
  - Transpilación de código en otros formatos, e.g typescript→JS, jsx→JS, scss→css, …
  - Empaquetado de los ficheros necesarios: eliminación de elementos no usados (*tree-shaking*), compresión (*minify*), agrupamiento en uno o varios ficheros (*bundle*)
- **Herramientas:** integran las herramientas necesarias
  - **Vite,** soporta múltiples frameworks y basada en módulos ES6 (utilizaremos ésta)
  - **Vue-CLI,** específica para Vue y basada en *webpack* (*bundler*)

Organización de ficheros con Vite

- **Directorio aplicación**
  - **public/** contenidos estáticos
    - favicon.ico
    - images
  - **src/**
    - **AppClientes.vue** Componente raíz
    - **main.js** Inicialización de Vue
    - **assets/** contenidos estáticos importables: json, css, imgs
    - **components/**
      - **clientes** e.g. carpeta por entidad/funcionalidad
        - **ClienteForm.vue** Implementación de componentes
      - **commons** Componentes reutilizables
        - ClubHeader.vue
        - ListaErrores.vue
    - **services**
      - **clientes.services.js** Implementación de servicios
    - **…**
  - **index.html** default page
  - **appclientes.html** layout para aplicación SPA
  - package.json Configuración npm del proyecto: dependencias, scripts inicio, configuraciom utilidades, etc.
  - package-lock.json Detalle de dependencias (generado con: *npm install*)
  - *node_modules* Dependencias descargadas (generado con: *npm install*)
  - *dist* Versión comprimida (bundled) de la aplicación (generado con: *npm run build*)

Inicialización de la aplicación:

```
$ npm init vue@latest
```

Nota: si el proyecto es parte (front-end) de un api REST JakartaEE (back-end) se puede incluir en la carpeta src/main del proyecto Maven

Operaciones con Vite

- El asistente de Vite genera un fichero de proyecto package.json para gestionarlo con el gestor de paquetes npm de *nodejs*
  - Contiene dependencias de la aplicación cuando se empaqueta (*development*) y al ejecutarse
  - Permite lanzar operaciones habituales (*scripts*)
  - Versión *package-lock.json* con todas las versiones de dependencias para garantizar un despliegue similar (debe borrarse si se desean actualizar las dependencias a versiones más recientes)
- Operaciones habituales
  - npm install o npm install paquete, descarga o añade dependencias al fichero package.json
  - npm run **dev**, ejecuta la aplicación en modo test para reflejar los cambios en tiempo real en el navegador (*hot-reload*). Acceso en http://localhost:3000
  - npm run **build**, genera la versión empaquetada de la aplicación. Por defecto en directorio *dist (*se puede cambiar a *webapp* desde *vite.config.json*)
  - npm run **preview**, lanza un servidor local de pruebas sobre carpeta *dist*

Conclusiones

- Los frameworks JS para desarrollo de aplicaciones web en el cliente definen **patrones de diseño** aplicando conceptos de Ingeniería del Software
- Los frameworks JS para desarrollo web actuales utilizan un enfoque de **diseño orientado a componentes**
- Un **componente** define la apariencia y comportamiento de un fragmento de vista. Está formado por una plantilla, un modelo de vista aislado y opcionalmente métodos para atender a eventos de la vista
- **VueJS** es un framework MVVM con características similares a otros frameworks como Angular o React
- Las **plantillas** de VueJS utilizan el DOM del documento añadiendo elementos sintácticos adicionales o **directivas** encargados de controlar su comportamiento
- Los **módulos VueJS** o SFC permiten agrupar en un fichero los aspectos de vista, control y estilos de cada componente

Bibliografía

- *VueJS Documentation*. VueJS, 2022. Disponible en https://vuejs.org/guide/introduction.html

Lecturas Complementarias

- Tutorial interactivo VueJS. Disponible en https://vuejs.org/tutorial/
- *VueJS Style-guide*. VueJS, 2022. Disponible en https://vuejs.org/style-guide/
- *VueJS API.* VueJS, 2022. Disponible en https://vuejs.org/api/
- López, Daniel, Pelaez, Andres. *Full-Stack Web Development with Jakarta EE and Vue.JS*. 1st ed. Apress, 2021. [Recurso electrónico UJA]
- Ribeiro, Heitor. Vue.js 3 Cookbook. 1st edition. Packt Publishing, 2020. [Recurso electrónico UJA]
- You, Evan. *Vue3 as the new default*. VueJS, 2022. Disponible en https://blog.vuejs.org/posts/vue-3-as-the-new-default.html