S I A M

S I S T E M A I N T E G R A D O D E A D M I N I S T R A C I Ó N M U N I C I P A L

COMPONENTES ANGULAR (V. 1.0)

FRONT END DESIGN

06/03/2020 – 07/03/2020

Contenido

1. HTMLInputs (Tipos de datos)

En la programación POO estamos acostumbrados a ver los tipos de datos como definiciones de un estado en memoria que representa la forma en que esta se va a comportar, sin embargo, en este caso se sobre extiende el comportamiento y la manera en cómo estos trabajarán.

Para el sistema se crearon algunos objetos de datos llamados ***Fake Objects***, ya que se estructuran como tales (Objetos planos) con una clase asociada, pero representan un elemento (HTMLInputElement HTML5) y se comportan como un dato (Typescript Type) – A estos Fake Objects les llamaremos **"**`HTMLInput`**"**, o Tipo de dato.

Nota: A continuación, se presentan las propiedades que poseen los diferentes tipos de datos, no obstante, la descripción viene en el apartado 1.2 de este documento.

El HTMLInput es objeto de tipo abstracto que trabaja de una u otra forma dependiendo del tipo (propiedad tipo type) que se desee crear; A continuación, se detallan las propiedades de los diferentes tipos de datos (HTMLInput.type):

1.1. HTMLInputs: Propiedades

```
type?: 'text' | 'hidden' | 'number' | 'date'
```

Estos tipos de datos son los más generales, de no declarase un tipo de dato dentro de HTMLInput, `'text'` será el valor por defecto. Similares a los elementos de HTML:

a. Text genera un input tipo texto.
b. Hidden genera un input escondido que será mandado al backend.
c. Number genera un input tipo número (no acepta texto).
d. Date genera un input tipo date – para formatearse con el backend se debe usar la función: `this.utils.javaDate2JS(<data>)`

Propiedad	Uso	Tipo de dato	Valor/defecto
primaryKey	Opcional	boolean	False
clazz	Opcional	String	Null
value	Opcional	Any	‘’
alias	Opcional	String	‘’
modalValue	Opcional	String	Null
required	Opcional	boolean	False
readonly	Opcional	boolean	False

```
type?: 'button'
```

Los botones son utilizados únicamente si se necesita una acción que **NO** sea (guardar / editar / Mostrar un modal o select).

Propiedad	Uso	Tipo de dato	Valor/defecto
buttonService	Opcional	() => any (Closure)	Null
clazz	Opcional	String	Null
alias	Opcional	String	‘’
iconName	Opcional	String	Null

```
type?: 'detail'
```

Los campos tipo detalle generaran un botón que abrirá un “***DetailView***”, este se explicará más adelante, el cual permitirá una acción “detailService” que será usado para guardar, modificar o crear un nuevo elemento.

Propiedad	Uso	Tipo de dato	Valor/defecto
detailService	Opcional	() => Any (Closure)	Null

```
type?: 'title' | 'subtitle'
```

Los títulos cumplen más de una función:

a. Identificadores de bloques (visualmente atractivo y poco confuso)
b. Acordeón (abrir o cerrar un bloque).
c. Busca darle legibilidad, a un proceso más intuitivo a la vista.

Propiedad	**Uso**	**Tipo de dato**	**Valor/defecto**
clazz	Opcional	String	Null
value	Opcional	String	‘’
titleCollapsed	Opcional	Boolean	false
titleUnHide	Opcional	Boolean	false

```
type?: 'select'
```

Simula un select en la aplicación, bastante sencillo de usar y sin mayor detalle.

Propiedad	**Uso**	**Tipo de dato**	**Valor/defecto**
clazz	Opcional	String	Null
value	Opcional	Any	‘’
alias	Opcional	String	‘’
required	Opcional	Boolean	False
readonly	Opcional	Boolean	False
selectService	Opcional	() => Any (Closure)	Null

```
type?: 'checkbox'
```

Simula un checkbox en la aplicación, a veces debemos decirle qué valor tomar en caso de estar chequeado o no, para ello son los valores checkedValue y checkedValue.

Propiedad	Uso	Tipo de dato	Valor/defecto
clazz	Opcional	String	Null
value	Opcional	Any	‘’
alias	Opcional	String	‘’
required	Opcional	Boolean	False
readonly	Opcional	Boolean	False
checkedValue	Opcional	Any	‘S’
uncheckedValue	Opcional	Any	‘N’

```
type?: 'mixture' | 'modal'
```

El modal es un valor **depreciado**, de lo posible no utilizar, ya que en otras versiones podría desaparecer. El mixture viene a reemplazar el modal, este es un grupo de un input + un modal el cual actualiza múltiples valores con un doble clic en una fila del conjunto.

Propiedad	Tipo de uso	Tipo de dato	Valor por defecto
clazz	Opcional	String	Null
value	Opcional	Any	‘’
alias	Opcional	String	‘’
required	Opcional	Boolean	False
readonly	Opcional	Boolean	False
modalService	Opcional	() => Any (Closure)	Null
modalValue	Opcional	String	Null
modalWidth	Opcional	String	Null

1.2. HTMLInputs: Detalle de las propiedades

Propiedad	**Descripción**
type?	Determina el tipo de dato que va a representar el HTMLInput que se está programando.
primaryKey?	De ser true la identificará como llave primaria y no lo dejará editar este valor.
value?	Es el valor por defecto de un input, normalmente vienen del backend.
alias?	Es el nombre de la columna en la tabla y el valor del label cuando se expande un DetailView.
titleUnHide?	Cuando está en true, siempre será visible (no podrá ser colapsado con clic). Por defecto es false.
titleCollapsed?	Cuando está en true, siempre será visible (no podrá ser colapsado con clic). Por defecto es false.
modalValue?	Puede asociar a la columna de un modal para obtener el dato. Un mixture crea una lista de HTMLInputs con nombres de columnas (“llaves de un objeto”), estas al ser colocadas en un elemento con esta propiedad – en el modal al darle doble clic a la fila o texto, buscará y reemplazará el valor por este.
modalWidth?	Modifica el width de un modal como String puede ser en pixeles (120px) o en porcentaje (80%) o inclusive en Viewports (85vw).
required?	True le indica a una Forma que debe validar que siempre esté presente cuando se busque guardar o modificar un registro.
readonly?	True determina que un dato no puede ser modificado bajo ninguna manera en ninguna circunstancia.
modalService()?	El servicio que se corre al abrir un modal o un mixture, además, está asociado con los inputs con la propiedad modalValue seteado.
detailService()?	Es el servicio que abre un botón “+ Información” el cual muestra toda una nueva lógica estructural en un DetailView.

Propiedad	Descripción

selectService()?	Es el servicio que genera un Select puede ser traída desde la base de datos (API Rest -> Promise) como un arreglo de arreglos (matriz). Ejemplo:
	`return [`
	`['M', 'Mayor'],`
	`['S', 'Subcuenta'],`
	`['D', 'Detalle']`
	`];`

buttonService()?	Es el servicio que se llama cuando un botón es presionado.
checkedValue?	Es el valor de un checkbox cuando su estado es checked
uncheckedValue?	Es el valor de un checkbox cuando su estado es unchecked
clazz?	Modifica o adhiere una clase a un elemento seleccionado; Aguanta cualquier valor de CSS mientras esté correctamente seteado.
iconName?	Está asociado a los mat-icons, si el presente elemento contiene esta propiedad, acepta cualquier nombre de ícono de esta página: *https://material.io/resources/icons/?style=round*. ***SOLO PONER EL NOMBRE, NADA DE HTML O PARECIDO***.

Nota: "?" Representa el identificador de opcional.

Las propiedades de los tipos de datos son dinámicas y tienden a aparecen nuevas funcionalidades a menudo – puede que existan más que no están en esta lista, para conocer la totalidad de propiedades, cuando se encuentre programando puede darle "Ctrl + Space" sobre un "`HTMLInput`" para verlas.

2. Data Table: Componente

El componente de tablas está diseñado conforme a las necesidades del departamento; Con ayuda de los tipos de datos (HTMLInput) - ***se busca disminuir el tiempo de desarrollo y aumentar la eficiencia.*** Cualquier necesidad adicional que pueda ser desarrollada en una tabla puede ser integrada de manera rápida.

2.1. Data Table: Paso a paso

1. Se debe crear la estructura HTML (app-data-table); **No hace falta incluir los métodos input/output**. A continuación, el detalle de estos:

Propiedad	**Tipo**	**Descripción**
doubleClickRowEmitter	Output	Este es único para el uso del Modal o mixture, devuelve un evento para ser consumida la data cuando le de doble clic a una fila o elemento. Nota: Que sea único para el uso del modal, no implica que no puede ser usado.
saveRowEmitter	Output	***MODIFICA*** un dato existente cuando se ***selecciona*** del listado de la tabla.
saveNewRowEmitter	Output	**GUARDA UN NUEVO DATO,** se dispara en el botón guardar cuando después de abrir un Detail View con la opción Nuevo elemento.
newItemVisible	Input	**TRUE por defecto**, si está en true muestra el botón para agregar un nuevo registro
deleteItemVisible	Input	**TRUE por defecto**, si está en true muestra el botón para eliminar un registro
exportVisible	Input	**TRUE por defecto**, si está en true muestra el botón para exportar a Excel la tabla
filterVisible	Input	**TRUE por defecto**, si está en true muestra el botón para agregar un nuevo registro
detailViewBottom	Input	**FALSE por defecto**, si está en true el DetailView se mostrará en la parte de abajo.

Nota: Un input u output son funciones o variables que son recibidas o emitidas respectivamente.

El siguiente es un ejemplo de cómo se vería el HTML con un Data Table, cabe destacar que en este caso se utiliza la misma función para modificar o guardar un nuevo elemento, solo se manda un booleano indicando cómo trabajar la función.

```
<!--
   Se crea un DataTableComponent con dos métodos output (listeners), tal como
   Se explicó en el apartado 2.1, los métodos que se ven en esta tabla
   "(saveRowEmitter)" y "(saveNewRowEmitter)" no son obligatorios. Lo que le
   estamos diciendo en este caso es que:

   tenemos un método guardarItem en alguna parte del Typescript, que está
   esperando a que se ejecute un botón para guardar datos – que enviará los
   datos de los inputs al backend.

   Nota: $event trae los datos del "Emmiter" (Evento corrido).
-->
<app-data table
   (saveRowEmitter)="guardarItem($event, false)"
   (saveNewRowEmitter)="guardarItem($event, true)">
</app-data-table>
```

2. Programar el servicio que escuchará la petición al backend**:**

   Para que el servicio corra exitosamente deberá ser ejecutado desde el método: "`this.DataTableComponent.createTable(<método>)`". Podríamos definir este método como el inicializador de la Data Table. En el siguiente ejemplo vemos que se llama desde el "`ngAfterViewInit`" (Después de que inicie la renderización), esto para que se muestre desde que se crea la vista; sin embargo, esto no es obligatorio, **puede ser llamado en el momento que se requiera**.

   También podremos notar el método: "`this.DataTableComponent.newItemFunction(<método>)`". Si esto tiene un método asociado, cuando se presione el botón de nuevo elemento correrá este método.

```
@ViewChild(DataTableComponent, { static: true })
           DataTableComponent: DataTableComponent;

 ngAfterViewInit() {
     this.DataTableComponent.createTable(() => this.loadCatalogo());
     this.DataTableComponent.newItemFunction = () => this.createDVCtaDet();
 }

 async loadCatalogo() {
     const dTable: any[] = [];
     const data = await this.apiRest.getCatalogoPresup().toPromise();
     for (const ele of data) {
         dTable.push({
             codCtaPre: new HTMLInput({
                 alias: 'Número Cuenta',
                 value: ele.id.codCtaPre
             }),
             nomCta: new HTMLInput({
                 alias: 'Nombre Cuenta',
                 value: ele.nomCta
             }),
             codCtaSup: new HTMLInput({
                 alias: 'Cuenta Superior',
                 value: ele.codCtaSup
             }),
             nivCta: new HTMLInput({
                 alias: 'Nivel',
                 value: ele.nivCta,
                 type: 'select',
                 selectService: () => this.getTipoCuenta()
             }),
             detalle: new HTMLInput({
                 type: 'detail',
                 detailService: () => this.createDVCtaDet(ele)
             })
         });
     }
     return dTable;
 }
```

En este ejemplo una vez importado el DataTableComponent creamos la tabla con "`createTable`" el cual entrará a la función "`loadCatalogo`" y recorrerá una lista llamada desde el backend.

IMPORTANTE: Esta al ser la tabla general (no la detalle) no requiere mayor detalle de las propiedades; Estas no se mostrarán, pero serán cargadas en memoria haciendo pesada la consulta.

Si podemos observar, hay un detalle (`type: 'detail'`) que llama al método "`this.createDVCtaDet(ele)`" aquí si podemos declarar los tipos a conveniencia:

```
async createDVCtaDet(data: any = null) {
    const obj = {
        subtitle: new HTMLInput({
            type: 'title',
            value: 'Información presupuestaria'
        }),
        'id.codCia': new HTMLInput({
            type: 'hidden',
            value: data?.id.codCia ?? '01',
            required: true
        }),
        nomCta: new HTMLInput({
            alias: 'Nombre Cuenta',
            value: data?.nomCta ?? null,
            clazz: 'col-4',
            required: true
        }),
        nivCta: new HTMLInput({
            alias: 'Nivel',
            value: data?.nivCta ?? null,
            type: 'select',
            clazz: 'col-2',
            selectService: () => this.getTipoCuenta(),
            required: true
        })
    };
    return obj;
}
```

A esta altura ya programamos la tabla inicial (`loadCatalogo()`) y el Detail View (`createDVCtaDet(data)`). El detail View es un estado que está esperando a ser disparado con un arreglo de elemento(s) de tipo "`HTMLInput`". Con esto ya terminamos nuestra tabla y si todo está correcto deberá poder verse en pantalla.

2.2. Data Table: Métodos públicos o lógicamente públicos

`reloadTable()`	Recarga los datos de la tabla (vuelve a realizar las consultas al backend). Utilizada cuando se guarda un registro y se quiere actualizar la tabla.
`createTable (<callback>)`	Crea la tabla de una manera asíncrona para que no interfiera con el rendimiento general de la página. **callback** llama a una función (llenado de datos) del componente.
`applyFilter()`	Aplica el filtro del input search sobre la tabla.

NOTA: Solo se muestran los métodos públicos o lógicamente públicos