

S I A M

S I S T E M A I N T E G R A D O DE 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) 3](#_Toc24301)

[1.1. HTMLInputs: Propiedades 3](#_Toc24302)

[1.2. HTMLInputs: Detalle de las propiedades 7](#_Toc24303)

[2. Data Table: Componente 9](#_Toc24304)

[2.1. Data Table: Paso a paso 9](#_Toc24305)

[2.2. Data Table: Métodos públicos o lógicamente públicos 13](#_Toc24306)

# 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):

## HTMLInputs: Propiedades

type?: 'text' | 'hidden' | ~~'number'~~ | 'date' | 'textarea'

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:

1. Text y textarea generan un input tipo texto.
2. Hidden genera un input escondido que será mandado al backend.
3. Number – ya no se usa, utilizar 'text' con la propiedad format: 'money'
4. Date genera un input tipo date – para formatearse con el backend se debe usar la función: this.utils.javaDate2JS(<data>), **importante**: por lo general no usamos el date que mandamos del frontend, en el backend sustituye este valor manualmente.

### *Propiedad* Uso Tipo de dato Valor/defecto

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *primaryKey* | Opcional | boolean | False | No es muy utilizado, genera un efecto visual en los Datatables. |
| *clazz* | Opcional | String | Null | Puede incorporar clases de CSS o inclusive Bootstrap. |
| *value* | Opcional | Any | ‘’ | Es el valor por defecto, normalmente viene desde el backend. |
| *alias* | Opcional | String | ‘’ | Se mostrará como un label en formularios y títulos de columna en listas y tablas. |
| *modalValue* | Opcional | String | Null | Cuando hay un modal, aquí le decimos que queremos que tome el valor de X columna al elegir una fila del modal. |
| *required* | Opcional | boolean | False | Verifica la necesidad de que este input esté seteado antes de aplicar un botón, los botones “custom” o creados manualmente, no validan esta información. |
| *readonly* | Opcional | Boolean | False | Hace lock a un input dentro de un **DATATABLE**, en los forms no funciona, usar **FORCEREADONLY** en cambio. |
| *forcereadonly* | Opcional | Boolean | False | Ejecuta la misma acción de readonly, funcionando también en FORMS |
| *Style* | Opcional | String | ‘’ | Inserta un style dentro, para corregir errores puntuales o realizar acciones extra. |
| *ignoreJson* | Opcional | Boolean | False | Le dice al frontend que ignore este campo. Funciona únicamente con la función **validateForm** |
| *Format* | Opcional | ‘money’,  ’cedula’ | ‘none’ | Money, genera el formato moneda utilizado por la municipalidad. Cédula aun no está programado |
| *onChange* | Opcional | Function |  | Ejecuta una acción al cambiar de input |
| *onClick* | Opcional | Function |  | Ejecuta una acción con click |
| *clickOnLock* | Opcional | Boolean |  | Permite utilizar el onClick aun con forcereadonly |

type?: 'button'

Los botones trabajan de la misma manera que en HTML, siempre importantes en los formularios.

**IMPORTANTE:** los Data-Tables tienen botones propios de guardar, modificar, agregar nuevo elemento. Si está usando Data-Tables, por favor usar estos botones siempre y cuando sea posible, ya que realizan la lógica interna para mandar los datos de manera fluida, sin mayor complejidad, al backend.

### *Propiedad* Uso Tipo de dato Valor/defecto

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *buttonService* | Opcional | Function | Null | Es la función que almacena la acción al hacer click en el botón |
| *clazz* | Opcional | String | Null | Puede incorporar clases de CSS o inclusive Bootstrap. |
| *alias* | Opcional | String | ‘’ | Se mostrará como un label en formularios y títulos de columna en listas y tablas. |
| *iconName* | Opcional | String | Null | El nombre del ícono de material-icons-design |
| *Style* | Opcional | String | ‘’ | Inserta un style dentro, para corregir errores puntuales o realizar acciones extra. |
| *btnType* | Opcional | Btn-error,  Btn-success | ‘’ | Genera un botón con un estilo predefinido. |

type?: 'detail'

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

A manera resumida el DetailView abre un “Sub-formulario” de detalle, el cual se puede guardar, modificar o eliminar, sin mayor dificultad y sin tener que crear otra pantalla.

### *Propiedad* Uso Tipo de dat Valor/defecto

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *detailService* | Opcional | Function |  | Es la función que se corre cuando se presiona el botón |
| *afterCall* | Opcional | Function |  | Ocurre cuando termina de ejecutar el botón |

Como se puede observar a la derecha hay un botón de “+ Info.”, el cual despliega la parte de arriba donde se aprecia “Detalle Cédula: …”. Nótese también las posibilidades arriba a la derecha, “registro anterior, registro siguiente, desbloquear o bloquear campos, salvar datos y cerrar ventana detalla”.

type?: 'title' | 'subtitle'

Los títulos cumplen más de una función:

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

### *Propiedad* Uso Tipo de dato Valor/defecto

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Clazz*  *Value* | Opcional | String | Null | Puede incorporar clases de CSS o inclusive Bootstrap. |
| Opcional | String | ‘’ | Nombre del título |
| *titleCollapsed* | Opcional | Boolean | false | Se muestra colapsado o no |
| *titleUnHide* | Opcional | Boolean | false | Se muestra escondido o no |

type?: 'bootstrapColumn'

en algunas ocasiones necesitamos tener bloques de elementos adyacentes, esto viene a simular lo que en HTML (Emmet) sería “row > col-6\*2”. En palabras bonitas, nos permitirá crear N columnas para presentar tablas o datos al lado.

**NOTA:** Para utilizar este tipo es necesario tener conocimientos básicos en col-\* de Bootstrap, para más información, pueden utilizar el enlace: <https://getbootstrap.com/docs/4.0/layout/grid/>. Cabe destacar que en esta aplicación usamos normalmente solo las columnas sin diseño adaptativo: col-\*

### *Propiedad* Uso Tipo de dato Valor/defecto

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Clazz* | Opcional | String | Null | 'col-1' | 'col-2' | 'col-3' | 'col-4' | 'col-5' | 'col-6' | 'col-7' | 'col-8' | 'col-9' | 'col-10' | 'col-11' | 'col-12' |'col-1 pr-0 pl-0' | 'col-2 pr-0 pl-0' | 'col-3 pr-0 pl-0' | 'col-4 pr-0 pl-0' | 'col-5 pr-0 pl-0' | 'col-6 pr-0 pl-0' |'col-7 pr-0 pl-0' | 'col-8 pr-0 pl-0' | 'col-9 pr-0 pl-0' | 'col-10 pr-0 pl-0' | 'col-11 pr-0 pl-0' | 'col-12 pr-0 pl-0' | |

type?: 'select'

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

***Propiedad* Uso Tipo de dato Valor/defecto**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *clazz*  *value* | Opcional | String | Null | Puede incorporar clases de CSS o inclusive Bootstrap. |
| Opcional | Any | ‘’ | Es el valor por defecto, normalmente viene desde el backend. |
| *alias* | Opcional | String | ‘’ | Se mostrará como un label en formularios y títulos de columna en listas y tablas. |
| *required* | Opcional | Boolean | False | Verifica la necesidad de que este input esté seteado antes de aplicar un botón, los botones “custom” o creados manualmente, no validan esta información. |
| *readonly* | Opcional | Boolean | False | Hace lock a un input dentro de un **DATATABLE**, en los forms no funciona, usar **FORCEREADONLY** en cambio. |
| *selectService* | Opcional | Function | Null | Es la función que corre al seleccionar una opción. |
| *forcereadonly* | Opcional | Boolean | False | Ejecuta la misma acción de readonly, funcionando también en FORMS |
| *ignoreJson* | Opcional | Boolean | False | Le dice al frontend que ignore este campo. Funciona únicamente con la función **validateForm** |

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 required* | Opcional | String | ‘’ |
| Opcional | Boolean | False |
| *readonly* | Opcional | Boolean | False |
| *checkedValue uncheckedValue* | Opcional | Any | ‘S’ |
| 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 value* | Opcional | String | Null |
| Opcional | Any | ‘’ |
| *alias required readonly* | Opcional | String | ‘’ |
| Opcional | Boolean | False |
| Opcional | Boolean | False |
| *modalService* | Opcional | () => Any (Closure) | Null |
| *modalValue* | Opcional | String | Null |
| *modalWidth* | Opcional | String | Null |

# 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.

## 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*  *saveRowEmitter*  *saveNewRowEmitter*  *newItemVisible*  *deleteItemVisible*  *exportVisible*  *filterVisible*  *detailViewBottom* | 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. |
| Output | ***MODIFICA*** un dato existente cuando se ***selecciona*** del listado de la tabla. |
| Output | **GUARDA UN NUEVO DATO,** se dispara en el botón guardar cuando después de abrir un Detail View con la opción Nuevo elemento. |
| Input | **TRUE por defecto**, si está en true muestra el botón para agregar un nuevo registro |
| Input | **TRUE por defecto**, si está en true muestra el botón para eliminar un registro |
| Input | **TRUE por defecto**, si está en true muestra el botón para exportar a Excel la tabla |
| Input | **TRUE por defecto**, si está en true muestra el botón para agregar un nuevo registro |
| 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.

## 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