



5 DE SEPTIEMBRE DE 2017

Gallery Manual Técnico

MANUAL DE USO Y EXTENSIÓN

Contenido

1	Introducción	2
2	Glosario	3
3	Requisitos	4
4	Instrucciones de Ejecución	4
4.1	Paso 1: Instalación Librerías y Herramientas	4
4.2	Archivos principales	5
4.3	Configuración General.....	5
4.4	Plantillas	6
4.5	Estructura de carpeta plantilla estándar.....	6
4.6	Edición GENERAL, POR PLANTILLA Y POR CLIENTE	11
4.7	Archivo de edición de collection.directive.videos.tpl.html.....	13
4.8	Edición personalizada extendida, archivos: properties.edit.json y properties.jslib.extension.js	14
4.9	Extensión de librerías y uso de app.options.custom.js	19
4.10	Creando Plantilla BASE desde línea de comandos con GULP	21

1 Introducción

El siguiente documento tiene como objetivo explicar todos los pasos necesarios para la correcta instalación, uso y extensión de plantillas, configuraciones y datos técnicos para la extensión del CORE y de las plantillas.

2 Glosario

- **CORE:** El Centro de la aplicación, el corazón de las librerías y funcionamiento de la aplicación.
- **GULP:** Herramienta encargada de los trabajos automatizados de la aplicación.
- **AngularJS:** Framework encargado del esqueleto de la aplicación.
- **Nodejs NPM:** Herramienta para el manejo de dependencias y librerías

3 Requisitos

Los principales requisitos para la extensión, edición y entendimiento del programa son:

1. Conocimiento HTML y Bootstrap nivel intermedio-avanzado
2. Conocimiento Angularjs nivel intermedio
 - Directivas
 - Servicios
 - Filtros
 - Controladores
 - Plantillas
3. Conocimiento JQuery nivel básico-intermedio
 - Selectores
 - Manipulaciones generales
4. Conocimiento CSS1,2,3 Nivel básico-intermedio

4 Instrucciones de Ejecución

A continuación se describen los pasos necesarios para poder ejecutar **Gallery** en su equipo

4.1 Paso 1: Instalación Librerías y Herramientas

1. Debe tener instalado NODEJS y GULP (En la carpeta del proyecto ejecutar npm -i)

```
{ } package.json x
1  {
2    "name": "gallery",
3    "version": "1.0.1",
4    "devDependencies": {
5      "canonical-path": "0.0.2",
6      "dgeni": "^0.4.9",
7      "dgeni-packages": "^0.20.0",
8      "gulp": "",
9      "gulp-clean": "",
10     "gulp-clean-css": "",
11     "gulp-concat": "",
12     "gulp-connect": "^5.0.0",
13     "gulp-header": "",
14     "gulp-livereload": "",
15     "gulp-markdox": "^0.1.2",
16     "gulp-minify": "",
17     "gulp-sass": "",
18     "lodash": "^4.17.4",
19     "run-sequence": ""
20   }
21 }
```

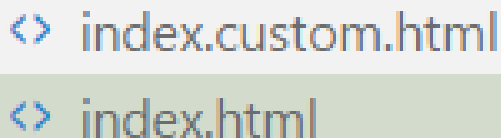
2. Para Compilar el proyecto se debe tener GULP y ejecutar el siguiente comando:

```
Toshiba@Toshiba-PC MINGW64 /c/AppServ/www/ZGroup/nuevo_gallery
$ gulp deployDev
[12:55:13] Using gulpfile C:\AppServ\www\ZGroup\nuevo_gallery\c
[12:55:13] Starting 'deployDev'...
[12:55:13] Starting 'cleanDist'...
[12:55:13] Finished 'cleanDist' after 32 ms
[12:55:13] Starting 'movingDist'...
[12:55:13] Finished 'movingDist' after 3.07 ms
[12:55:13] Starting 'sass'...
[12:55:13] Finished 'sass' after 63 ms
[12:55:13] Starting 'distCss'...
[12:55:13] Finished 'distCss' after 14 ms
[12:55:13] Starting 'distCssEditMode'...
[12:55:13] Finished 'distCssEditMode' after 14 ms
[12:55:13] Starting 'minify-css'...
[12:55:13] Finished 'minify-css' after 637 ms
[12:55:13] Starting 'distJsDev'...
[12:55:13] Finished 'distJsDev' after 40 ms
[12:55:13] Starting 'distJsEditMode'...
[12:55:13] Finished 'distJsEditMode' after 66 ms
[12:55:13] Starting 'compress'...
[12:55:13] Finished 'compress' after 3.01 ms
Deploy Finishing ... (timestamp: 2017-8-30 12:55)
```

Esto generará los archivo necesarios en la carpeta **dist/**

4.2 Archivos principales

Se tienen dos archivos principales INDEX.HTML y INDEX.CUSTOM.HTML uno es el index general de usuarios y el otro el index de la edición de componentes y css respectivamente.



The image shows two file icons. The top icon is light blue and labeled 'index.custom.html'. The bottom icon is light green and labeled 'index.html'.

La diferencia principal es que el de edición tiene más librerías especiales para poder ejecutar las necesidades en edición, en ningún caso el index.html podrá ejecutar temas de edición por su origen, no tiene más librería.

4.3 Configuración General

Existe un archivo de configuración Global para modificar en ambiente desarrollo, se llama config.js, dentro de la carpeta **app/**:

```

/**
 * @ngdoc method
 * @name zgconfig
 *
 * Inicializa las variables globales y el servicio inicial
 *
 * @param {string} APIHOST ruta del servicio
 * @param {string} ENVIRONMENT variable
 * @param {string} URL Base del sistema, URL RAIZ
 * @param {string} URL De <<siteAlias>> Para hacer match con la configuración y JSON del servicio entregado
 * @return nothing
 */
zgconfig.init('https://pc95hghh0g.execute-api.us-east-1.amazonaws.com/', 'v1', window.location.host, window.location.host);

```

Una vez modificado se necesita correr el proceso GULP , misma situación si se modifica uno de todos los otros JS dentro de la carpeta APP/.

4.4 Plantillas

Las carpetas de las plantillas están en **template/**, ante cualquier nueva plantilla se debe crear una carpeta dentro de template y agregar las configuraciones necesarias para su funcionamiento.

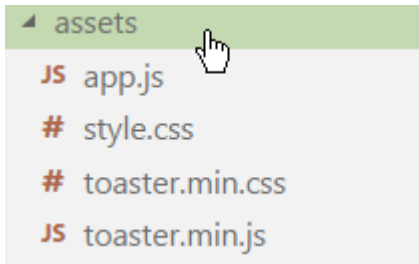
Nota: Puede ser útil copiar una plantilla como **“general”** y modificar sobre esa en su nueva carpeta.

4.5 Estructura de carpeta plantilla estándar.

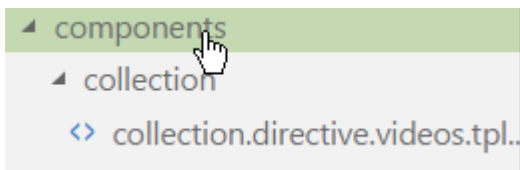
Los archivos estándar necesarios para crear una plantilla son los siguientes.



- **ASSETS:**
 - Carpeta con los recursos generales de la plantilla
 - Recursos como css, html y js



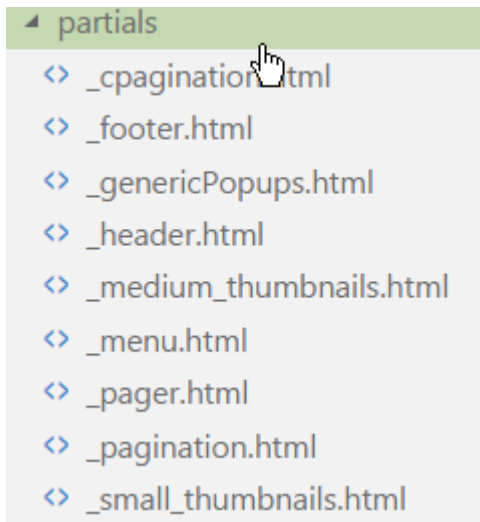
- **COMPONENTS:**
 - Si se desea reemplazar un componente general como el de colecciones de videos de la directiva en esta carpeta se deja el html o el js a reemplazar.
 - Es opcional



- **CONFIG:**
 - Son configuraciones para ediciones especiales, podemos agregar nuevos campos, formulas y situaciones al editar. Si el campo CSS que queremos editar no nos sirve con las opciones actuales, podemos deliberadamente crear el nuestro para nuestra plantilla, por ejemplo campo de edición de Font-weight.
 - Es opcional



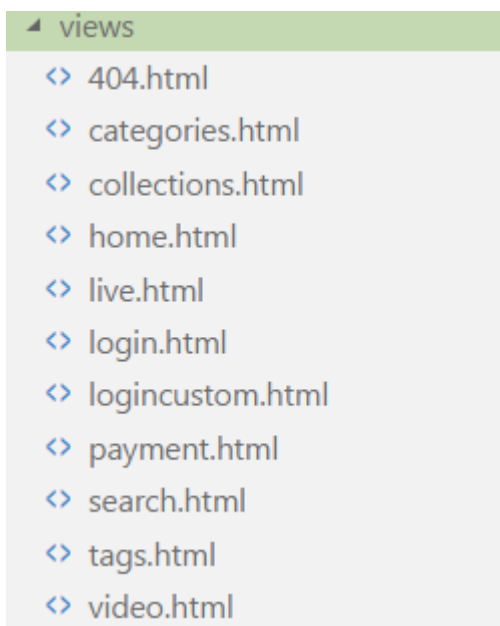
- **PARTIALS:**
 - Las vistas/htmls parciales son incluidos por otras vistas para su reutilización a nivel de la aplicación general del sitio.
 - Opcionales



- PAYMENT:
 - Es la carpeta con la plantilla y html de pago
 - Opcional



- VIEWS:
 - Son las vistas generales que contienen un html y sus componentes, en ellas encontramos vistas home, de videos y otras.
 - Están controladas de forma genérica por el CORE de la aplicación
 - No son opcionales “home”, es la vista de partida, se deben mantener los nombres, rutas y extensiones para poder funcionar.



- App.optionscustom.js
 - Módulo de extensión de angularjs, nos permite extender: RUTAS, CONTROLADORES, SERVICIOS, FILTROS Y DIRECTIVAS si se requiere. La idea de este módulo es no modificar el CORE de la aplicación para situaciones específicas de la plantilla, por ejemplo un nuevo componente o un nuevo formulario que necesite tener un controlador nuevo para validar situaciones, con este modulo es sencilla su extensión y uso para cualquier programador angularjs nivel básico intermedio.
 - Opcional

Ejemplo:

```

JS app.options.custom.js x
1
2  angular.module("gallery.options.custom")
3  .run(['$rootScope', function($rootScope) {
4  |
5  |     console.log("Desarrollar situaciones generales")
6  | })
7  .controller('homeInitCtrl.custom', ['$rootScope','$scope',
8  |     function($rootScope,$scope) {
9  |         console.timeEnd("Termino Custom JS extension")
10 |
11 |         $scope.custom = {};
12 |
13 |
14 |         $scope.custom.msg ="PROPIEDAD DESDE EXTENSION";
15 |
16 |
17 |         $scope.custom.fnexample = function(){
18 |             alert("examples")
19 |         }
20 |
21 |         $scope.custom.pop = function(){
22 |             toaster.pop('info', "title", "text");
23 |         };
24 |
25 |     })
26 |

```

- Properties.edit.json
 - Este archivos es importante para poder definir los componentes que se vana editar, si no tenemos este archivo no podemos editar los componentes angularjs y html involucrados para activar las ediciones en línea de las directivas, se hace un match entre el id o clase del html y su correspondiente posibilidad de edición con el tipo

involucrado, también se puede llamar al tipo especial generado a partir de la carpeta de configuración.

- Es necesario para poder editar, no se encuentran elementos en los htmls para resolver ediciones, todo está en este archivo de configuración.

Ejemplo:

```
{ } properties.edit.json x
1  {
2    "custom_collection" : false,
3    "max_visible_optionsmenu": 5,
4    "edit_sections"    : [{
5      "cmp"             : "id",
6      "val"             : "bc_headerWrapper",
7      "options"         : ["bgurl", "bgcolor"],
8      "msg"             : "mínima de 1920x330 pixeles"
9    }, {
10     "cmp"             : "id",
11     "val"             : "bc_clientLogo",
12     "options"         : ["bgurl", "logoLayout"],
13     "msg"             : "máxima de 500 pixeles"
14   }, {
15     "cmp"             : "id",
16     "val"             : "otroid",
17     "options"         : ["color", "extras"],
18     "extras"          : [ ... ]
19   }, {
20     "cmp"             : "id",
21     "val"             : "bc_clientTitle",
22     "options"         : ["color", "extras"],
23     "extras"          : [ ... ]
24   }
25 ]
```

- Properties.json
 - Tenemos variables globales de la pagina normal de cara al usuario general, también disponemos la posibilidad de agregar librerías extras a la plantilla que no maneja el core de la aplicación.
 - También se pueden agregar CSS y JS arbitrarios para cargar de forma dinámica a la plantilla.
 - Cualquier carga de librería, JS, CSS u otro archivo involucrado en esta configuración solo afectara a la plantilla actual y no a las demás plantillas en el proyecto GALLERY. Obteniendo un mejor uso de la performance de las plantillas.
 - Si la librería requiere una inyección de modulo a angularjs esta configurado para poder inyectar un modulo de forma dinámica al core de la aplicación, con eso no es

necesario nunca tocar el core, quedando totalmente independiente una plantilla de otra, sin posibilidad de cruzar información entre estas.

- Maneja un orden de inyección, dependiendo de la posición del arreglo es la posición que se estará inyectando en la plantilla el archivo JS.

Ejemplo:

```
{ } properties.json x
1  {
2    "custom_collection" : false,
3    "libs": [
4      {
5        "id": "$1",
6        "src": "https://maps.googleapis.com/maps/api/js?libraries=placeses",
7        "async": false
8      },
9      {
10       "ref": "$1",
11       "src": "libs/vendor/google-maps/ng-map.min.js",
12       "async": false,
13       "module": "ngMap"
14     }
15   ],
16   "max_visible_optionsmenu": 7,
17   "css_files" : ["assets/style.css"],
18   "js_files" : ["assets/app.js"]
19 }
```

4.6 Edición GENERAL, POR PLANTILLA Y POR CLIENTE

- 1) En la edición GENERAL, todo lo que se modifique en el CORE afecta indirecta o directamente a todas las plantillas. Si cargamos un módulo en el CORE aunque no se ocupen en las otras plantillas, de todas formas presentara una CARGA DE KBS que quizás no es necesaria para otras plantillas.
- 2) En la edición por PLANTILLA, cada edición que se hace en esta sección, ya sea agregando nuevos componentes (properties.json, app.options.custom.js,...) de angular, librerías JS, css, configuraciones de videos y otras afectan solo a la plantilla y todos los CLIENTES que la utilizan. No afecta a otras plantillas ni directa ni indirectamente, simplemente están desconectadas unas de otras a este nivel.
- 3) En la edición POR CLIENTE tenemos un consumo de JSON inicial que tendrá configuraciones especiales para CADA cliente, totalmente opcionales. Sera capas de agregar secciones y dentro de estas secciones pueden ir código JS, incluso con angular,

HTML y CSS. Cualquier cambio a este nivel, solo afecta al cliente específico y no a todos los demás clientes, incluso si un cliente decide ocupar decenas de librerías esto no afecta ni directa ni indirectamente a ningún otro cliente, ya que toda esta carga se produce de forma dinámica por CLIENTE, esto es beneficioso para mantener una escalabilidad a nivel de aplicación Gallery, ya que aunque crezcan los clientes, no afecta a otros.

Ejemplo de particularidades por cliente:

```
var arr = [{
  id : "cmp_section_1",
  html: '<h4 id="head_11"></h4><h5 class="whirl">{{extra.subtitulo}}</h5><div id=
js : "console.log('Ejemplo...Extra'); $('head_11').html('Titulo dinamico')",
css : "#head_11{color:red}",
curlcss : [
  'https://bootswatch.com/cerulean/bootstrap.min.css',
  'http://www.cssscript.com/demo/simple-cross-browser-pure-css-loading-animat
]
}, {
  id : "cmp_section_2",
  html: "<a class='twitter-timeline' href='//twitter.com/cnn' data-chrome='nofoot
js : \"\",
script : [\"//platform.twitter.com/widgets.js\"],
css : \"\"
}]
```

Las secciones en estas particularidades tienen que estar previamente definidas en las plantillas como TAGS HTMLS para poder insertar la opción de cada cliente. Podríamos tener una casilla y un cliente querer TWITTER y otro cliente querer FB por ejemplo. Eso es gracias a estas secciones en blanco que quedan disponibles a nivel de PLANTILLA para su posterior uso.

En resumen, con esta escala, tenemos la posibilidad de tener decenas, cientos o miles de clientes y plantillas sin que a nivel de aplicación se vea afectada, o que una plantilla comience a afectar a otras, ya que todo crecimiento o agregación es individual así como sus extensiones de las plantillas, inclusive por cliente. Con este modelo evitamos la sobrecarga de librerías no deseadas en el CORE. Podemos tener un escenario de 1 plantilla y mil clientes, y cada particularidad de cada cliente no afecta a otro cliente usuario de la plantilla, como a su vez cada particularidad de las plantillas no afectan al CORE de la aplicación.

4.7 Archivo de edición de collection.directive.videos.tpl.html

Parara extender la visualización e videos se tiene que dejar la plantilla en la carpeta: /template/xy/components/collection/collection.directive.videos.tpl.html y el código html será reemplazado con la visualización normal de los videos.

As variables disponibles en esta plantilla son las siguientes:

```
<!--  
Variables disponibles:  
    Array<Object>    - Listado de Videos: videosCollectionList  
    String           - Url Base: rootPath  
    String           - Url completa: urlSite  
  
Params disponibles:  
    idCollection: '=',  
    idBlock: '@',  
    brcItems: '@',  
    brcClass: '@',  
    brcClassInfo: '@',  
    brcSize: '@',  
    brcPagination: '=',  
    brcCollections: '=',  
    brcPanel: '=',  
    brcShowTitle: '=',  
    brcPlayerAutoStart: '@',  
    brcPlayerSkin: '=',  
    brcIdVideo: '=',  
    favs: '=',  
    hideLikes: '='  
-->
```

Nota: Recordar dejar en **“true”** la colección en las propiedades para activar la directiva personalizada:

```

{} properties.json x
1  {
2  |  "custom_collection" : true,
3  |  "libs": [

```

4.8 Edición personalizada extendida, archivos: `properties.edit.json` y `properties.jslib.extension.js`

Para editar propiedades distintas a las disponibles, podemos extender este archivo y programar estas propiedades.

Tipos existentes:

- 1) color
- 2) bgcolor
- 3) logoLayout
- 4) bgurl
- 5) **extras (extensión de elementos de edición):**
 - a. text: Caja de texto normal
 - b. combo: Listado de elementos
 - c. **otro***: Se debe programar en `properties.jslib.extension.js`

Ejemplo de extensión de tipo **EXISTENTE**:

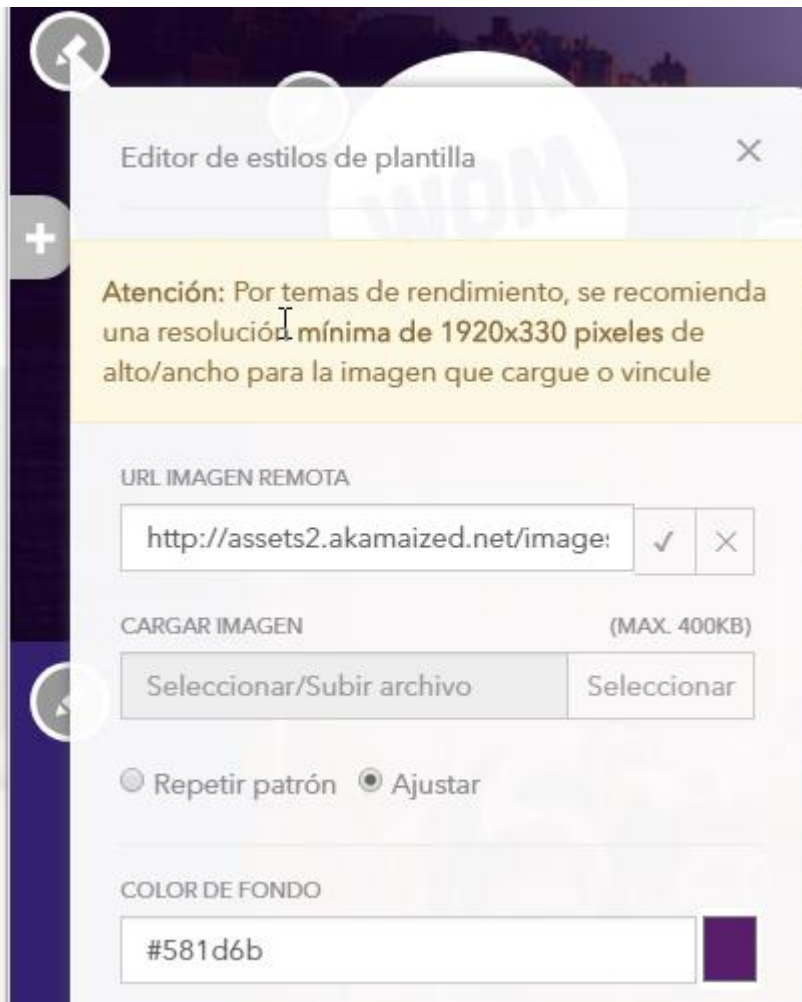
(`properties.edit.json`)

```

"cmp"      : "id",
"val"      : "bc_headerWrapper",
"options"  : ["bgurl", "bgcolor"],
"msg"      : "mínima de 1920x330 pixeles"

```

Demo



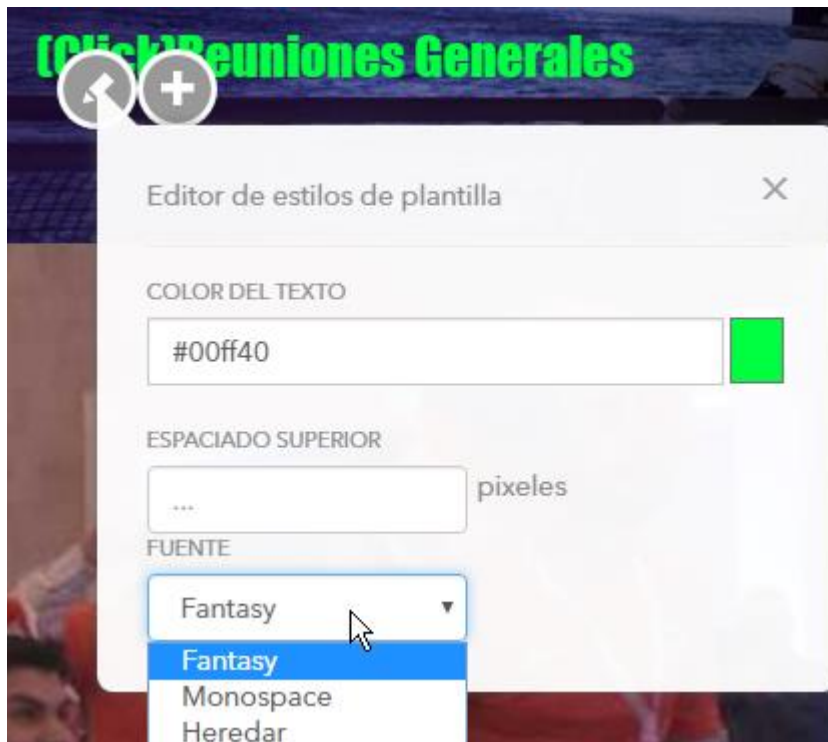
Ejemplo de extensión **EXTRA** de tipo **EXISTENTE**:
([properties.edit.json](#))


```

,{
  "cmp"      : "id",
  "val"      : "bc_clientTitle",
  "options"  : ["color", "extras"],
  "extras"   : [
    { ...
    },
    {
      "description" : "Fuente",
      "klass"       : "font-family",
      "tvalue"      : "combo",
      "extension"   : "",
      "msg"         : "",
      "options"     : [
        {
          "name" : "Fantasy",
          "value" : "fantasy"
        },
        {
          "name" : "Monospace",
          "value" : "monospace"
        },
        {
          "name" : "Heredar",
          "value" : "inherit"
        }
      ]
    }
  ]
}

```

Demo



Ejemplo de extensión EXTRA de tipo **NO EXISTENTE** (En este caso **number** no existe como **tipo**):

([properties.edit.json](#))

```
"extras" : [
  {
    "description" : "Espaciado Superior",
    "klass"       : "padding-top",
    "tvalue"      : "number",
    "extension"   : "px",
    "msg"         : "píxeles"
  },
]
```

([template/xy/config/properties.jslib.extension.js](#))

Se crea el elemento nuevo de edición:

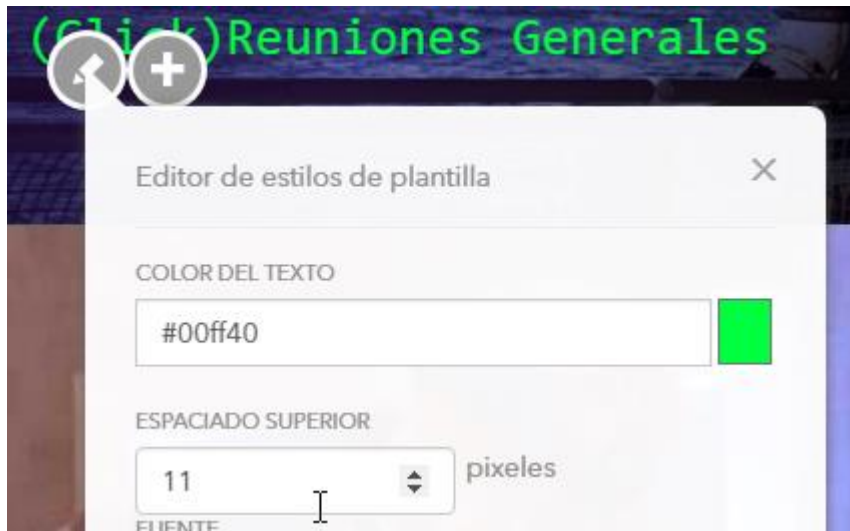
```

/**
 *
 * @ngdoc overview
 * @name gallery.extra
 *
 * @requires window.eGBlprocess
 *
 * @description
 *
 * eGBlprocess es la clase principal de extensión.
 * ## Extension de elementos extras de edición
 *
 * - keytext: Es la clave del elemento extra expuesto en el archivo properties.edit.json
 * - process: es la función callback que se llamará al procesar el elemento personalizado clave
 *
 */
eGBlprocess.addInput({
  keytext: 'number',
  process: function(extobj, refElement, customId){
    var strContent = "";
    var cValDyn = eGBlprocess.getValue(refElement, extobj.klass, extobj.extension);
    var idcminterno = eGBlprocess.genId(extobj.klass, customId);

    strContent += '<label >'+extobj.description+'</label>';
    strContent += '<input type="number" style="float:left;float: left;width: 50%;" id="'+idcmintern
    return strContent;
  }
});

```

Demo:



4.9 Extensión de librerías y uso de `app.options.custom.js`

Para agregar librerías y usarlas tenemos el ejemplo de un toaster de angularjs, modulo que requiere un js y un css para funcionar.

Agregamos Librerías:

```
{} properties.json x
1  {
2    "custom_collection" : false,
3    "libs": [
4      {
5        "id": "$1",
6        "src": "https://maps.googleapis.com/maps/api/js?libraries=places",
7        "async": false
8      },
9      {
10       "ref": "$1",
11       "src": "libs/vendor/google-maps/ng-map.min.js",
12       "async": false,
13       "module": "ngMap"
14     },
15     {
16       "src": "template/general/assets/toaster.min.js",
17       "async": false,
18       "module": "toaster"
19     }
20   ],
21   "max_visible_optionsmenu": 7,
22   "css_files" : ["assets/style.css", "assets/toaster.min.css"],
23   "js_files" : ["assets/app.js"]
24 }
```

Agregamos el HTML que nos pide la librería

```
home.html x
1  <div class="homeView appear" >
2    <update-title title="{{siteData.descripcion}}"></update-title>
3    <toaster-container></toaster-container>
```

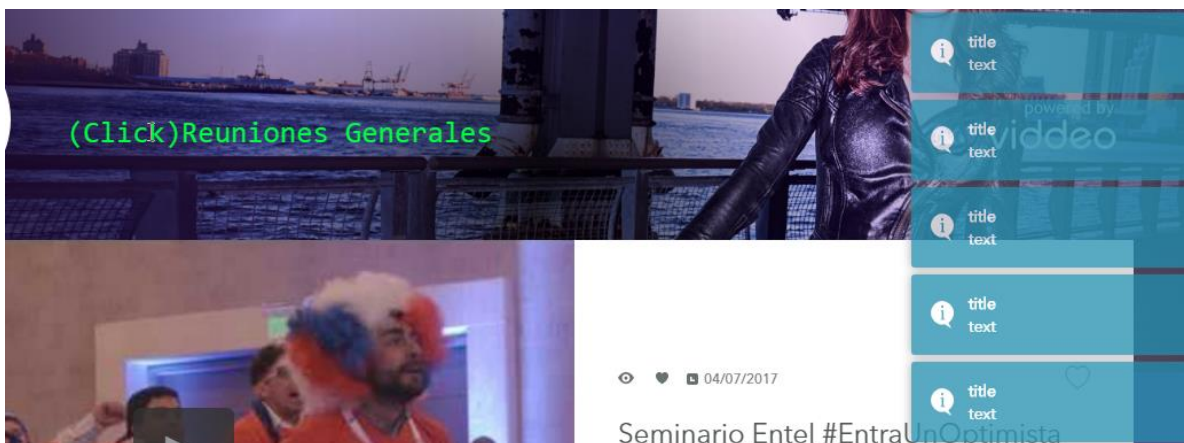
Agregamos un evento click para probar

```
<div class="logo" id="bc_clientLogo" style="width:50%; height:50px; border: 1px solid black; margin-bottom: 10px;">
```

Definimos la función en nuestro archivo de extensión de controladores. (app.options.custom.js)

```
JS app.options.custom.js x
1
2 angular.module("gallery.options.custom")
3 .run(['$rootScope', function($rootScope) { ...
6   }])
7 .controller('homeInitCtrl.custom', ['$rootScope', '$scope', 'toaster',
8   function($rootScope, $scope, toaster) {
9     $scope.custom = {};
10
11     $scope.custom.pop = function(){
12       console.log(toaster)
13       toaster.pop('info', "title", "text");
14     };
15
16   }])
17
```

Demo:



En resumen, podemos extender más módulos, controladores, fases RUN y CONFIG, rutas, y todo lo que tenga que ver con angular a partir de este módulo custom. También podemos crear o llamar nuevos módulos e inyectarlos como librerías específicas.

4.10 Creando Plantilla BASE desde línea de comandos con GULP

Existe una línea de comando para crear una plantilla con los elementos bases,

:GULP CREATE_TEMPLATE --tfolder nombre_plantilla

```
Toshiba@Toshiba-PC MINGW64 /c/AppServ/www/ZGroup/nuevo_gallery (master)
$ gulp create_template --tfolder otra
[20:09:24] Using gulpfile C:\AppServ\www\ZGroup\nuevo_gallery\gulpfile.js
[20:09:24] Starting 'cmmake'...
[20:09:24] Starting 'cp_template'...
otra
carpeta creada(./template/otra)...
```

La idea al crear una plantilla nueva es tener elementos básicos para no copiar o comenzar desde cero, también ir teniendo un orden.

Si se quiere modificar la PLANTILLA BASE se debe ir a la carpeta

\template\common\._genout*

Lo que se modifique acá será de base para las futuras plantillas, con esto aceleramos el proceso de creación de plantillas.