Contenido

[Gestión de código con GIT 3](#_Toc8230499)

[Clonando un proyecto con submódulos 3](#_Toc8230500)

[Pasos para clonar el proyecto tierra3 3](#_Toc8230501)

[Gestión de código en el submódulo 3](#_Toc8230502)

[Subida de versiones en el submódulo 5](#_Toc8230503)

[Entorno de desarrollo. 5](#_Toc8230504)

[Extensiones necesarias 5](#_Toc8230505)

[Extensiones recomendadas 6](#_Toc8230506)

[Librerías principales añadidas al proyecto 6](#_Toc8230507)

[PrimeNG 6](#_Toc8230508)

[FlexGrid 6](#_Toc8230509)

[Openlayers 6](#_Toc8230510)

[Proj4js 6](#_Toc8230511)

[Font Awesome 6](#_Toc8230512)

[Scripts 6](#_Toc8230513)

[Development server 6](#_Toc8230514)

[Code scaffolding 6](#_Toc8230515)

[Construir 7](#_Toc8230516)

[Documentación 7](#_Toc8230517)

[Ayuda adicional 7](#_Toc8230518)

[Módulo Tema 7](#_Toc8230519)

[Layouts 7](#_Toc8230520)

[the-visor-layout 7](#_Toc8230521)

[Componentes 7](#_Toc8230522)

[Button 7](#_Toc8230523)

[Input Group 8](#_Toc8230524)

[Tab Menu 9](#_Toc8230525)

[Side Panel 10](#_Toc8230526)

[Full Panel 11](#_Toc8230527)

[Multipanel 12](#_Toc8230528)

[Card 13](#_Toc8230529)

[Fieldset 14](#_Toc8230530)

[Servicios del tema 14](#_Toc8230531)

[Colores y eventos del tema 14](#_Toc8230532)

[Control de errores 15](#_Toc8230533)

[Idiomas 16](#_Toc8230534)

[Logs 18](#_Toc8230535)

[Configuración 18](#_Toc8230536)

[Uso 18](#_Toc8230537)

[Toast 18](#_Toc8230538)

[Configuración 18](#_Toc8230539)

[Uso 19](#_Toc8230540)

[Servicio de paneles 19](#_Toc8230541)

Proyecto Tierra3

Proyecto generado con [Angular CLI](https://github.com/angular/angular-cli) version 7.3.6 usando la versión ~7.2.0 de Angular y ~6.3.3 de RxJS

# Gestión de código con GIT

El proyecto se encuentra alojado en un repositorio GIT en la siguiente dirección:

* Acceso desde red interna Cotesa: ssh://git.grupotecopy.es:22/data/git/tierra3/visor.git
* Acceso externo: ssh://git.grupotecopy.es:6007/data/git/tierra3/visor.git

Para el desarrollo por parte de dos equipos de sus funcionalidades y en concreto del módulo de usuarios se ha incluido como submódulo del proyecto el repositorio git donde se desarrollará este submódulo.

El submódulo de usuarios se encuentra en el siguiente repositorio GIT:

* Acceso desde red interna Cotesa: ssh://git.grupotecopy.es:22/data/git/tierra3/visor-users-altia.git
* Acceso externo: ssh://git.grupotecopy.es:6007/data/git/tierra3/visor-users-altia.git

Se ha hecho *checkout* de este repositorio en el proyecto del visor en la carpeta

src\app\modules\users de esta forma esta carpeta se comportará como un repositorio git independiente permitiendo hacer sus propios commits y push.

A medida que se vaya avanzando con el módulo se deberán realizar subidas a la rama master con un tag que indique la versión del módulo que servirá para ir fijando las versiones en el visor.

## Clonando un proyecto con submódulos

Si se tiene un proyecto con submódulos dentro de él. Cuando se recibe, se reciben también las carpetas que contienen los submódulos; pero no se reciben ninguno de los archivos de dichos submódulos.

Son necesarios otros dos comandos:

* **git submodule init** para inicializar el archivo de configuración local
* **git submodule update** para recuperar (fetch) todos los datos del proyecto y extraer (checkout) la confirmación de cambios adecuada desde el proyecto padre.

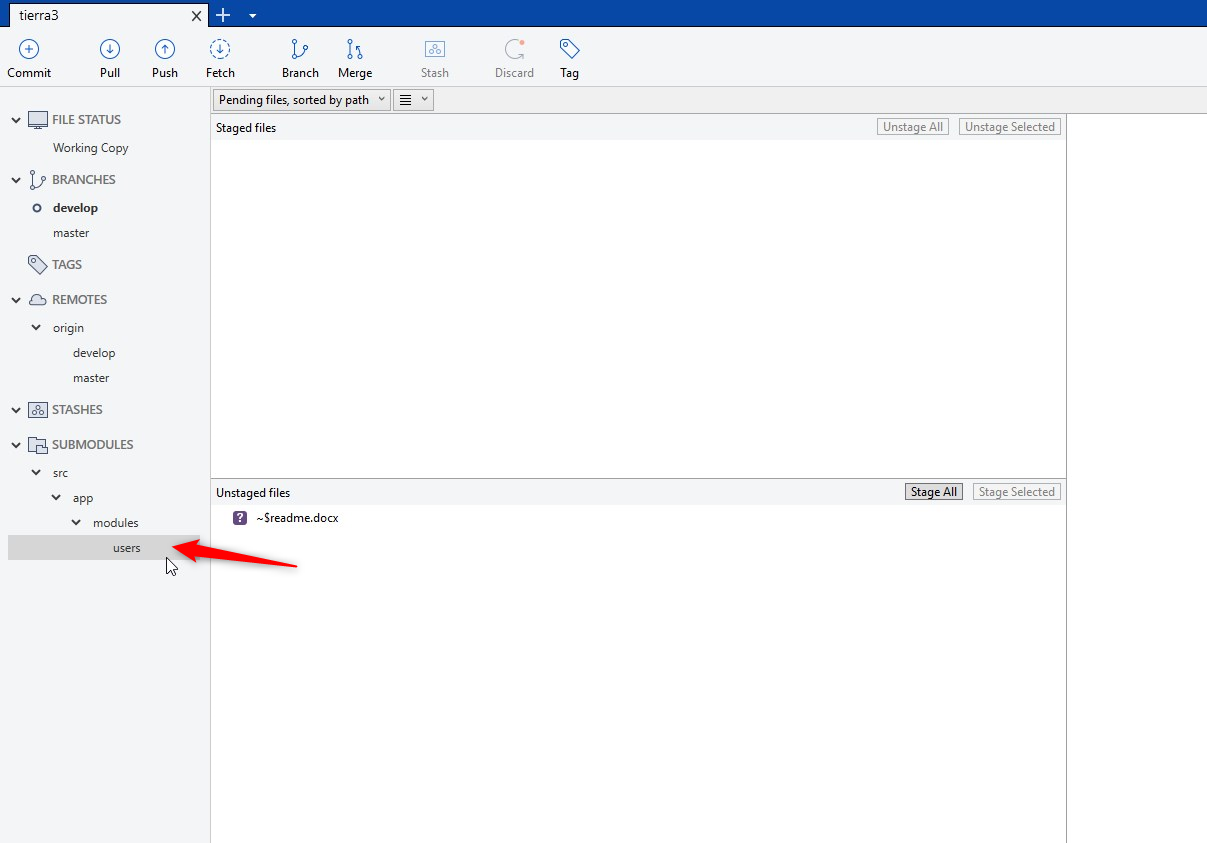
## Pasos para clonar el proyecto tierra3

1. **git clone ssh://git.grupotecopy.es:XXXX/data/git/tierra3/visor.git prueba-tierra3** - XXXX será el puerto indicado en el punto 1)
2. **git checkout develop** – Pasamos a la rama develop
3. **git submodule init** – Inicialización de submódulos.
4. **git submodule update** – Actualización del cambios del submódulo.

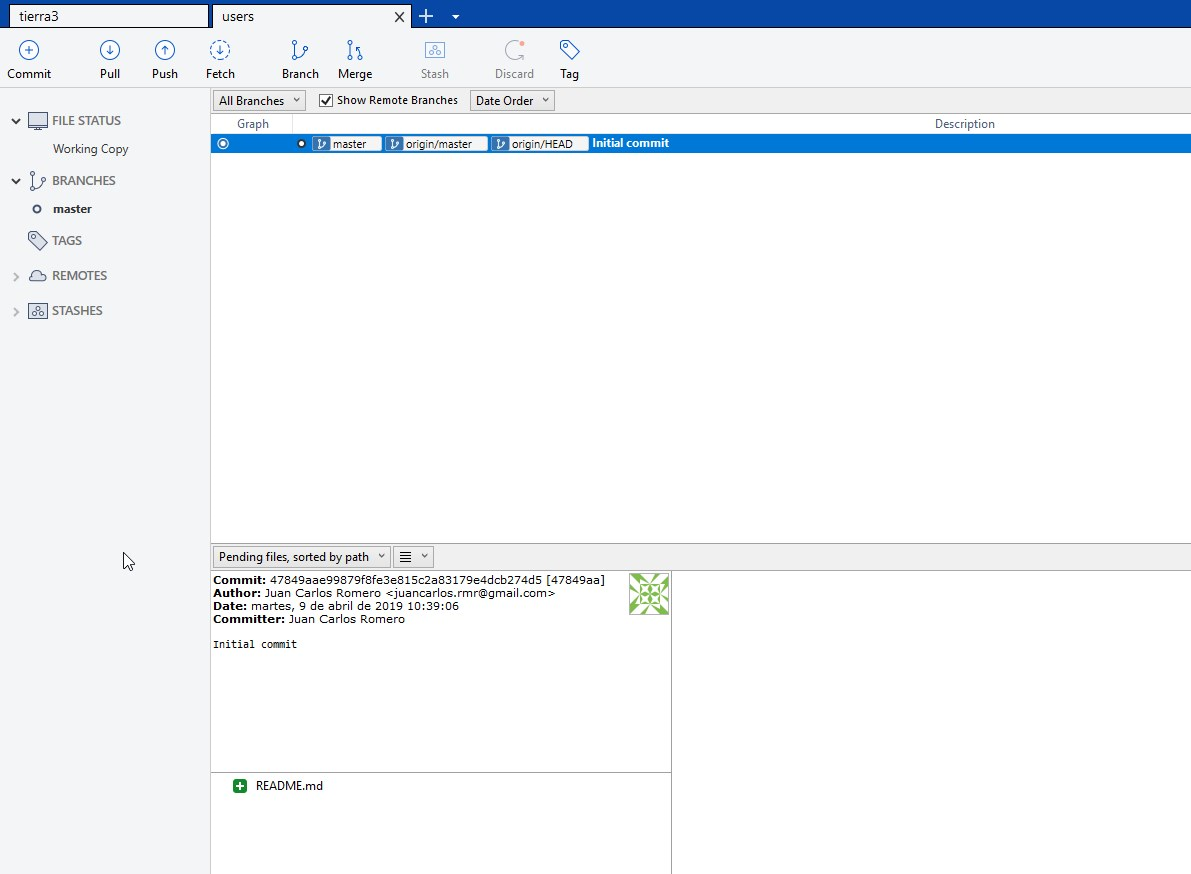
## Gestión de código en el submódulo

La carpeta donde se aloja el submódulo se comportará como un repositorio GIT independiente por lo que ubicándonos con nuestro intérprete de comandos en dicha carpeta se podrá realizar cualquier acción con línea de comandos.

Si usamos un cliente GIT como Sourcetree tendremos la posibilidad de abrir el submódulo desde el proyecto del visor.



Haciendo doble clic sobre el submódulo accederemos al GIT del submódulo pudiendo realizar cualquier acción permitida.



De igual forma podemos abrir con nuestro cliente el repositorio del submódulo abriendo directamente su ubicación desde Sourcetree.

## Subida de versiones en el submódulo

Como comentamos, para subir una nueva versión del módulo se realizará una subida a la rama master con el tag de la versión y se deberá comunicar mediante correo a todo el equipo de desarrollo junto con los cambios realizados.

Seguidamente se realizará por parte de Cotesa el *checkout* del tag de la versión deseada en el visor quedando fijado en el proyecto el código del módulo en la versión deseada.

Esto permite que Altia pueda continuar el desarrollo del módulo sin afectar al visor global.

# Entorno de desarrollo.

El IDE elegido para el desarrollo es Visual Studio Code (<https://code.visualstudio.com/>).

## Extensiones necesarias

**Beautify (**hookyqr.beautify**)**

Para mantener la homogeneidad en el formato del código se incluido una configuración en el proyecto que usa la extensión Beautify.

**Document This** (joelday.docthis)

Extensión que facilita la documentación interna de la aplicación. El código generado deberá tener una documentación que facilite su uso. Esta documentación interna posteriormente autogenerará la documentación externa.

## Extensiones recomendadas

Angular Language Service (hookyqr.beautify)

**Debugger for Chrome** (msjsdiag.debugger-for-chrome)

# Librerías principales añadidas al proyecto

## [PrimeNG](https://www.primefaces.org/primeng/#/)

PrimeNG es una colección de componentes de interfaz de usuario para Angular. Todos los widgets son de código abierto y de uso gratuito bajo la licencia MIT. PrimeNG es desarrollado por PrimeTek Informatics, un proveedor con años de experiencia en el desarrollo de soluciones UI de código abierto.

## [FlexGrid](https://www.primefaces.org/primeng/#/flexgrid)

Flex Grid CSS es una utilidad ligera de diseño flexible basada en flex, optimizada para teléfonos móviles, tabletas y computadoras de escritorio. Flex Grid CSS no se incluye en PrimeNG, ya que lo proporciona PrimeFlex, es una biblioteca de grid compartida entre los proyectos PrimeFaces, PrimeNG y PrimeReact.

## [Openlayers](https://openlayers.org/en/v4.6.5/apidoc/)

OpenLayers facilita colocar un mapa dinámico en cualquier página web. Puede mostrar mosaicos de mapas, datos vectoriales y marcadores cargados desde cualquier fuente. OpenLayers ha sido desarrollado para promover el uso de información geográfica de todo tipo. Es completamente gratuito, Open Source JavaScript, publicado bajo la licencia BSD de 2 cláusulas (también conocida como FreeBSD).

## [Proj4js](https://github.com/proj4js/proj4js)

Proj4js es una biblioteca de JavaScript para transformar coordenadas de puntos de un sistema de coordenadas a otro, incluidas las transformaciones de referencia.

## Font Awesome

Version 5 de Font Awesome para el uso de iconos además de los disponibles en la librería PRIMENG

<https://fontawesome.com/icons?d=gallery&m=free>

# Scripts

## Development server

Ejecutar ng serve para levantar un servidor de desarrollo.  
Navegar a http://localhost:4200/. La aplicación se volverá a cargar automáticamente si cambia alguno de los archivos de origen.

## Code scaffolding

Ejecute ng generate component component-name para generar un nuevo componente. También puede usar ng generate directive|pipe|service|class|guard|interface|enum|module.

## Construir

Ejecute ng build para construir el proyecto. Los artefactos de compilación se almacenarán en el directorio dist/. Use la bandera --prod para una construcción de producción.

## Documentación

Ejecute npm run docs:serve para levantar un servidor con la documentación de todos los componentes incluidos en el proyecto.

## Ayuda adicional

Para obtener más ayuda sobre Angular CLI, use ng help o visite el [Angular CLI README] (<https://github.com/angular/angular-cli/blob/master/README.md>).

# Módulo Tema

Para su uso con la aplicación se ha desarrollado un tema que está basado en los componentes de PrimeNg pero que encapsula aquellos componentes necesarios para dotarlos de un funcionamiento y un aspecto homogéneo en la aplicación.  
Por este motivo **no se deben importar componentes de PrimeNg que no sean exportados (si se da el caso) a través del módulo ThemeModule**.

Los componentes del módulo del tema tendrán selectores con el prefijo **the-**

## Layouts

Componentes que definen el layout de la página

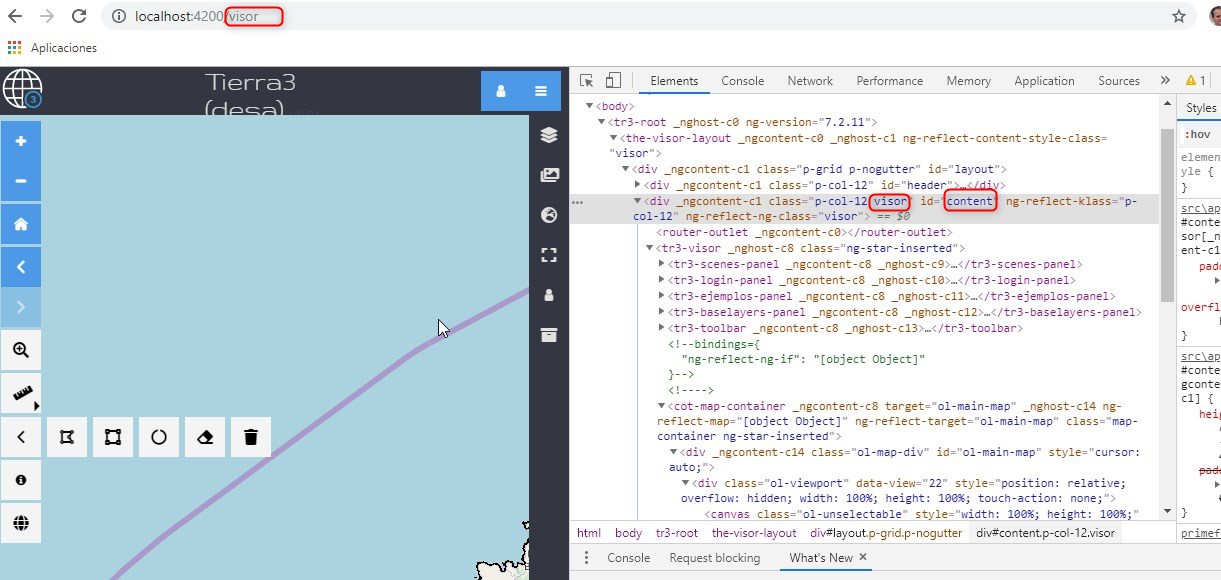
### the-visor-layout

Layout definido para el visor, se compone de un header y el resto es ocupado por el componente que cargue el router

Uso  
En el app.component.html hará que todas las páginas tengan ese template

<the-visor-layout [contentStyleClass]="contentStyleClass">  
<router-outlet></router-outlet>  
</the-visor-layout>

Mediante el parámetro contentStyleClass que se establece en app.componetn.ts, se define en el contenedor del layout con la clase correspondiente a la página cargada en la url. Esto permitirá definir estilos específicos para un mismo layout pero en distintas páginas



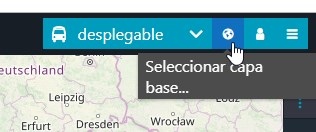
Si se desea el uso de diferentes layouts en diferentes páginas sería necesario eliminar el layout de app.component.html y aplicarlo a cada página, quedando el app.component.html sólo con el router-outlet:

<router-outlet></router-outlet>

## Componentes

### Button

Componente que genera un botón.



#### Uso

<the-button tooltip="tooltip" icon="pi pi-user" class="ui-button-succes"

(onClick)="openPanel('login-panel')">

</the-button>

#### Parámetros

|  |  |  |  |
| --- | --- | --- | --- |
| Parámetro | Tipo | Valor por defecto |  |
| icon | string |  | Icono a mostrar, Soporta iconos de PrimeIcons y FontAweson 5 |
| label | string | Cadena vacía | Etiqueta del botón |
| tooltipPosition | string | top | Texto de ayuda del botón |
| styleClass | string | ui-button-succes | Clase del botón, se puden usar los predefinidos:   * .ui-button-secondary * .ui-button-success * .ui-button-info * .ui-button-warning * .ui-button-danger |
| disabled | boolean | false | Deshabilita el botón |
| showSpinner | boolean | false | Muestra un spiner como icono del botón. Mientras el spinner está activo, el botón está deshabilitado |

#### Eventos

|  |  |  |
| --- | --- | --- |
| Evento | Tipo |  |
| onClick | click | Evento que se lanza al pulsar sobre el botón |

### Input Group

Componente que genera un input con un botón a su derecha.



#### Uso

<the-input-group required="true" [(value)]="url" placeholder="{{'WMS\_LOADER.URL\_WMS' | translate}}"

iconButton="fas fa-search" [pattern]="pattern" (clickButton)= "getDataFromReaderServices()"></the-input-group>

#### Parámetros

|  |  |  |  |
| --- | --- | --- | --- |
| Parámetro | Tipo | Valor por defecto |  |
| value | string | Cadena vacía | Valor a mostrar en el input. Es bidireccional definiéndolo como [(value)] |
| placeholder | string | Cadena vacía | Etiqueta interior del input |
| pattern | string |  | Patrón que debe cumplir el input. Si no lo cumple, deshabilita el botón de la derecha. |
| type | string | ‘text’ | Tipo de input (‘text’, ‘number’ …) |
| required | boolean | false | Indica si el input es requerido |
| labelButton | string | Cadena vacía | Etiqueta del botón |
| iconButton | string |  | Icono a mostrar, Soporta iconos de PrimeIcons y FontAweson 5 |
| tooltip |  |  | Tooltip a mostrar en el botón |
| tooltipPosition |  |  | Posición del tooltip respecto a el botón |
| showSpinner | boolean | false | Muestra el spinner en el botón eliminado el icono. Mientras está activo el spinner, el botón estará inactivo |

#### Eventos

|  |  |  |
| --- | --- | --- |
| Evento | Tipo |  |
| clickButton | click | Evento que se lanza al pulsar sobre el botón |

### Tab Menu

Componente que genera un menú de pestañas de selección y un contenedor para albergar el contenido de cada pestaña.

|  |  |
| --- | --- |
|  |  |

#### Uso

<the-tab-menu [items]="tabs" [activeItem]="activeTab" styleClass='secondary'>

<div tabs-content>

<cot-wms-loader \*ngIf="activeTab.id == 'externas'"></cot-wms-loader>

<cot-local-layer-loader \*ngIf="activeTab.id == 'locales'"></cot-local-layer-loader>

</div>

</the-tab-menu>

#### Parámetros

|  |  |  |  |
| --- | --- | --- | --- |
| Parámetro | Tipo | Valor por defecto |  |
| items | MenuItem[] | [] | Array con los objetos MenuItem que formarán cada una de las pestañas |
| activeItem | MenuItem |  | La pestaña activa |
| styleClass | string | primary | Clase del contendor de menú, los valores posibles serán ‘primary’ o ‘secondary’ |

En el contenedor de tabs se proyectará el elemento recibido con el atributo tabs-content.

Por ejemplo en el siguiente código:

<div tabs-content>

<cot-wms-loader \*ngIf="activeTab.id == 'externas'"></cot-wms-loader>

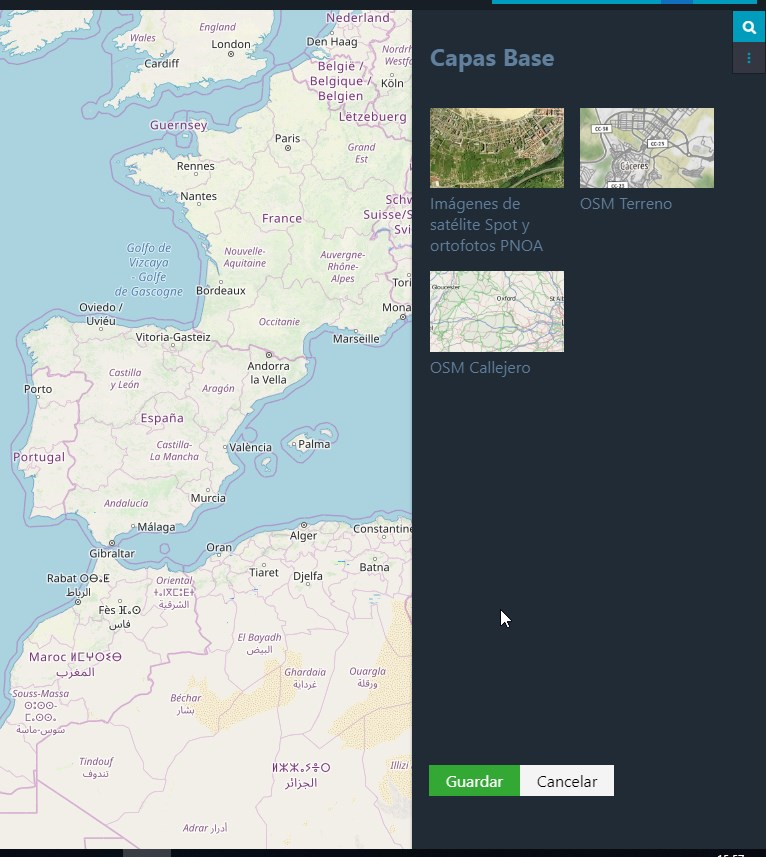
<cot-local-layer-loader \*ngIf="activeTab.id == 'locales'"></cot-local-layer-loader>

</div>

Se define el contenido de los tabs en un div con el atributo tabs-content. El contenido de los tabs deberá gestionar la visualización de un contenido u otro dependiendo del estado del tab activo del menú.

### Side Panel

Panel que abre desde uno de los cuatro laterales de la pantalla.



#### Uso

<the-side-panel [panelName]='panelName' position='right'>

<cot-base-layers panel-content></cot-base-layers>

<div panel-footer>

<button pButton type="button" label="{{'BUTTON.SAVE'|translate}}" class="ui-button-success"></button>

<button pButton type="button" label="{{'BUTTON.CANCEL'|translate}}" class="ui-button-secondary"></button>

</div>

</the-side-panel>

El panel dispone de dos contenedores en los que se proyectará el elemento con el atributo panel-content y el elemento con el atributo panel-footer.

El contenedor panel-content dispone de scroll para poder visualizar elementos de tamaño mayor.

El contenedor panel-footer se fija en la parte inferior del componente.

#### Parámetros

|  |  |  |  |
| --- | --- | --- | --- |
| Parámetro | Tipo | Valor por defecto |  |
| panelName | string |  | Nombre del panel para su gestión por el servicio de paneles |

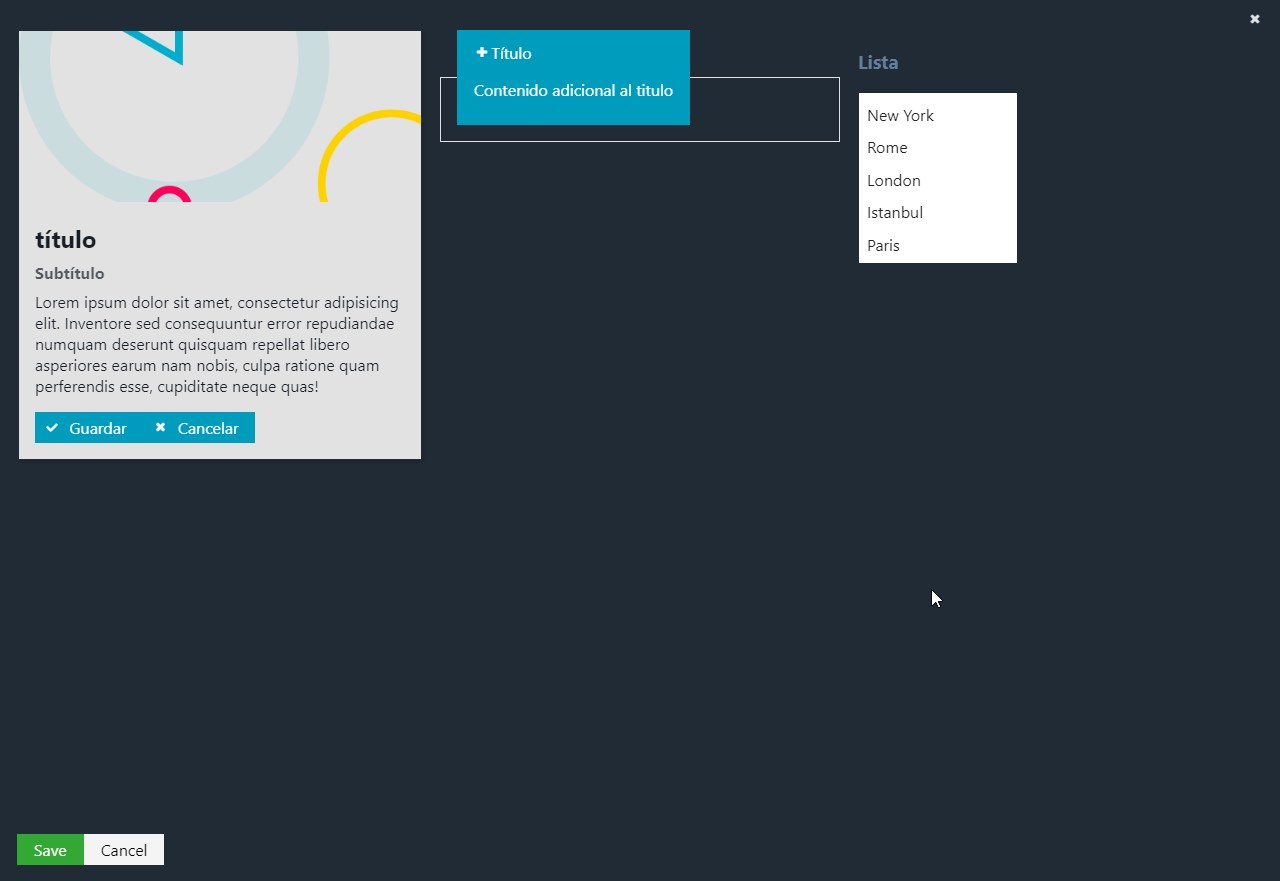
Mediante el servicio de gestión de paneles será posible abrir o cerrar o cambiar de estado un panel que previamente haya sido registrado en él y se haya incluído en la página visualizada.

Por ejemplo:

this.panelsManagerService.openPanel(nombrePanel)

### Full Panel

Panel que abre a pantalla completa



#### *Uso*

<the-full-panel [panelName]='panelName'>

<div panel-content>

Contenido

</div>

<div panel-footer>

<button pButton type="button" label="Save" class="ui-button-success"></button>

<button pButton type="button" label="Cancel" class="ui-button-secondary"></button>

</div>

</the-full-panel>

#### Parámetros

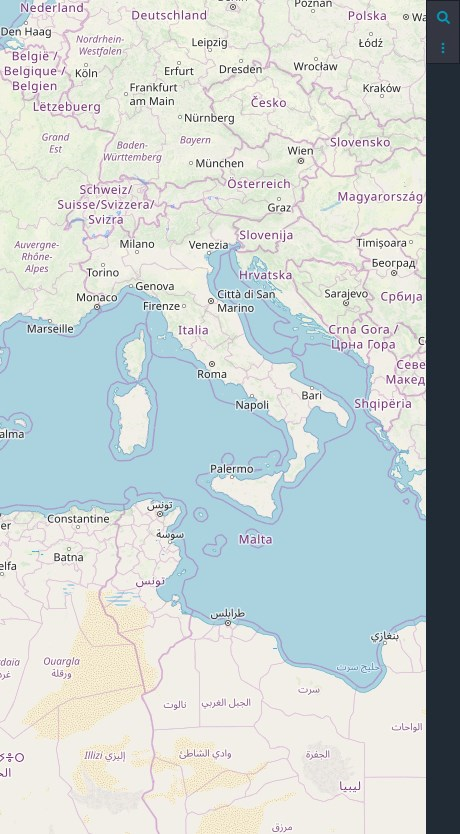
El uso del full panel es igual al del SIDE PANEL.

|  |  |  |  |
| --- | --- | --- | --- |
| Parámetro | Tipo | Valor por defecto |  |
| panelName | string |  | Nombre del panel para su gestión por el servicio de paneles |

Mediante el servicio de gestión de paneles será posible abrir o cerrar un panel que previamente haya sido registrado en él y se haya incluído en la página visualizada.

this.panelsManagerService.openPanel (nombrePanel)

### Multipanel



Es un componente que permite cargar paneles desde una barra lateral que contendrá un botón toggle por cada panel registrado en el componente

#### *Uso*

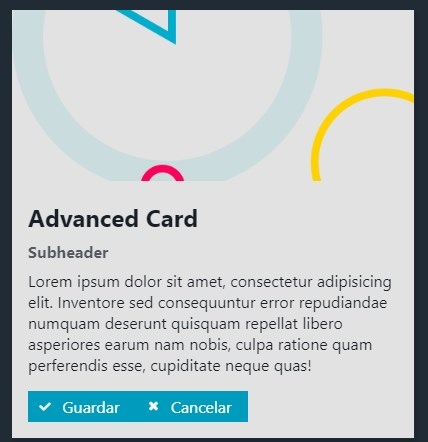
<the-multipanel [panelsIcons]="['pi pi-search','pi pi-ellipsis-v']" [panelsNames]="['base-layers','login-panel']">

</the-multipanel>

#### Parámetros

|  |  |  |  |
| --- | --- | --- | --- |
| Parámetro | Tipo | Valor por defecto |  |
| panelsIcons | string[] | [] | Array de iconos que se usarán para mostrar en el botón que activa cada panel. Los iconos se mostrarán en el orden en el que se definan y se asignarán a cada panel en el mismo orden.  Soporta iconos de PrimeIcons y FontAweson 4.7 |
| panelsNames | string[] | [] | Nombre de los paneles a mostrar desde el multipanel. Para cada panel se genera un botón que abre y cierra el panel |

### Card



Es un componente que permite mostrar contenido en forma de card.

#### Uso

<the-card title="título" subtitle="Subtítulo">

<div card-header>

<img src="Card" src="assets/images/examples/usercard.png">

</div>

Lorem ipsum dolor sit amet,

<div card-footer>

<the-button type="button" label="Guardar" icon="pi pi-check"></the-button>

<the-button type="button" label="Cancelar" icon="pi pi-times"></the-button>

</div>

</the-card>

El componente card dispone de dos contenedores en los que se proyectará el elemento con el atributo card-header y el elemento con el atributo card-footer.

El contenedor card-header será el la parte superior de la tarjeta.

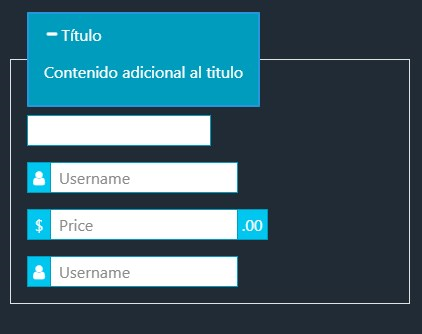
El contenedor card-footer mostrará el contenido en la parte inferior de la tarjeta.

El contenido incluido en el componente sin los anteriores atributos se mostrará en el cuerpo de la tarjeta.

#### Parámetros

|  |  |  |  |
| --- | --- | --- | --- |
| Parámetro | Tipo | Valor por defecto |  |
| title | string | null | Título a mostrar en el cuerpo de la tarjeta |
| subtitle | string | null | Subtítulo a mostrar en el cuerpo de la tarjeta |

### Fieldset



Es un componente que permite agrupar campos con la posibilidad de colapsar.

#### *Uso*

<the-fieldset [title]='"Título"'>

<p fieldset-header> Contenido adicional al título</p>

<div class="p-grid p-dir-col-rev">

Lorem ipsum dolor sit amet,

</div>

</the-fieldset>

El componente dispone de un contenedores en el que se proyectará el elemento con el atributo fieldset-header

El contenedor fieldset-header se mostrará baje el título del componente

#### Parámetros

|  |  |  |  |
| --- | --- | --- | --- |
| Parámetro | Tipo | Valor por defecto |  |
| title | string | null | Título a mostrar en el cuerpo de la tarjeta |
| collapsed | boolean | false | Indica si el grupo se muestra colapsado |

## Servicios del tema

El tema del visor expone diversos servicios que podrán ser utilizados por los módulos que le importe. La configuración que se realiza a nivel global no estará disponible para los módulos.

### Colores y eventos del tema

El tema pone a disposición de los módulos un servicio que informará del estado de algunos componentes del tema y los colores del mismo.

#### Servicio ThemeService

Este servicio proporciona acceso a las siguientes propiedades y métodos:

|  |  |
| --- | --- |
| navigatingBetweenPages$: Observable<boolean>; | Observable que emite un booleano cada vez que el router inicia la navegación de una página a otra y cuando esta finaliza. Observa los eventos del Router de Angular. |
| panelOpen$: Observable<boolean>; | Observable que emitirá un nuevo valor cuando un panel manejado por el servicio de paneles se abra o se cierre. |
| notifyPanelOpen(open: boolean) | Método que permite a otros componentes notificar la apertura de un panel |
| getThemeColor(color: string) | Obtiene un color del tema, el nombre del color deberá ser alguno de los soportados por el tema:  'primaryFg', 'primaryBg', 'secondaryBg', 'secondaryFg', 'tertiaryBg', 'tertiaryFg' |
| setThemeColor(color: string, colorValue: string) | Establece un color en el tema. El establecimiento de un color no se aplicará a los componentes si estos no han utilizado el color como var(--color)  Siendo color alguno de los indicados en el anterior punto. |

### Control de errores

#### Clases

Para el control de errores se define una jerarquía de clases de error que nos permitirán tener un control sobre el tipo de error generado. Todas las clases de error deberán extender de la clase ErrorBase.

Esta clase implementa la interfaz Error lo que nos permitirá mantener el stack del error generado

export class ErrorBase implements Error {

name: string;

title: string;

message: string;

stack?: string; *// Stack del error*

callback?: () => void;

}

##### Parámetros opcionales

**stack** :Cuando usamos una clase de error para relanzar un error recibido, podemos incluir el stack del error original.

**callback**: función a ejecutar cada vez que se produzca un error.

##### Ejemplo

Como ejemplo de clase de error definida tenemos la clase ErrorTheme que usamos para lanzar errores en el tema.

export class ErrorTheme extends ErrorBase {

constructor(errorSource: string, message: string) {

super(errorSource, message, 'ErrorTheme');

}

}

Por ejemplo para lanzar un error de este tipo cuando se intenta abrir un panel que no existe:

openSideBar(sidebarName: string) {

if (!this.sidebars[sidebarName]) {

throw new ErrorTheme("PanelsManagerService", `Panel ${sidebarName} no registrado`);

} else this.sidebars[sidebarName].visible = true;

}

Lanzamos una nueva instancia de esta clase con los parámetros adecuados, esta excepción será capturada por nuestro servicio de errores FrontendErrorManagerService que realizará la gestión del error necesaria. Actualmente muestra un toast por pantalla con la descripción del error.

En nuestro servicio FrontendErrorManagerService podremos identificar el tipo de error generado. Todo error, mostrará un toast de error con el título y el mensaje del error

Tras mostrar el error si el objeto error contiene un atributo callback, se ejecutará la llamada.

if (error.callback) { error.callback() }

### Idiomas

Los literales de los componentes serán gestionados por el módulo ngx-translate. Esto nos permite tener centralizados los literales de los componentes, tener la posibilidad de definir diferentes idiomas y cambiar los literales de la aplicación en tiempo de ejecución.

#### Configuración

A través de las variables de entorno de enviroment.ts es posible configurar la ubicación del archivo de idiomas

language\_json: '/assets/i18n/',

language\_json: Carpeta con la ubicación del archivo de idiomas.

Los archivos de idioma dependerán del código de idioma:

* es.json – Castellano
* en.json – Ingles
* …

#### Uso

##### Definición de literales

Los literales son definidos en la carpeta src\assets\i18n. En el caso del idioma castellano, se definen en el archivo json es.json.

Para la definición de literales se usará la norma COMPONENTE.LITERAL donde COMPONTENTE identificará al componente para el que se definen y literal el identificador.

Tendremos algunos valores para componentes reservados como

|  |  |
| --- | --- |
| GLOBAL | Literales comunes de uso global. Evitarán la repetición de literales en otros componentes |
| ERRORS | Literales de errores genéricos |

{

"GLOBAL.FILTER": "Filtro",

"GLOBAL.ID": "ID",

"GLOBAL.NEW": "Nuevo",

"GLOBAL.PRINT": "Imprimir",

"ERRORS.ERROR": "Error",

"ERRORS.LOADING": "Se ha producido un error al cargar los datos... Por favor inténtelo de nuevo",

"ERRORS.SAVING": "Se ha producido un error al guardar los datos... Por favor inténtelo de nuevo",

}

##### Uso en componentes

###### Uso en template

Para su uso en los templates del componentes se utilizará el pipe translate que lo aplicaremos sobre la clave del literal que queremos usar. Esto mostrará el texto correspondiente a la clave y en el caso de cambio de idioma en la aplicación, el cambio se realizará dinámicamente cargando y mostrando los literales del nuevo idioma.

<div class="p-col-12">

<h2>{{"BASE\_LAYERS.TITLE" | translate }}</h2>

</div>

###### Uso en componente

Para su uso dentro de un componente debemos extender el componente del tema ThemeBaseComponentClass. Esta clase base implementa la gestión de los literales y la subscripción al servicio para un conjunto de literales lo que proporciona soporte para el cambio de idiomas. De esta forma si en el servicio de traducción se establece otro idioma, los literales se actualizarán automáticamente.

Mediante el método **useLiterals** que implementa **ThemeBaseComponentClass** podemos pasar un array de los literales que queremos usar dentro del componente.

export class BaselayersPanelComponent extends ThemeBaseComponentClass {

panelName = 'Capas Base';

constructor() {

super();

this.useLiterals(['BASE\_LAYERS.TITLE', 'TOC.TITLE']);

}

Posteriormente podremos usar estos literales como propiedades del objeto componentLiterals que declara la clase padre accediendo a la propiedad

alert(this.componentLiterals['BASE\_LAYERS.TITLE'])

En el caso de necesitar conocer **cuándo se ha producido un cambio en los literales** para realizar reasignaciones u otra tarea en el componente, se emite el evento **onComponentLiteralsChange** cada vez que el objeto **componentLiterals** cambia por cambio de idioma.

## Logs

El servicio de log está proporcionado por el servicio LoggerService, no se podrán usar el objeto console para generar logs.

### Configuración

A través de las variables de entorno de enviroment.ts es posible configurar el nivel de log para consola y para un posible servidor de logs.

logging: {

//Nivel de error a mostrar por consola

consoleLogLevel: NgxLoggerLevel.DEBUG,

//Configuracion opcional de servidor de logs, si e configura, cambiar el nivel de serverErrorLevel

serverLogUrl: 'http://',

serverLogLevel: NgxLoggerLevel.OFF,

}

* consoleLogLevel : define el nivel mínimo de severidad de los errores a mostrar. Los posibles valores son:
* TRACE = 0,
* DEBUG = 1,
* INFO = 2,
* LOG = 3,
* WARN = 4,
* ERROR = 5,
* FATAL = 6,
* OFF = 7,
* serverLogUrl: Url de un servidor de logs opcional
* serverLogLevel: define el nivel mínimo de severidad de los errores a mandar al servidor

### Uso

El servicio dispone de varios métodos en función de la severidad del log a generar:

log(logText: string)

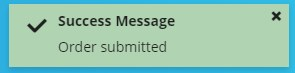
info(logText: string)

error(logText: string)

warn(logText: string)

## Toast

Para mostrar mensajes asíncronos al usuario se dispone del servicio ToastService.



### Configuración

A través de las variables de entorno de enviroment.ts es posible configurar la posición y duración de los mensajes:

toast: {

duration: 3000,

position: "bottom-center"

},

Duración: Duración del mensaje en milisegundos

Posición: algunas de las siguientes opciones:

* top-right
* top-left
* bottom-right
* bottom-left
* top-center
* bottom-center
* center

### Uso

Este servicio dispone varios métodos para mostrar un mensaje tipo Toast al usuario dependiendo de su severidad:

showError(message: ToastMessageInterface)

showSuccess(message: ToastMessageInterface)

showInfo(message: ToastMessageInterface)

showWarning(message: ToastMessageInterface)

clear()

Para emitir el mensaje se implementará en la llamada la interfaz ToastMessageInterface

export interface ToastMessageInterface {

summary: string;

detail: string;

}

En la que se definirá el texto detail y el texto summary

this.toast.showError({ summary: ‘titulo’, detail: ‘detalle del mensaje’ })

Todos los mensajes mostrados por pantalla se mostrarán por log en los entornos de desarrollo:

2019-04-08T12:28:59.594Z ERROR [main.js:887] PanelsManagerService - Panel bese-layers no registrado

## Servicio de paneles

Mediante el servicio de paneles podemos realizar diversas acciones sobre los paneles registrados.

Cuando usamos cualquier componente panel (sidePanel o fullPanel), automáticamente se registra con el nombre pasado como parámetro panelName en el servicio.

Una vez registrado podemos mediante el servicio realizar diversas acciones:

registerPanel(panelName: string)

registra una barra en el servicio para que pueda ser gestionada desde otros componentes, normalmente no será necesario usar este método pues la clase base ya registra el componente

getPanelItem(panelName: string)

Obtiene el item de control del panel en el servicio

openPanel(panelName: string)

Abre el panel por su nombre

setClassPanel(panelName: string, styleClass: string)

Establece una clase CSS específica al panel

setButtonClosePanel(panelName: string, show: boolean)

Establece si se muestra o no el botón cerrar del panel

closePanel(panelName: string)

Cierra el panel por su nombre

togglePanel(panelName: string)

Cambia el estado del panel abierto/cerrado