

Manual para generar contenido con el marco de trabajo basado en MDE

[Descargar estos apuntes](#)

Índice

▼ Pre-Requisitos

- Esquema de carpetas tras la instalación

▼ Acceso al Workspace

- Descripción del contenido del repositorio

▼ Trabajo básico con markdown

▼ Markdown básico

- Encabezados
- Párrafos
- Citas

▪ Listas

- Tablas simples
- Enlaces

▼ Imágenes

- Estándar imágenes básico
- Ampliación imágenes

▪ Expresiones matemáticas

▪ Bloques de código

▪ Admonitions

▪ Emojis de GitHub Markdown

▼ Diagramas

▪ Renderizado a través de Kroki

▪ Incluir diagramas desde un fichero externo

▼ Ejemplos con PlantUML

- Ejemplo contenido carpetas (wireframe puml)
- Ejemplo arquitectura (C4 puml)
- Ejemplo mapa mental
- Ejemplo diagrama de flujo

▼ Ejemplos con Mermaid

- Ejemplo arquitectura (mermaid)

▼ Maquetación compleja basada en Bootstrap 5

▪ Maquetación a 2 o 3 columnas

▪ Generar PDF

▼ Conclusiones

▪ Vantajas

▪ Desventajas

Pre-Requisitos

1. Tener una **cuenta de GitHub**.
2. Tener instalado **Google Chrome** en el equipo.
3. Tener instalado **Git** en local. **Si es la primera vez que trabajas con git**, deberás configurar tu nombre y correo electrónico. Para ello, puedes usar el siguiente comando en la terminal de Windows o Git Bash:

```
C:\materiales> git config --global user.name "Nombre Apellido"  
C:\materiales> git config --global user.email "cuenta@iesdoctorbalmis.com"
```

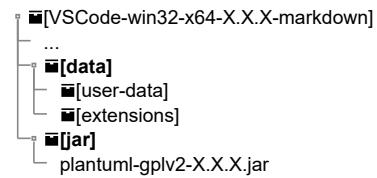
4. Tener el Java Runtime instalado mínimo la versión 17.
5. Tener instalado **Visual Studio Code**- Puedes descargar la versión portable ya preparada de **GDrive** en la carpeta del Departamento.

Si ya lo tienes instalado puedes crear un perfil personalizado de "**Apuntes con Markdown**" con las siguientes extensiones:

- **Spanish Language Pack for VSCode** (para traducir el IDE al español)
- **Code Spell Checker** (para detectar corrección ortográfica y sintáctica en textos)
- **Spanish - Code Spell Checker** (paquete de idioma español para el corrector ortográfico)
- **Open in browser** (para abrir un archivo HTML en el navegador con botón derecho)
- **Markdown All in One** (para trabajar con archivos markdown)
- **Markdown Preview Enhanced** (para visualizar el archivo .md)
- **markdownlint** (para comprobar el contenido del archivo)
- **PlauntUML** (para mostrar colores en bloques de código de diagramas)
- Visual Studio Keymap (para usar los atajos de teclado de Visual Studio)

Esquema de carpetas tras la instalación

En la carpeta **[data]** está la configuración local del usuario de VSCode, en la carpeta **[extensions]** están las extensiones instaladas y en la carpeta descritas en el punto anterior y en la carpeta **[user-data]** está la configuración global del usuario de VSCode como pueden ser los snippets, temas, configuraciones de usuario, etc.

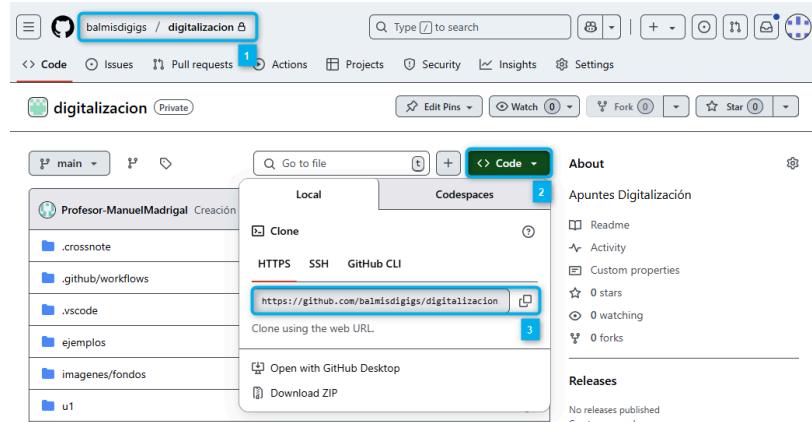


En la carpeta **[jar]** está el ejecutable de PlantUML que markdown-preview-enhanced utiliza para renderizar los diagramas, ya está preconfigurado en la instalación portable, pero para funcionar necesitarás tener el JRE de Java instalado como se comentaba anteriormente.

Acceso al Workspace

Este manual está pensado para trabajar con las herramientas anteriores y con un workspace de trabajo ya creado. Si lo que deseas es crear un nuevo workspace, puedes hacerlo seguir los pasos de la siguiente la guía [manual_crear_repositorio.pdf](#) que se encuentra junto a este manual.

Supongamos que estamos en una **organización de GitHub** y tenemos un espacio de trabajo ya creado en una organización siguiendo los pasos de la guía anterior. Veamos un ejemplo de cómo **clonarla en local** con el siguiente supuesto...



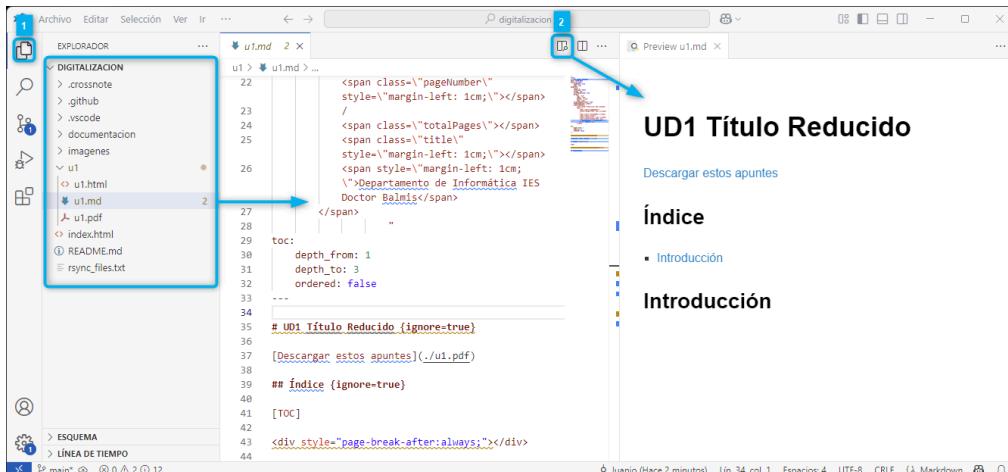
Como se muestra en la imagen, nos encontramos en el repositorio privado de materiales en nuestra organización 1. Pulsaremos el botón **Code** 2 y copiaremos la URL del repositorio 3. Por último, abriremos la terminal de Windows y ejecutaremos el siguiente comando para clonar el repositorio en local:

```
C:\materiales> git clone https://github.com/balmisdigigs/digitalizacion.git  
C:\materiales> cd digitalizacion  
C:\materiales\digitalizacion> code .
```

Descripción del contenido del repositorio

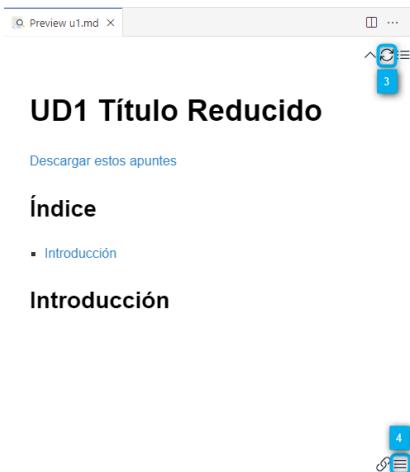
No es la idea de este manual profundizar en el uso de **VSCode**, pero sí trabajar las opciones principales que nos ofrece para trabajar con Markdown.

Al cargar nuestro workspace de digitalización y abrir un fichero **.md** (markdown) nos encontraremos con la siguiente pantalla:

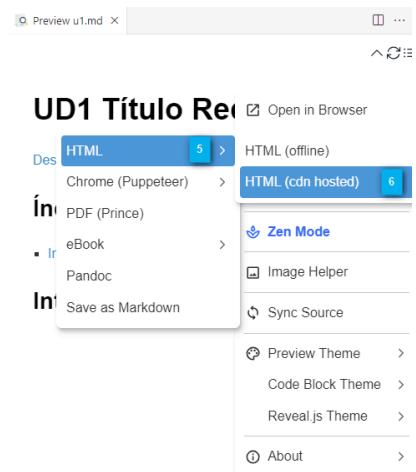


En ella, para ver el árbol de carpetas y archivos, pulsaremos el ícono de la parte superior izquierda **Explorador** 1. Si tenemos la extensión **Markdown Preview Enhanced**, en la parte superior derecha, tendremos el ícono de **Vista previa** (Markdown Preview Enhanced: Open Preview to the Side, **Ctrl + K, V**) 2.

En el **vista de previa**, nos renderizará a HTML todo el código que vayamos escribiendo y al situarnos sobre la misma, nos aparecerán nuevos iconos con opciones sobre el mismo. A destacar, el ícono arriba a la derecha para refrescar el contenido del preview 3 y el ícono para de menú 4 con diferentes opciones. En la imagen de ejemplo hemos seleccionado **Export HTML -> HTML (cdn hosted)** 5 & 6.



4



5

6

Trabajo básico con markdown

Dispones de la siguiente página con la documentación del Plugin en la [página de Markdown Preview Enhanced](#)

Dispones de una guía básica de [Markdown Básico](#) en la misma página.

Nota

La extensión **markdownlint** te ayuda ha escribir markdown normalizado, pero como estamos usando una extensión, no es es muy importante salvo respetar los saltos de línea.

En la carpeta `.vscode/` se han pre-definido varios code snippets (fragmentos de código) para el workspace situados en en fichero `FragmentosPernosalizados.code-snippets` dentro de la carpeta `.vscode/`. Estos fragmentos de código son plantillas que se insertan en el editor. Para usarlas, una vez abierto un fichero con extensión markdown escribiremos `mde_` seguido de `Ctrl + Space` que es el (trigger suggestions) en mi KeyMap.

Markdown básico

Encabezados

```
# Título 1  
## Título 2  
### Título 3  
...
```

Párrafos

Estándar en Markdown, solo hay que dejar una línea en blanco entre párrafos.

Párrafo con una palabra en **negrita**
Párrafo con una palabra en **cursiva**
Párrafo con una palabra en **negrita y cursiva**
Párrafo con una palabra en **~~tachado~~**
Párrafo con una palabra en **`código`**
Párrafo con una palabra en **```código en negrita```**
Párrafo con una palabra en **```código en cursiva```**
Párrafo con una palabra en **```código en negrita y cursiva```**

Los metacaracteres de Markdown son: `*`, `_`, `~`, `\`` y `#` y deberás escarparlos con `\`` si quieras que aparezcan en el texto.

Nota

Al tener el plugin "**Markdown All in One**" instalado, si selecciones una palabra y escribes un `*` o ``` se le añade el formato correspondiente a ambos lados de la selección y no sustituye el texto seleccionado.

Citas

En el estándar de Markdown, cada párrafo precedido del carácter `>` formará parte de la cita.

> Un país, una civilización se puede juzgar
> por la forma en que trata a sus animales.
— Mahatma Gandhi

Un país, una civilización se puede juzgar
por la forma en que trata a sus animales.
— Mahatma Gandhi

Citas fuera del stándar

También podremos definir una cita con las [admonitions de Mkdocs](#). Las tienes definidas como fragmentos de código con la raíz `mde_cuadro_` por tanto si pulsas `mde_cuadro_` seguido de `Ctrl + Space` te ofrecerá `mde_cuadro_cita` que te insertará el fragmento para citas.

!!! Quote Cita

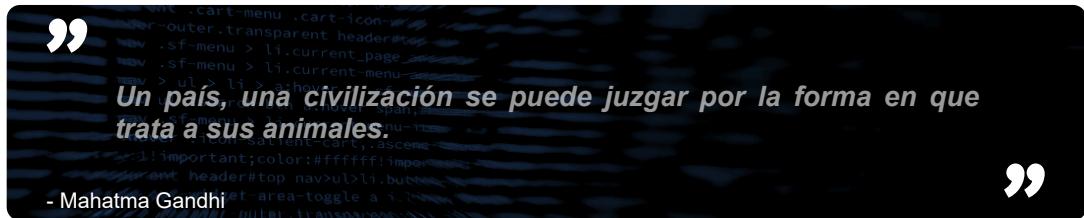
Un país, una civilización se puede juzgar
por la forma en que trata a sus animales.
— Mahatma Gandhi

„ Cita

Un país, una civilización se puede juzgar
por la forma en que trata a sus animales.
— Mahatma Gandhi

Un poco más vistosas, las tienes definidas como fragmentos de código con `mde_cita_HTML` que generará lo siguiente:

```
<div class="contenedor">
  <div class="fondo">
    <div class="abre_comilla">"</div>
    <div class="cierra_comilla">"</div>
    <div class="cita">Un país, una civilización se puede
      juzgar por la forma en que trata a sus animales.</div>
    <div class="autor">- Mahatma Gandhi</div>
  </div>
</div>
```



Listas

```
* Elemento 1
* Elemento 2
 * Elemento 2.1 (alineado primera letra item padre)
   * Elemento 2.1.1
* Elemento 3
```

```
1. Elemento 1
2. Elemento 2
 1. Elemento 2.1 (alineado primera letra item padre)
    1. Elemento 2.1.1antes)
3. Elemento 3
```

Respetar los espacios es muy importante para que cierto contenido pertenezca o no a un elemento de la lista.

```
* Elemento 1
Este párrafo pertenece al elemento 1
* Conviene separar las listas indentadas
Este párrafo pertenece al elemento 1.1
Este párrafo pertenece al elemento 1.1

Separa con un **salto de línea** para cotinuar
en el nivel anterior
* Elemento 2
Este párrafo pertenece al elemento 2
Este párrafo pertenece al elemento 2

Este párrafo ya está fuera de la lista
```

- Elemento 1

Este párrafo pertenece al elemento 1

- Conviene separar las listas indentadas

Este párrafo pertenece al elemento 1.1

Este párrafo pertenece al elemento 1.1

Separa con un **salto de línea** para cotinuar
en el nivel anterior

- Elemento 2

Este párrafo pertenece al elemento 2

Este párrafo pertenece al elemento 2

Este párrafo ya está fuera de la lista

Tablas simples

| | | |
|------------|------------|------------|
| Columna 1 | Columna 2 | Columna 3 |
| ----- | ----- | ----- |
| Elemento 1 | Elemento 2 | Elemento 3 |
| Elemento 4 | Elemento 5 | Elemento 6 |

| Columna 1 | Columna 2 | Columna 3 |
|------------|------------|------------|
| Elemento 1 | Elemento 2 | Elemento 3 |
| Elemento 4 | Elemento 5 | Elemento 6 |

 **Tip:** Con **Ctrl + K + D** puedes hacer que la tabla se alinee correctamente.

Enlaces

1. Enlace normal por ejemplo a [Goole](#):

```
[Goole](http://www.google.com)
```

2. Enlace **negrita** por ejemplo a [Goole](#):

```
**[Goole](http://www.google.com)**
```

3. Referencia directa <http://www.google.com>

```
<http://www.google.com>
```

Imágenes



Tip

Si usas VSCode y arrastras el archivo de imagen a la ventana del editor y justo antes de soltarla (drop) pulsas la tecla **Shift** te insertará la imagen en el formato correcto para Markdown.

Estándar imágenes básico

```
![Texto alternativo](assets/imagenes/generar_contenido/imagen1.png)
![Texto alternativo](assets/imagenes/generar_contenido/imagen1.png "título alternativo al pasar el ratón")
![Texto alternativo](assets/imagenes/generar_contenido/imagen1.png){ width=200 }
![Texto alternativo](assets/imagenes/generar_contenido/imagen1.png){ height=100 }
```

Por ejemplo:

```
![Texto alternativo](assets/imagenes/generar_contenido/imagen1.png "Gata calico"){ width=300}
```



Ampliación imágenes

Un poco más vistosas, las tienes definidas como fragmentos de código con `mde_imagenCentrada` que generará lo siguiente, donde controlamos el tamaño de la imagen y al tener el margen automático, se centra la imagen en la página.

```
![nombre](assets/imagenes/generar_contenido/imagen1.png){ style="display:block; margin:0 auto; width:75%;max-width:300px;" }
```



Expresiones matemáticas

Fuera del estándar de Markdown, para escribir expresiones matemáticas, **Markdown Preview Enhanced** utiliza **KaTeX** que representa las expresiones como si fuera Latex, en este enlace tienes una [chuleta de uso](#).

Añadiendo **un solo símbolo del dolar** antes y después del código Latex `$f(x) = x^2+3$` que generaría como está función $f(x) = x^2 + 3$ con display en línea.

Añadiendo **dos símbolos del dólar** antes y después del código Latex `$$f(x) = x^2+3$$` la misma función con display block, esto es en un párrafo a parte y centrada.

$$f(x) = x^2 + 3$$

Ejemplo en línea:

The homomorphism f is injective if and only if its kernel is only the singleton set e_G , because otherwise $\exists a, b \in G$ with $a \neq b$ such that $f(a) = f(b)$.

The homomorphism f is injective if and only if its kernel is only the singleton set e_G , because otherwise $\exists a, b \in G$ with $a \neq b$ such that $f(a) = f(b)$.

Ejemplo en bloque:

```
$$
\cos x =
\sum_{k=0}^{\infty} \frac{(-1)^k}{(2k)!} x^{2k}
```

$$\cos x = \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k)!} x^{2k}$$

Otros ejemplos:

```
$$
\lim_{x \rightarrow \infty} \frac{1}{x} = 0
$$
```

$$\lim_{x \rightarrow \infty} \frac{1}{x} = 0$$

```
$$
x = \begin{cases} a & \text{if } \\ c & \text{if } \end{cases}
\end{cases}
$$
```

$$x = \begin{cases} a & \text{if } b \\ c & \text{if } d \end{cases}$$

```
$$
\begin{pmatrix} a & b \\ c & d \end{pmatrix}
\begin{pmatrix} a & b \\ c & d \end{pmatrix}
$$
```

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

```
$$
```

```
\begin{pmatrix}
1 & 2 \\
3 & 4
\end{pmatrix}
```

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \text{ es distinto a } \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$$

```
$$
```

```
\alpha, \beta, \gamma, \Delta
$$
```

```
$$
\int_a^b f(x) dx
$$
```

$$\int_a^b f(x) dx$$

Bloques de código

Enlaces

Code Chunks - Markdown Preview Enhanced

Además de la ` para escribir código en línea, podemos usar ``` para escribir bloques de código. En el caso de que queramos resaltar el lenguaje a usar, **lo indicaremos justo después de la primera línea de apertura del bloque de código**. **Markdown Preview Enhanced** usa la librería **Prism.js** para resaltar el código. Puedes ver la lista de lenguajes soportados en la [página de Prism.js](#).

❖ **Nota:** Puedes usar el code snippet `mde_bloque_C#` para recordar como añadir líneas y resaltarlas.

```
```js
function hola() {
| console.log("Hello world!");
}
```

```

```
```js {.line-numbers}
function hola() {
| console.log("Hello world!");
}
```

```

```
```js {highlight=2}
function hola() {
| console.log("Hello world!");
}
```

```

```
function hola() {
|   console.log("Hello world!");
}
```

```

```
1 | function hola() {
2 | console.log("Hello world!");
3 | }
```

```

```
function hola() {
②   console.log("Hello world!");
}
```

```

# Admonitions

## Enlaces

### Admonitions de MKdocs

Deberemos usar `!!! <tipo> <título>` y posteriormente el código englobado en el cuadro en una línea inferior indentado/tabulado a la derecha.

```
!!! note Nota
Este es un bloque de nota.
Con dos líneas de texto.
```

Dispondremos de **varios tipos** que puedes visualizar en la siguiente tabla:

|                                                                                                     |                                                                                            |                                                                                              |
|-----------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------|
|  note              |  info     |  abstract |
|  example           |  quote    |  tip      |
|  success           |  question |  warning  |
|  failure o error |  danger |  bug    |

# Emojis de GitHub Markdown

Con el plugin de **Markdown Preview Enhanced** podemos usar los emojis de GitHub. Para ello, simplemente escribiremos el nombre del emoji entre dos puntos `:`. Por ejemplo: `:smile:` representará

## Ejemplos de iconos que pueden ser útiles:

| Números                      | Señales                           | Objetos                  | Otros                       |
|------------------------------|-----------------------------------|--------------------------|-----------------------------|
| <code>1 :one:</code>         | <code>:heavy_check_mark:</code>   | <code>:computer:</code>  | <code>:fire:</code>         |
| <code>2 :two:</code>         | <code>:heavy_plus_sign:</code>    | <code>:book:</code>      | <code>:star:</code>         |
| <code>3 :three:</code>       | <code>:warning:</code>            | <code>:pencil2:</code>   | <code>:+1:</code>           |
| <code>4 :four:</code>        | <code>:x:</code>                  | <code>:bulb:</code>      | <code>:-1:</code>           |
| <code>5 :five:</code>        | <code>:construction:</code>       | <code>:mag:</code>       | <code>:hand:</code>         |
| <code>6 :six:</code>         | <code>:white_check_mark:</code>   | <code>:pushpin:</code>   | <code>:mega:</code>         |
| <code>7 :seven:</code>       | <code>:arrow_right:</code>        | <code>:package:</code>   | <code>:link:</code>         |
| <code>8 :eight:</code>       | <code>:arrow_forward:</code>      | <code>:computer:</code>  | <code>:rocket:</code>       |
| <code>9 :nine:</code>        | <code>:information_source:</code> | <code>:clipboard:</code> | <code>:pill:</code>         |
| <code>10 :keycap_ten:</code> | <code>:skull:</code>              | <code>:key:</code>       | <code>:mortar_board:</code> |

**Nota:** Puedes ver la lista completa de emojis en la [GitHub Emoji Cheat Sheet](#).

# Diagramas

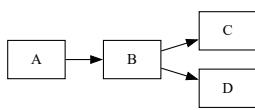
## Enlaces

- [Diagrams - Markdown Preview Enhanced](#)
- [Graphviz](#)

**Markdown Preview Enhanced** soporta varios tipos de diagramas, entre ellos: **PlantUML**, **Mermaid**, **Graphviz** etc. Vamos a ver algunos de los más comunes.

La sintaxis es similar a la de los bloques de código, pero en vez de usar el nombre del lenguaje, usaremos el nombre del tipo de diagrama. Por ejemplo vemos un diagrama de tipo **dot** de **Graphviz**:

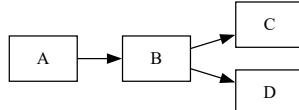
El tamaño ( "ancho,alto" ) lo establecemos en pulgadas con la propiedad **size**. Por defecto el diagrama quedará alineado a la derecha dentro de su contenedor.



No especificamos tamaño al interprete de Graphviz y lo hacemos al parser de Markdown Preview Enhanced. Donde le decimos que esté centrado y le aplicamos un zoom del 0.7 (70%) con la propiedad **style**.

💡 Aplicar zoom a través del estilo no es buena práctica.

💡 Es la mejor opción, ya que establecemos un **viewport en píxeles** y el **zoom de 0.7 (70%)** para que el diagrama quepa en el viewport. Además, de forma opcional le podemos indicar que **centre el viewport en el nodo B**.



```
```dot
digraph G {
    size = "3,2";
    rankdir = LR;
    node [shape=box];
    A --> B;
    B --> C;
    B --> D;
}
````
```

```
```dot [align="center", style="zoom: 0.7;"]
digraph G {
    rankdir = LR;
    node [shape=box];
    A --> B;
    B --> C;
    B --> D;
}
````
```

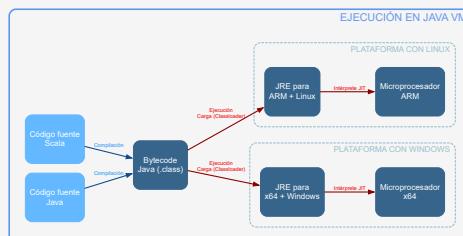
```
```dot
digraph G {
    viewport = "200,100,0.7,B";
    rankdir = LR;
    node [shape=box];
    A --> B;
    B --> C;
    B --> D;
}
````
```

## Ejemplo diagrama graphviz representando JRE:

```

graph TD
 viewPort = "350,180,0.4";
 graph LR
 rankDir = LR
 ranksep = 0.5,
 fontname = "Arial"
]
 node[
 fontname = "Arial",
 fontsize = 15,
 style = "filled, rounded",
 color = steelblue4, fontcolor = white,
 shape = box,
 height = 1.25,
 width = 1
]
 edge[
 fontname = "Arial",
 color = dodgerblue4,
 penwidth = 3,
 style = "solid, rounded"
 fontcolor = dodgerblue,
 fontsize = 10
]
 subgraph cluster_ejecucion_jvm {
 label="EJECUCIÓN EN JAVA VM";
 margin = 30
 fontsize = 20
 labelLoc = "t"
 labelJust = "c"
 style = "solid, rounded"
 fontcolor=cornflowerblue
 color = cornflowerblue
 penwidth = 3
 }
 subgraph cluster_plataforma_x64w {
 label="PLATAFORMA CON WINDOWS";
 margin = 20
 fontsize = 15
 labelLoc = "t"
 labelJust = "r"
 style = "dashed, rounded"
 fontcolor=lightblue3
 color = lightblue3
 penwidth = 2
 }
 subgraph cluster_plataforma_armlin {
 label="PLATAFORMA CON LINUX";
 margin = 20
 fontsize = 15
 labelLoc = "t"
 labelJust = "r"
 style = "dashed, rounded"
 fontcolor=lightblue3
 color = lightblue3
 penwidth = 2
 }
 jre_x64w [label="JRE para\x00a1x64 + Windows"];
 microprocesador_x64w [label="Microprocesador\x00a1x64"];
 jre_x64w --> microprocesador_x64w [label="Intérprete JIT", fontcolor="red2", color="red4"];
 jre_x64w [label="JRE para\x00a1x64 + Windows"];
 microprocesador_armlin [label="Microprocesador\x00a1ARM"];
 jre_armlin [label="JRE para\x00a1ARM + Linux"];
 jre_armlin --> microprocesador_armlin [label="Intérprete JIT", fontcolor="red2", color="red4"];
 jre_x64w [label="JRE para\x00a1x64 + Windows"];
 bytecode_java [label="Bytecode Java (.class)"];
 bytecode_scala [label="Bytecode Scala (.class)"];
 bytecode_java --> bytecode_scala [label="Compilación"];
 bytecode_java [label="Bytecode Java (.class)"];
 bytecode_java --> jre_x64w [label="Ejecución\nCarga (ClassLoader)", fontcolor="red2", color="red4"];
 bytecode_java --> jre_armlin [label="Ejecución\nCarga (ClassLoader)", fontcolor="red2", color="red4"];
 bytecode_scala [label="Bytecode Scala (.class)"];
 bytecode_scala --> bytecode_java [label="Compilación"];
 bytecode_scala [label="Bytecode Scala (.class)"];
 bytecode_scala --> jre_x64w [label="Ejecución\nCarga (ClassLoader)", fontcolor="red2", color="red4"];
 bytecode_scala --> jre_armlin [label="Ejecución\nCarga (ClassLoader)", fontcolor="red2", color="red4"];
}

```



## Renderizado a través de Kroki

**Markdown Preview Enhanced** soporta además renderizado a través de **servidor de Kroki**, que es un servicio de diagramas online. Para ello, deberás configurar el servidor <https://kroki.io/> en la extensión de **Markdown Preview Enhanced** (ya está configurado por defecto). Puesto que es un servicio online, no es necesario instalar nada en local. Pero solo funcionará si tienes acceso a Internet y el servidor está activo o admite el tipo de diagrama que deseas generar. Para renderizar con Kroki, simplemente debes escribir el nombre del diagrama seguido de `{kroki=true}`.

Por ejemplo: ````dot {kroki=true}````

## Incluir diagramas desde un fichero externo

### Enlaces

#### File Imports - Markdown Preview Enhanced

La sintaxis para incluir un fichero externo es `@import "<fichero>"{<modificadores_opcionales>}`

Realmente puedes importar muchos tipos de ficheros..

- **Imágenes:** png, jpg, svg, bmp, gif, ...
- **Datos a ver como tabla:** csv
- **Diagramas:** dot, puml, mermaid.
- **Código:** js, py, json, html, css, less, md

Un uso común es importar diagramas. Por ejemplo, el diagrama dot del ejemplo anterior lo tenemos guardado en un fichero `JRE.dot`. Para importarlo, simplemente escribimos:

```
@import "assets/imagenes/generar_contenido/JRE.dot" {align="center"}
```

y el plugin se encargará de renderizarlo en local y si ponemos `{align="center", kroki=true}` a través del servidor de Kroki.

# Ejemplos con PlantUML

## Enlaces

[Página oficial de PlantUML](#)

En ella puedes todos los tipos de diagramas que se pueden generar y suelen estar soportados por **Markdown Preview Enhanced** y otros parsers de markdown. Si renderizamos en local, deberemos tener el JRE instalado y el parser `plantuml.jar` referenciado en la configuración de la extensión (si estás utilizando nuestro entorno esto ya está hecho), en caso contrario, podemos usar **Kroki**.

Se han añadido algunos fragmentos de código para facilitar la creación de diagramas.

## Ejemplo contenido carpetas (wireframe puml)

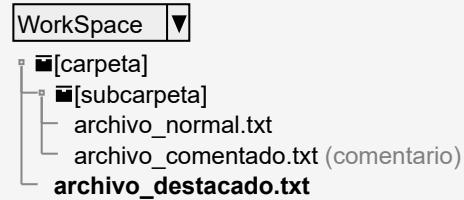
Es un diagrama de tipo **wireframe** que representa un espacio de trabajo con carpetas y archivos.

Usaremos el fragmento `mde_puml + Ctrl + Space` y seleccionaremos `mde_puml_folder_tree + Tab` insertándonos el siguiente código. Fíjate que el tamaño, lo hemos controlado con la propiedad `Scale` y si queremos que el color de fondo sea transparente, lo indicamos con `skinparam backgroundColor Transparent`.

```
@startsalt
```

```
scale 1.5
```

```
{
 ^Workspace^
 {T
 + <&box>[carpeta]
 ++ <&box>[subcarpeta]
 +++ archivo_normal.txt
 +++ archivo_comentado.txt <color:gray>(comentario)
 ++ **archivo_destacado.txt**
 }
}
```



## Ejemplo arquitectura (C4 puml)

El **modelo C4** es un estándar para la representación de arquitecturas de software. La idea es representar la arquitectura de un sistema en diferentes niveles de detalle (hasta 4).

Para usarlo con **PlantUML** usaremos la librería **C4-PlantUML** que es un conjunto de plantillas para representar el modelo C4. Para obtener los iconos de los diferentes tipos de sistemas, puedes usar la librería **devicons2** que ya los define el formato que entiende **PlantUML**. En este caso, usaremos el icono de **android** para la aplicación, **tomcat** para el servidor y **mysql** para la base de datos.

Usaremos el fragmento `mde_puml + Ctrl + Space` y seleccionaremos `mde_puml_c4 + Tab` insertándonos el siguiente código.

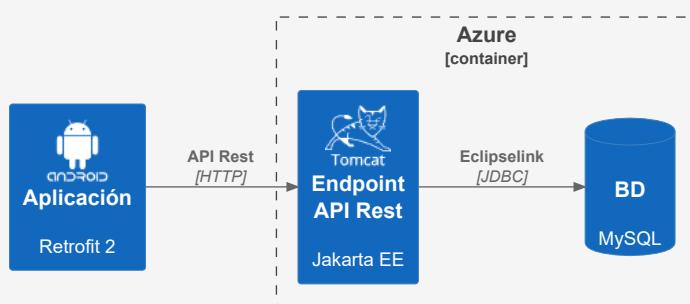
```
```puml {align="center", korki=true}
@startuml
!include https://raw.githubusercontent.com/plantuml-stdlib/C4-PlantUML/master/C4_Container.puml
!include <tupadr3/devicons2/tomcat_line_wordmark>
!include <tupadr3/devicons2/mysql_wordmark>
!include <tupadr3/devicons2/android_wordmark>

skinparam backgroundColor Transparent
Scale 1.0

HIDE_STEREOTYPE()
LAYOUT_LEFT_RIGHT()

System(android, "Aplicación", "Retrofit 2", $sprite="android_wordmark")
Container_Boundary(azure, "Azure") {
    System(api, "Endpoint\nnAPI Rest", "Jakarta EE", $sprite="tomcat_line_wordmark")
    SystemDb(db, "BD", "MySQL", $sprite="mysql")
}
Rel(android, api, "API Rest", "HTTP")
Rel(api, db, "Eclipselink", "JDBC")
@enduml
```

```



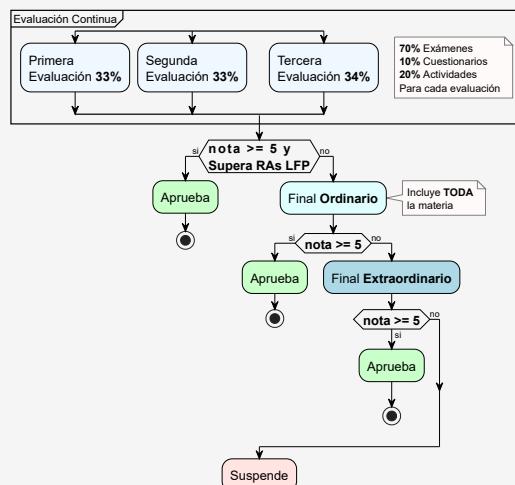
## Ejemplo mapa mental

En este ejemplo vamos a incluir un mapa mental. Además, hemos definido un estilo personalizado por niveles de profundidad. Para así ver como añadir estilos a Los diagramas para que todos queden homogéneos.

Usaremos el fragmento `mde_puml + Ctrl + Space` y seleccionaremos `mde_puml_mindmap + Tab` insertándonos el siguiente código.

Puesto que el código es un poco largo, lo hemos sacado a un fichero externo `mindmap.puml` y lo importamos con `@import`. Puedes ver el código completo en el fichero `mindmap.puml` que está en la carpeta `assets/imagenes/generar_contenido/`.

```
@import "assets/imagenes/generar_contenido/mindmap.puml" {align="center"}
```



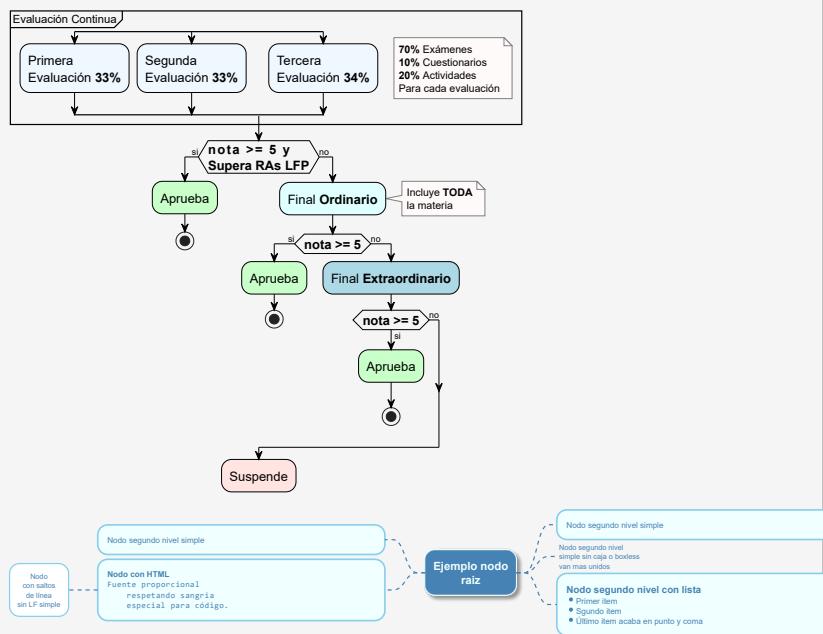
## Ejemplo diagrama de flujo

```

@startuml
skinparam classAttributeIconsize 0
skinparam ArrowColor Black
skinparam ActivityBackgroundColor WhiteSmoke
skinparam ActivityDiamondBackgroundColor WhiteSmoke
skinparam ActivityBorderColor Black
skinparam ActivityDiamondBorderColor Black
skinparam ClassBackgroundColor Snowstart
skinparam ActivityFontSize 16
skinparam ActivityDiamondFontSize 16
skinparam ActivityDiamondFontStyle bold
skinparam NoteBackgroundColor Snow
skinparam NoteBorderColor Gray
skinparam BackgroundColor Transparent
Scale 0.7

partition "Evaluación Continua" {
 split
 #aliceBlue:Primera\nEvaluación **33%**;
 split again
 #aliceBlue:Segunda\nEvaluación **33%**;
 split again
 #aliceBlue:Tercera\nEvaluación **34%**;
 floating note right
 70% Exámenes
 10% Cuestionarios
 20% Actividades
 Para cada evaluación
 end note
 end split
 end split
 end partition
 if (nota >= 5 y\nSupera RAs LFP) then (si)
 #technology:Aprueba;
 stop
 else (no)
 #lightCyan:Final **Ordinario**;
 note right
 Incluye **TODA** la materia
 end note
 if (nota >= 5) then (si)
 #technology:Aprueba;
 stop
 else (no)
 #lightBlue:Final **Extraordinario**;
 if (nota >= 5) then (si)
 #technology:Aprueba;
 stop
 else (no)
 endif
 endiff
 endif
 -[#green]->
 #mistyrose:Suspende;
 endifum
}

```



# Ejemplos con Mermaid

[Enlaces](#)

[Página oficial de Mermaid](#)

No necesitas ningún tipo de instalación extra, ya que **Markdown Preview Enhanced** ya lo incluye. Puedes ver la lista de diagramas soportados en la [página de Mermaid](#).

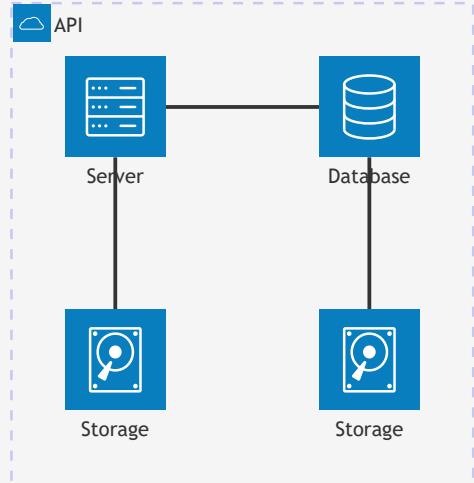
**Aviso:** Este tipo de diagramas están más pensado para redenizar diagramas con frameworks de generación de documentación basados en JavaScript. Como por ejemplo: Docusaurus, VuePress o Astro Starlight, etc.

## Ejemplo arquitectura (mermaid)

Para este tipo de diagramas, usaremos el fragmento `mde_mermaid + Ctrl + Space` y seleccionaremos `mde_mermaid_architecture + Tab` insertándonos el siguiente código.

```
```mermaid {align="center"}
architecture-beta
group api(cloud) [API]
    service db(database) [Database] in api
    service disk1(disk) [Storage] in api
    service disk2(disk) [Storage] in api
    service server(server) [Server] in api

    db:L -- R:server
    disk1:T -- B:server
    disk2:T -- B:db
````
```



# Maquetación compleja basada en Bootstrap 5

En el marco de trabajo se ha incluido un enlace a **Bootstrap 5** para poder usar su sistema de maquetación y sus componentes. El enlace se ha incluido en el fichero `.crossnote/style.less`. Esto nos va a permitir maquetar de forma responsive sin mucha complejidad.

## Aviso

Markdown no tiene contemplada la maquetación compleja. Por lo que estaremos obligados a usar etiquetas HTML para ello. Además y más importante, requeriremos de conexión a Internet para que funcione correctamente pues el CSS de Bootstrap 5 es CDN Hosted. Por tanto, **si no tienes conexión a Internet, la maquetación no se verá correctamente**.

## Maquetación a 2 o 3 columnas

Cualquier fragmento de código (Snippet) que empiece por la raíz `mde_bs5` + `Ctrl + Space` incluirá Bootstrap 5. Por ejemplo, el siguiente Snippet `mde_bs5_dosColumnas` + `Tab` generará el siguiente código:

```
<div class="row">
<div class="col-sm-6 my-auto">

<!-- Escribe aquí tu markdown respetando una línea de separación -->

</div>
<div class="col-sm-6 my-auto">

<!-- Escribe aquí tu markdown respetando una línea de separación -->

</div>
</div>
```

En los comentarios puedes incluir cualquier markdown de los vistos anteriormente **respetando el salto de línea antes y después de comentario**. Por ejemplo, vamos a insertar 2 imágenes....

```
<div class="row">
<div class="col-sm-6 my-auto">

![Texto alternativo](assets/imagenes/generar_contenido/imagen1.png "título alternativo al pasar el ratón")

</div>
<div class="col-sm-6 my-auto">

![Texto alternativo](assets/imagenes/generar_contenido/imagen2.png "título alternativo al pasar el ratón")

</div>
</div>
```



Podremos cambiar añadir y cabiar el ancho de las columnas siempre y cuando la suma de los anchos sea 12. Por ejemplo ...

```
<div class="row">
<div class="col-sm-6">

Aquí incluimos un texto en la primera columna de 6 y dos imágenes en la segunda y tercera de 3 y 3 de ancho respectivamente.
Además hemos quitado el **`my-auto`** para que ajuste los párrafos a la parte superior de la columna.

</div>
<div class="col-sm-3 my-auto">

![Texto alternativo](assets/imagenes/generar_contenido/imagen1.png "título alternativo al pasar el ratón")

</div>
<div class="col-sm-3 my-auto">

![Texto alternativo](assets/imagenes/generar_contenido/imagen2.png "título alternativo al pasar el ratón")

</div>
</div>
```

Aquí incluimos un texto en la primera columna de 6 y dos imágenes en la segunda y tercera de 3 y 3 de ancho respectivamente.

Además hemos quitado el `my-auto` para que ajuste los párrafos a la parte superior de la columna.



No vamos a entrar en más detalles de Bootstrap 5, ya que puedes ver la documentación en su [página oficial](#). Además, no es una buena idea usarlo ya que está fuera del estándar de Markdown.

## Generar PDF

Si estamos viendo la vista

```

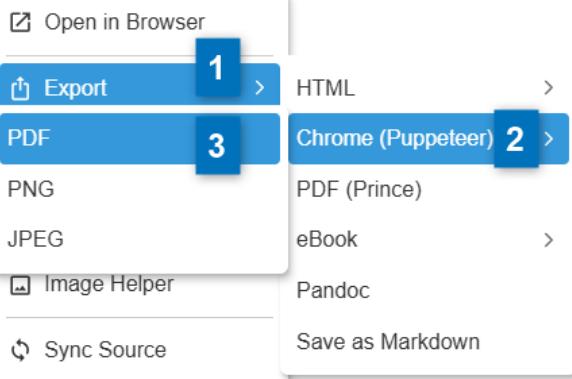
...
export_on_save:
 puppeteer: true
 html: true
puppeteer:
 scale: 1
 landscape: false
 preferCSSPageSize: true # Use CSS @page size if available
 format: "A4"
 printBackground: true
 margin:
 top: "1cm"
 right: "1cm"
 bottom: "2.5cm"
 left: "1cm"
 displayHeaderFooter: true
 headerTemplate: " "
 footerTemplate: "

 /

 >Departamento de Informática IES Doctor Balmis

 ...

```



## Conclusiones

### Vantajas

- ✓ Ideal para trabajo en equipo o en grupo.
- ✓ Homogeneidad en la generación de contenido.
- ✓ Historial y revisión de cambios integrado.
- ✓ Automatización en la generación de documentos y despliegue.
- ✓ Permite bifurcaciones.
- ✓ Permite familiarizarse con tecnologías como MarkDown y Git (GitHub) para después aplicarlas con los alumnos en proyectos de grupo o transversales.

### Desventajas

- ✗ Hacer maquetaciones complejas es 'costoso'.
- ✗ Requiere del manejo de varias herramientas y tecnologías.
- ✗ Requiere de una estructura inicial '*compleja*'.
- ✗ Tiene poco sentido si no se va a generar documentación técnica o se va a trabajar de forma individualizada.
- ✗ Documentación '*pobre*' en ocasiones.