

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

[Descargar estos apuntes](#)

Índice

- ▼ 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)
 - ▼ Ejemplos con Mermaid
 - Ejemplo arquitectura (mermaid)
- ▼ Generar PDF
 - Añadir saltos de página al PDF

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.

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 escaparlos con `\\` si quieras que aparezcan en el texto.

Nota

Al tener el plugin "**Markdown All in One**" instalado, si seleccionas 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

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 2.1.1
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 continuar
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 continuar
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 alienee 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 donde se puede controlar el tamaño de la imagen y con el margen automático, que permite que se centre la imagen en la página.

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



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:

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

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

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

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

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

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

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

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

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

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

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

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

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](#).

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

```

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

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

```

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

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

```
function hola() {
 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
1 :one:	✓ :heavy_check_mark:	💻 :computer:	🔥 :fire:
2 :two:	+ :heavy_plus_sign:	📖 :book:	⭐ :star:
3 :three:	⚠ :warning:	✏️ :pencil2:	👍 :+1:
4 :four:	✗ :x:	💡 :bulb:	👎 :-1:
5 :five:	🚧 :construction:	🔍 :mag:	✋ :hand:
6 :six:	✓ :white_check_mark:	📌 :pushpin:	📣 :mega:
7 :seven:	➡ :arrow_right:	📦 :package:	🔗 :link:
8 :eight:	▶ :arrow_forward:	💻 :computer:	🚀 :rocket:
9 :nine:	ℹ :information_source:	📋 :clipboard:	💊 :pill:
10 :keycap_ten:	💀 :skull:	🔑 :key:	🎓 :mortar_board:

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

# Diagramas

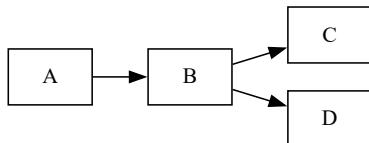
## Enlaces

- [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.



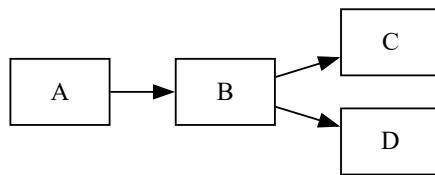
```

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

```

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`.



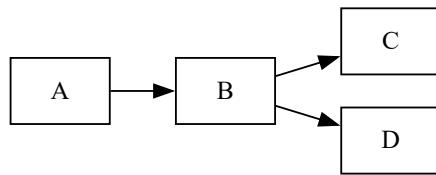
```

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

```

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

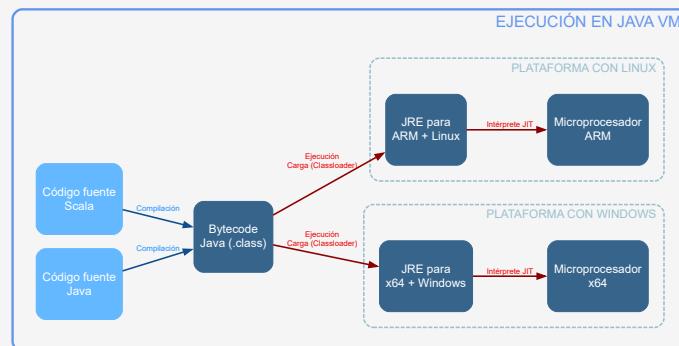
```

## Ejemplo diagrama graphviz representando JRE:

```

digraph {
 viewport = "350,180,0.4";
 graph [
 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 = 2,
 fontcolor = dodgerblue,
 fontsize = 10
]
 subgraph cluster_ejecucion_jvm {
 label="EJECUCIÓN EN JAVA VM";
 margin = 30
 fontsize = 20
 labelloc = "t"
 labeljust="r"
 style = "solid, rounded"
 fontcolor=cornflowerblue
 color =cornflowerblue
 penwidth = 3
 }
 codigo_fuente_java [label="Código fuente\nJava", color = steelblue1, fontcolor = white];
 codigo_fuente_scala [label="Código fuente\nScala", color = steelblue1, fontcolor = white];
 bytecode_java [label="Bytecode\nJava (.class)"];
 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
 }
 jre_x64w [label="JRE para\nx64 + Windows"];
 microprocesador_x64w [label="Microprocesador\nx64"];
 jre_x64w -> microprocesador_x64w [label="Intérprete JIT", fontcolor="red2", color="red4"];
}
 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_armlin [label="JRE para\nARM + Linux"];
 microprocesador_armlin [label="Microprocesador\nARM"];
 jre_armlin -> microprocesador_armlin [label="Intérprete JIT", fontcolor="red2", color="red4"];
}
 codigo_fuente_java -> bytecode_java [label="Compilación"];
 codigo_fuente_scala -> bytecode_java [label="Compilación"];
 bytecode_java -> jre_x64w [label="Ejecución\nCarga (ClassLoader)", fontcolor="red2", color="red4"];
 bytecode_java -> 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, en caso contrario, podemos usar **Kroki**.

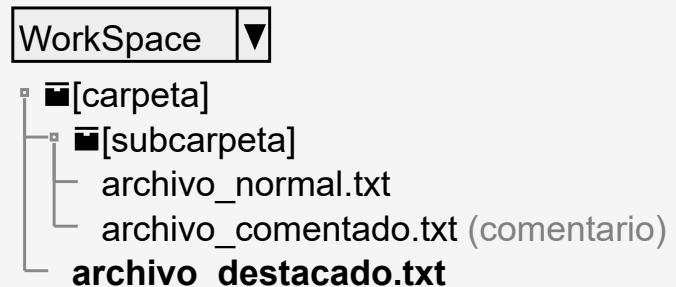
Se han añadido algunos fragmentos de código para facilitar la creación de diagramas.

## Ejemplo contenido carpetas (wireframe puml)

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

Como ejemplo, puedes ver el resultado del 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**
 }
}
@endsalt
```



## 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 ícono 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.

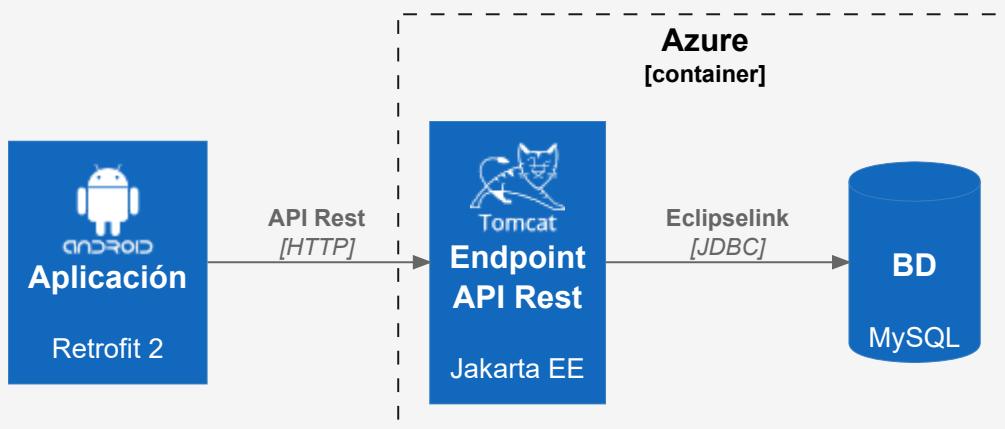
```
```puml {align="center", korki=true}
@startuml
!include https://raw.githubusercontent.com/plantuml-stdlib/c4-PlantUML/master/C4_Container.puml
!include <tupadr3/devicons2/tomcat_line_wordmark>
!include <tupadr3/devicons2/mysql_wordmark>
!include <tupadr3/devicons2/android_wordmark>

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\nAPI 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
```

```





# Ejemplos con Mermaid

## Enlaces

[Página oficial de Mermaid](#)

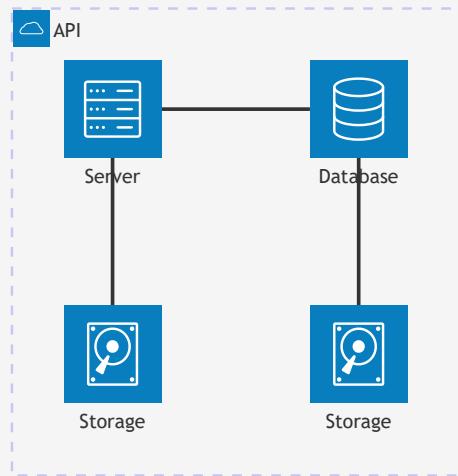
No necesitas ningún tipo de instalación extra, ya que **Markdown Preview Enhanced** ya lo incluye. Puedes ver la lista de diagramas soportados en la [página de Mermaid](#).

 **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)

```
```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
````
```



# Generar PDF

Para generar el PDF, usaremos la librería **Puppeteer** que es una librería de Node.js que proporciona una API de alto nivel para controlar Chrome o Chromium a través del protocolo DevTools. Permite generar PDFs a partir de páginas web, entre otras cosas. **Solo deberemos tener instalado Google Chrome o Chromium en el sistema para poder usarla.**

Con el plugin de *Markdown Preview Enhanced*, si estamos viendo la vista, modificamos el markdown y lo guardamos, se generará un PDF automáticamente. Para ello, en el frontmatter en YAML, deberemos tener configurada la propiedad `export_on_save` y configurado el atributo `puppeteer` a `true`. Además, a través de la propiedad `puppeteer`, podemos **configurar** el tamaño del PDF, márgenes, orientación, etc. Como se muestra en la configuración del frontmatter del siguiente ejemplo:

```

...
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

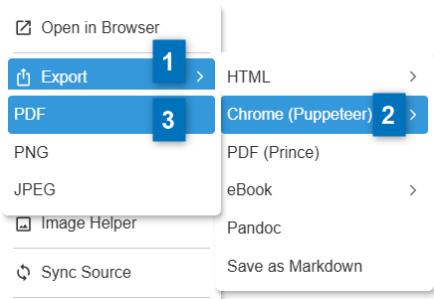
...

```

Fíjate que hemos añadido `printBackground` a `true` para que se impriman los fondos de los elementos, ya que por defecto está a `false`. Además, hemos añadido `preferCSSPageSize` para poder

configurar configuraciones de página específicas en el CSS distintas a las comunes definidas en el frontmatter.

También podemos generar el PDF desde la vista previa del markdown, para ello, simplemente botón derecho para obtener el menú contextual y seleccionar la opción 1 Export > posteriormente la opción 2 Chrome (Puppeteer) y por último la opción 3 PDF . Esto generará el PDF en la carpeta pdf del proyecto. Esto además de generar el PDF, nos lo previsualizará.



Cómo lo que hace es imprimir el HTML generado por el parser de Markdown Preview Enhanced, a través de Puppeteer. Muchas veces no controlamos los saltos de página y puede que el PDF no quede como esperamos.

## Añadir saltos de página al PDF

Para añadir saltos de página al PDF, podemos usar la etiqueta HTML:

```
<div style="page-break-after:always;"></div>
```

El siguiente código nos permitirá intercalar páginas en horizontal en el PDF. Para mostrar tablas grandes o imágenes grandes. Para ellos, deberemos seguir el siguiente esquema.

```
<div class="landscape" style="page-break-after:always;"></div>
```

```
<p>Página 1 en vertical</p>
```

```
<div style="page-break-after:always;"></div>
```

```
<p>Página 2 en vertical</p>
```

```
7 <div class="landscape" style="page-break-after:always;">
```

```
<p>Página 3 en HORIZONTAL</p>
```

```
11 </div>
```

```
<p>Página 4 en vertical</p>
```