

DOCUMENTACIÓN DE APLICACIONES

Versión v1.0, 20 de diciembre de 2025: Primera versión.

Tabla de Contenido

1. ¿Por qué es importante documentar?	2
2. Tipos de documentación	3
2.1. Documentación de pruebas	3
2.2. Documentación técnica	3
3. Manuales y guías	4
3.1. Manual y guía de usuario	4
3.2. Manual y guía de explotación	5
3.3. Guía rápida y guía de referencia	6
4. Ficheros de ayuda. Formatos. Herramientas para generarlos	7
4.1. Formatos	7
4.2. Herramientas para generar ficheros de ayuda	8
4.3. Generación de un sistema de ayuda con JavaHelp	8
4.4. Ayuda genérica y sensible al contexto	11
5. Tablas de contenidos	13
6. Flutter	14
6.1. Adoptando la filosofía de documentación de Flutter	14
6.1.1. El enfoque de las 5 W	17
6.1.2. DartDoc	17

Objetivos

- ☒ Conocer los diferentes sistemas de generación de ayudas que existen.
- ☒ Saber cuáles son los procedimientos para generar ayudas en los formatos habituales y de manera sensible al contexto.
- ☒ Comprender la importancia de la correcta documentación de aplicaciones.
- ☒ Realizar manuales de usuario y guías de referencia, así como manuales de instalación, configuración y administración, identificando claramente las diferencias existentes entre los mismos.

Capítulo 1. ¿Por qué es importante documentar?

Cuando se habla de **documentación de aplicaciones** software nos referimos a todos los elementos que detallan y aclaran las características de una aplicación, y que pueden ser necesarios para su uso o modificación. En concreto, se trata de ficheros de ayuda, manuales y demás guías de uso y/o mantenimiento, ya sean dirigidas para los usuarios de la aplicación, para equipos de soporte o incluso para otros diseñadores.

Para un sistema software es especialmente importante contar con una documentación completa y que sea sencilla de consultar. Los motivos para ello son varios:

- Facilitar su comprensión y posibilitar su uso por parte del usuario, independientemente de cuál sea su perfil.
- Facilitar el mantenimiento de la aplicación.
- Facilitar el desarrollo de mejoras por otros equipos de trabajo.
- Facilitar que el sistema pueda ser comprendido por otros



Figura 1. Menú de ayuda de Word

A pesar de tratarse de un trabajo que puede resultar tedioso, se le debe otorgar toda la importancia que tiene. No es suficiente con desarrollar aplicaciones funcionales y atractivas, sino que, además, estas deben contar con una documentación adecuada. Se trata de una tarea tan importante como la de realizar el propio código del programa.

Capítulo 2. Tipos de documentación

A grandes rasgos, podemos dividir la documentación de aplicaciones software en dos grupos: documentación de pruebas y documentación técnica.

2.1. Documentación de pruebas

Documentar las pruebas realizadas sobre un programa determinado es fundamental para detectar y corregir posibles errores. Podemos distinguir a su vez dos tipos:

a) **Documentación de entrada:** en la que se especifican los escenarios de prueba y se detallan los procedimientos de las mismas. b) **Documentación de salida:** se trata de los informes resultantes de aplicar las pruebas sobre el programa definidas en el punto anterior.

Estos informes recogerán el histórico de pruebas realizadas, documentándose así cada problema que haya podido surgir y su posterior corrección.

2.2. Documentación técnica

Pertenece a este grupo el resto de documentación: guías, hojas de especificaciones, manuales. Al igual que en el caso anterior, puede dividirse en dos grupos:

a. **Documentación interna:** se trata de los comentarios incluidos por el desarrollador en el código, que deben describir distintos aspectos sobre el mismo.

Se debe incidir en la importancia de realizar un programa ordenado y claro, en el que los comentarios ayuden a entender el código, pero este ya de por sí debe ser claro. En este ámbito, debe tenerse en cuenta lo siguiente:

- Incluir comentarios al comienzo de módulos.
- Comentar variables, constantes, procedimientos y funciones.
- Incluir comentarios introductorios sobre bloques de código, funciones o módulos.
- No comentar lo obvio, ya que el exceso de comentarios puede ser contraproducente.

b. **Documentación externa:** en este caso, suele tratarse de un manual técnico (orientado a programadores para facilitar el mantenimiento y desarrollo de la aplicación a futuro) y de un manual y/o guía de usuario (para facilitar su uso).

Capítulo 3. Manuales y guías

Existen diferentes tipos de manuales y guías, diferentes entre sí en función del contenido de cada uno y de su manera de utilización. Estos documentos pueden crearse en diferentes formatos y/o soportes: papel, formato electrónico, web de ayuda, presentación, vídeo, combinación de varios, según cuál sea la complejidad de la aplicación resultante y del perfil de usuario a quien vaya dirigido.



En el momento de realizar un manual o guía es muy importante pensar como lo haría el usuario que utilice la aplicación para entender qué problemas puede encontrarse y de qué forma podemos ayudarle a resolverlos. Este es el fin principal de los manuales y las guías de usuario.

3.1. Manual y guía de usuario

El manual de usuario contiene toda la información necesaria con el fin de hacer más fácil el uso y la comprensión del programa. Los ámbitos de aplicación son varios y diversos: formación del usuario, guía de consulta antes dudas, ayuda para detectar y corregir errores, etc.

Es cierto que no existe ninguna norma acerca de cómo se debe elaborar, aunque hay ciertos patrones y usos que se han de seguir. En primer lugar, el documento en sí mismo debe ser claro y atractivo desde el punto de vista del usuario, que a la postre es quien va a utilizarlo. Para ello, puede resultar aconsejable utilizar imágenes y gráficos para ayudar a entender el texto, utilizar un vocabulario claro y conciso, ejemplos prácticos, etc.

Una posible estructura del manual de usuario puede ser la siguiente:



Figura 2. Posible estructura de manual de usuario de una aplicación.



Lo estructura de un manual de usuario dependerá en gran medida del objetivo que se pretenda conseguir, así como del tipo de usuario que vaya a utilizarlo.

3.2. Manual y guía de explotación

El manual y guía de explotación contiene la información necesaria para utilizar y explotar la aplicación. Es decir, va muy orientado a la instalación, configuración y puesta en marcha, por lo que, evidentemente, irá muy ligado al contexto en el que se vaya a realizar la instalación (tipo de organización, requerimientos, número de usuarios).

Aunque en este caso tampoco hay una norma para su elaboración, debe contener al menos los siguientes puntos:

- Requerimientos mínimos del sistema en cuanto a procesador, memoria, espacio libre.
- Proceso de instalación:
 - Necesidad o no de preparación previa, si es necesaria alguna tarea para ello (por ejemplo, habilitar la conexión a Internet, activar o desactivar permisos).
 - Opciones de instalación: a través de USB, red, etc.
 - Procedimiento de instalación paso a paso, detallando y explicando sus diferentes opciones.
- Actualizaciones del sistema y copias de seguridad.
- Glosario.
- Índice.

En función de la complejidad de la aplicación, puede dividirse en manual de instalación y manual de configuración.

3.3. Guía rápida y guía de referencia

De manera adicional a los manuales estudiados anteriormente, la documentación de una aplicación puede completarse con este tipo de guías, en función del caso en concreto. Sus características son las siguientes:

1. **Guía rápida:** se orienta a usuarios del sistema y/o encargados de mantenimiento. En función de la complejidad de la aplicación puede ser necesario realizar varias, cada una con distinta temática. Las guías rápidas aportan información muy concreta y muy diferente, relacionada con diversos procedimientos o campos de una aplicación.
2. **Guía de referencia:** al contrario que en el caso de las guías rápidas, estas suelen estar pensadas para usuarios que tienen un mayor nivel de conocimiento sobre el uso de la aplicación, así como una mayor experiencia. Por ello, es habitual que contengan información relacionada con aspectos más técnicos: tipos de mensajes de error y su causa, tipos de datos de entrada que le son permitidos a la aplicación, tipos de comandos, etc.

Capítulo 4. Ficheros de ayuda. Formatos. Herramientas para generarlos

Un fichero es un elemento que puede contener varios tipos de información en diferentes formatos, ya sea en soporte físico o en soporte digital, que suele ser el procedimiento habitual en el campo de las aplicaciones informáticas. De esta forma, un fichero de ayuda es un documento que contiene información de ayuda sobre un determinado campo. Un ejemplo de fichero de ayuda podría ser un manual de uso de una herramienta concreta, dirigido a los usuarios de la misma. Los ficheros de ayuda están formados por dos partes diferenciadas:

- **Mapa del fichero:** se trata de un apartado que facilita la navegación por el fichero, que identifica claramente el contenido de este a través de identificadores.
- **Vista de información:** muestra al usuario contenidos en forma de glosario o tabla de contenidos.

4.1. Formatos

En el siguiente cuadro se muestran algunos de los principales formatos de ficheros de ayuda, junto con las principales características de estos:

Formato	Características	Plataforma
CHM	<ul style="list-style-type: none">• Generado a partir de HLP.• Utiliza HTML para generar la ayuda.• Enlaces mediante hipervínculos a la tabla de contenido. - Permite fusionar varios ficheros de ayuda.• Puede ser creado a partir de HTML Help Workshop.	Microsoft Windows
HLP	<ul style="list-style-type: none">• Puede incluir tabla de contenido en fichero .cnt.• Incluye información extra en fichero .gid. - Utiliza ficheros RTF para generar la ayuda.• Necesita compilación (por ejemplo mediante HTML Help Workshop).	Windows

Formato	Características	Plataforma
HPJ	<ul style="list-style-type: none"> • Contiene tabla de contenido (archivo .cnt). • Se crea mediante herramienta tipo Help Workshop. 	Windows
IPF	<ul style="list-style-type: none"> • Similar a HTML. • Se utiliza para ayuda en línea o web. • Es necesario compilar para generarlo. 	Web (ayuda en línea)
JavaHelp	<ul style="list-style-type: none"> • Implementado en Java, por lo que es utilizado habitualmente Varias para implementar la ayuda de aplicaciones desarrolladas en este lenguaje. 	Multiplataforma



La elección del formato depende de cómo se implemente la ayuda de la aplicación en sí misma. Es importante tener claro este aspecto previamente a la realización del contenido para intentar optimizarlo.

4.2. Herramientas para generar ficheros de ayuda

A continuación, se detallan las principales características de algunas herramientas que se pueden utilizar para generar ficheros de ayuda:

- **Help Work Shop**: posibilita crear ficheros de ayuda en entornos Windows. Es una aplicación que consta de un editor de imágenes, un administrador de proyectos y un compilador.
- **HelpMaker**: es una herramienta de carácter gratuito que cuenta con múltiples opciones para personalizar los ficheros. Utiliza un editor de texto para facilitar la generación de los ficheros de ayuda.
- **JavaHelp**: Java Help por sí mismo es un sistema de creación de ficheros de ayuda y, además, un formato de ficheros de ayuda propiamente dicho. Se caracteriza por:
 - Utilizar Java para su implementación.
 - Utilizarse en aplicaciones Java.
 - Ser independiente de la plataforma.
 - Facilitar la realización de ayuda online.
- **Shalom Help Maker**: se trata de otra herramienta gratuita, que utiliza un editor de texto para la creación de ficheros de ayuda, de manera similar a HELPMAKER. Permite incluir enlaces externos, insertar o crear imágenes, crear índices, etc.
- **DartDoc**: Similar a Java Help, es un sistema de creación de ficheros de ayuda que genera ficheros HTML y Markdown. Se caracteriza por:
 - Utilizarse en aplicaciones Dart y Flutter.
 - Ser independiente de la plataforma.
 - Facilitar la realización de ayuda online.

4.3. Generación de un sistema de ayuda con JavaHelp

Para crear un sistema de ayuda con *JavaHelp* se deben seguir unos pasos concretos, indicados en el siguiente diagrama:

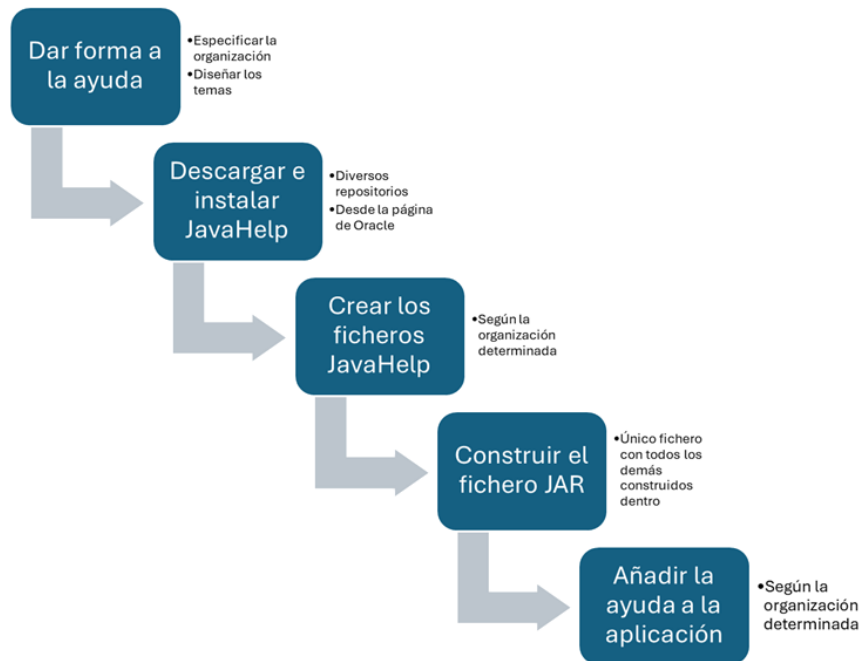


Tabla 1. Ficheros JavaHelp y su uso

Fichero	Extensión	Utilización
Índice	XML	Incluye la distribución del sistema de ayuda.
Map	JHM	Asocia elementos (imágenes, ficheros HTML, etc.) del fichero HTML con un identificador. Tabla de contenidos XML Incluye el contenido de la ayuda y su distribución.
Helpset	HS	Contiene la información necesaria para que el sistema de ayuda se ejecute.
Temas o <i>topics</i>	HTML	Para crearlos se puede utilizar cualquier herramienta para generar HTML. Contienen la información de ayuda como tal, debiéndose realizar uno por cada tema o asunto de ayuda. Se muestra uno de ellos por pantalla. Deben tener organización jerárquica.
Base de datos de búsqueda	N/A	Se debe utilizar la herramienta <i>jhindexer</i> para generarla.

El directorio con los ficheros creados debe seguir una estructura similar a la siguiente:

```

/help
- Índice.xml
- Map.jhm
- Helpset.hs
- TablaDeContenidos.xml
/html
- Tema1.html
- Tema2.html
/JavaHelpSearch
  
```

Es fundamental que los ficheros HTML estén en el directorio `./help`, ya que, en caso contrario, la aplicación no funcionará.

Para generar fichero *JavaHelp* de prueba vamos a seguir paso a paso lo indicado anteriormente. Para visualizar un

ejemplo, nos crearemos una ayuda ficticia basada en tres páginas HTML:

- Página principal a modo de introducción.
- Página con la ayuda necesaria para el “tema 1”.
- Página con la ayuda necesaria para el “tema 2”.

Veamos el procedimiento paso a paso.

1. Dar forma a la ayuda

En este primer paso, construiremos la ayuda de nuestra aplicación, creando tantos ficheros HTML como necesitemos. Es importante construir adecuadamente la estructura de la ayuda, ya que es fundamental que sea clara y concisa para facilitar la localización de la información por parte del usuario.

En nuestro ejemplo, al tratarse de un caso sencillo, únicamente hemos creado dos temas de ayuda, junto a una página principal. El código HTML del fichero principal es el siguiente:

```
<DOCTYPE html>
<html>
<head>
  <meta charset=utf-8 />
  <title>Esta es nuestra ayuda JavaHelp de prueba </title>
</head>
<body>
  <h1>Esta es una ayuda creada con JavaHelp</h1>
  <h2>Esta ayuda la hemos creado a modo de prueba</h2>
  <p>Para ello podemos utilizar un generador de código HTML cualquiera</p>
  <p>Podemos también añadir imágenes o incluso referenciar
entre sí las diferentes páginas de la ayuda</p>
  
  <p>Este es el enlace al <a href="tema1.html">Tema 1</a> de ayuda</p>
  <p>Este es el enlace al <a href="tema2.html">Tema 2</a> de ayuda</p>
</body>
</html>
```

2. **Descargar e instalar *JavaHelp*** Los ficheros necesarios para poder trabajar con *JavaHelp* los podemos encontrar en diversos repositorios. Es recomendable descargarlos desde un sitio seguro y oficial, como puede ser la página de Oracle. Una vez descargados, seguiremos el proceso de instalación hasta completarlo.

3. Crear ficheros JavaHelp

- *Índice*. Será un fichero con extensión **.xml** en el que se detallarán todas las páginas que tiene nuestra ayuda (tres en nuestro ejemplo). El fichero tendrá la siguiente apariencia: se compone de una etiqueta principal llamada **<index>**, que a su vez contiene varias subetiquetas llamadas **<indexitem>**, que hacen referencia a cada elemento de ayuda que compondrá el Índice. De esta forma, se mostrará el texto que se incluya en “text”, mientras que el campo “target” hace referencia a la clave o ID. Esta clave debe coincidir con la indicada en el fichero map dentro del mismo campo “target”.
- *Fichero map*. En este caso, hemos creado un “mapID” por cada fichero HTML, con el fin de poder referenciarlo. Evidentemente, se debe indicar la ruta en la que está dicho fichero. El fichero se puede crear con cualquier editor de texto, salvándolo en formato **jhm**.
- *Helpset*. Se podría decir que es el fichero principal, que debe construirse en formato **.xml**, aunque se guardará con extensión **.hs**. Un ejemplo de fichero *HelpSet* para nuestro caso puede ser el siguiente, aunque habría diversas posibilidades: el contenido del fichero se encuentra dentro de la etiqueta **<helpset>** que define el principio y final del mismo. A partir del comienzo, podemos definir tres tipos de etiquetas:

<title>: será el título que aparezca en la ventana de ayuda. **<maps>**: indica la ubicación del fichero map. **<homeID>**: fichero que aparece por defecto al ejecutar la ayuda. **<mapref>**: nombre del fichero map. **<view>**: uno por cada elemento de la ayuda. En este caso, Búsqueda, Índice y Tabla de Contenidos. El orden de los mismos en el fichero determina cómo aparecen en la ayuda **<name>**: nombre del view. **<label>**: etiqueta que se mostrará en relación con este elemento. **<type>**: muestra la clase de *JavaHelp* relacionada con el contenido. **<data>**: indica el fichero que está relacionado con esa parte, de los que se crearon anteriormente. En este caso, en el campo de Búsqueda, se incluye el subdirectorío que se crea a partir de la herramienta **jhindexer.jar**, como se verá a continuación.

- *Tabla de contenidos*. Detallará la organización de los ficheros de nuestra ayuda, que en nuestro ejemplo es muy sencilla. Deberá guardarse en formato .xml, con cualquier editor de texto. La etiqueta **<toc>** describe el elemento en su conjunto, que a su vez se divide en varios **<tocitem>**. Estos elementos pueden anidarse entre sí. Dentro de los mismos, de igual forma que en el fichero Índice, se mostrará el texto que se incluya en “text”. Así mismo, el “target” hace referencia a la clave o ID, que debe coincidir con la indicada en el fichero map.
 - *Base de datos de búsqueda*. Para crearla es necesaria la ejecución del comando **jhindexer.jar**, que invoca a una de las herramientas de *JavaHelp*. Para ello, basta con ejecutar la orden “jhindexer.jar html”, cambiando previamente el PATH al directorio en el que tenemos los ficheros de ayuda, es decir, el directorio *help*. El parámetro html hace referencia al directorio en el que están almacenados los ficheros de ayuda. La creación de esta base de datos permitirá contar con una pestaña de búsqueda, que hará posible navegar por la ayuda de una manera más ágil y eficaz.
4. **Construir fichero JAR** Para construir los ficheros JAR será necesario ejecutar: **Jar -cvf help.jar**. Mediante esta orden se crean ficheros JAR independientes. Se debe ejecutar desde el directorio *help* en el que están los ficheros de ayuda.
5. **Añadir ayuda a la aplicación** Para añadir la ayuda que hemos creado a cualquier aplicación Java, se deben incluir los siguientes apartados en el mismo:
- Paquete javax.help.
 - Clase HelpSet, que posibilita el uso de ficheros y datos de la ayuda.
 - Clase HelpBroker, que ayuda a visualizar el contenido de la ayuda.
 - Método createHelpBroker.
 - Método findHelpSet.
 - Método enableHelpOnButton, para poder visualizar la ayuda al pulsar un menú.
 - Método enableHelp, que al hacer clic sobre un elemento indicará el tema de ayuda.
 - Método enableHelpKey, que activa la tecla de ayuda sobre un elemento.

4.4. Ayuda genérica y sensible al contexto

Aunque a priori pudiera parecer una distinción evidente, debemos tener en cuenta las diversas formas en las que puede presentarse y/o elaborar una documentación de ayuda relacionada con una aplicación:


Tabla 2. Características de los diferentes tipos de ayuda


Ayuda genérica	Ayuda sensible al contexto
<ul style="list-style-type: none">• Contiene toda la información de ayuda sobre una determinada aplicación, dividida por temas o campos.• Generalmente presenta un índice o menú.• Permite navegar o buscar entre los elementos que la forman.	<ul style="list-style-type: none">• Se trata de ayuda particularizada para un tema o acción concreta.• Puede presentarse de diferentes maneras: ayuda en línea, breve detalle del uso de un menú en el programa.


Capítulo 5. Tablas de contenidos


Las tablas de contenidos nos ayudan a modelar tanto el contenido como la estructura de un documento determinado. La información estará organizada en forma de esquema, con diversos niveles entre los que se cabe destacar títulos y subtítulos. Lógicamente, el diseño de las mismas depende de quién haga el desarrollo, así como de otros muchos factores (objetivo, usuario al que se dirigen, etc.). Sin embargo, existe una serie de características comunes a todas ellas:

- Es posible o no que muestren el número de páginas que contienen, depende del diseño que se realice sobre las mismas. Se trata de un elemento completamente opcional.
- Las tablas de contenidos habitualmente contienen enlaces directos a cada título o subtítulo, que apunta al contenido de los mismos. Así se consigue una navegación y utilización más fluida y sencilla para el usuario.
- Suelen situarse al comienzo de cada documento, de manera similar al índice de los libros.
- Aunque puede resultar sencillo, su proceso de elaboración no es elemental, ya que es necesario realizar un análisis previo y pormenorizado de toda la documentación que contienen. Además, es necesario que los títulos sean lo más claros e intuitivos posible. Es también importante que no haya información duplicada en varias partes del documento.
- Como resumen, en la siguiente figura se muestran algunos consejos que se han de tener en cuenta para elaborar una tabla de contenidos clara y de calidad, que sea útil para el usuario:

 Seleccionar todos los temas y subtemas que tendrá la tabla.

 Numerar temas y subtemas según el patrón de diseño.

 Crear la tabla de contenido, procurando añadir los enlaces necesarios en la misma.

 Actualizar y modificar la tabla cada vez que se necesite.

Capítulo 6. Flutter

6.1. Adoptando la filosofía de documentación de Flutter

La metodología de documentación de Flutter está meticulosamente estructurada y da prioridad a la información clara, completa y práctica. Las características que he observado en la documentación de Flutter ejemplifican lo que a menudo se considera una «buena» documentación. Por lo tanto, cada elemento que comento puede servir como guía para elaborar una documentación eficaz.

Profundicemos en los componentes básicos de esta filosofía:

1. **Cobertura exhaustiva:** Flutter insiste en documentar todos los miembros públicos de la biblioteca, utilizando herramientas como DartDoc para Dart y tecnologías similares para otros lenguajes. Esto garantiza que todos los aspectos de la API estén bien explicados y sean accesibles. Si una decisión requiere documentación sobre miembros privados, es una buena idea escribir documentación para ellos.
2. **Actualizaciones inmediatas de la documentación:** si surge una pregunta durante el desarrollo, añade la respuesta a la documentación donde se buscó por primera vez. Esto crea un documento dinámico que evoluciona con la comprensión del proyecto.
3. **Claridad por encima de la tradición:** la filosofía da prioridad al código y la documentación claros y autoexplicativos (incluidos todos los procesos y conversiones) por encima de la dependencia excesiva de la «tradición oral». Este enfoque hace que el marco sea accesible para los nuevos colaboradores sin conocimientos previos.
4. **Calidad por encima de cantidad:** Flutter aboga por una documentación significativa. En lugar de llenar páginas con descripciones genéricas, se centra en proporcionar información valiosa, especialmente para los elementos que no se explican por sí mismos.

```
// MALO:

/// El color de fondo.
final Color backgroundColor;

// BUENO:

/// El color con el que rellenar el círculo.
///
/// Cambiar el color de fondo hará
```

```
/// que el avatar lance una nueva animación con el nuevo color.
final Color backgroundColor;
```



Documentar tu código Dart/Flutter es muy fácil, solo tienes que añadir tus frases documentadas después de *///* y, en un IDE, se mostrarán cada vez que pases el cursor por encima de esa clase o función.

5. **Documentación específica:** La documentación está diseñada para responder a preguntas reales que puedan tener los usuarios. Evita descripciones genéricas y proporciona información detallada y rica en contexto, especialmente para aspectos complejos o poco intuitivos.
6. **Aprendizaje por capas:** La documentación de Flutter trata cada parte como si fuera la primera vez que el lector se encuentra con un concepto concreto. Evita dar por sentados conocimientos previos y se asegura de que los términos se definan o se vinculen a la documentación básica.

```
// BUENO:

/// Un objeto que representa una secuencia
/// de operaciones gráficas registradas.
///
/// Para crear una [Picture], utilice un [PictureRecorder].
///
/// Una [Picture] se puede colocar en
/// una [Scene] utilizando un [SceneBuilder], a través del
/// método [SceneBuilder.addPicture].
/// Una [Picture] también se puede
/// dibujar en un [Canvas],
/// utilizando el método [Canvas.drawPicture].
///
/// Véase también:
///
/// * [FooBar], que es otra forma de pelar naranjas.
/// * [Baz], que quuxea el wibble.
class abstracta Picture // ...
```

7. **Ejemplos prácticos y casos de uso:** Proporcionar código de muestra, ejemplos de aplicaciones y ejemplos prácticos es un aspecto fundamental de la estrategia de documentación de Flutter. Estos ejemplos ayudan a los usuarios a comprender cómo implementar y utilizar diversas funciones en situaciones reales.

En combinación con la posibilidad de ejecutar Flutter en la web, esto permite al equipo de Flutter disponer de aplicaciones autónomas con ejemplos de código que se ejecutan en el contexto de la documentación que estás leyendo para mostrar y demostrar la clase en sí de forma interactiva.

```
/// Especifica cómo se cerró un [MaterialBanner].
///
/// La función [ScaffoldMessengerState.showMaterialBanner]
/// devuelve un
/// [ScaffoldFeatureController].
/// El valor del controlador cerrado correctamente
/// es un Future que se resuelve en
/// un MaterialBannerClosedReason. Las aplicaciones que necesitan
/// saber cómo se cerró un [MaterialBanner]
/// pueden utilizar este valor.
///
**/// Ejemplo:
```

```

///
/// ```dart
/// ScaffoldMessenger.of(context).showMaterialBanner(
///   const MaterialBanner(
///     content: Text("Mensaje..."),
///     actions: <Widget>[
///       // ...
///     ],
///   )
/// ).closed.then((MaterialBannerClosedReason reason) {
///   // ...
/// });
/// ```**
enum MaterialBannerClosedReason {

```

8. **Ayudas visuales:** Para los widgets y los elementos de la interfaz de usuario, Flutter incluye diagramas, capturas de pantalla y otras ayudas visuales en su documentación para dar una idea clara de lo que el usuario puede esperar visualmente del componente.

```

/// Widgets de Flutter que implementan Material Design.
///
/// Para utilizarlos, importe `package:flutter/material.dart`.
///
/// { @youtube 560 315 https://youtu.be/DL0Ix1lnC4w }
///
/// Véase también:
///
/// ... Más enlaces aquí ...
library material;

```

9. **Terminología canónica:** Flutter utiliza de forma coherente términos como «método» para los miembros de clase, «función» para los cierres invocables fuera de las clases, «parámetro» para las variables en las firmas de cierre y «argumento» para los valores pasados a los cierres, y utiliza «llamar» en lugar de «invocar» y «variable miembro» en lugar de «variable de instancia».
10. **Refactorización para mayor claridad:** Si la documentación se vuelve demasiado compleja o enrevesada, es preferible refactorizar el código para que resulte más comprensible, en lugar de complicar en exceso la documentación.
11. **Gramática correcta:** la documentación debe tener una gramática adecuada en el idioma elegido, por ejemplo, un inglés correcto. También es importante mantener un tono orientativo y no prescriptivo en la documentación. Es preferible utilizar la voz pasiva y sugerencias como «considerar» en lugar de órdenes directas. Evite utilizar «usted» o «nosotros» y hacer juicios de valor. Reconozca que los lectores provienen de diversos orígenes y niveles de conocimiento; evite dar a entender simplicidad o facilidad, lo que puede desanimar a aquellos que encuentran el material difícil.

```

// MALO

/// [foo] no debe ser nulo.

// BUENO

/// El argumento [foo] no debe ser nulo.

```

12. **Evitar la prosa vacía:** La filosofía de la documentación de Flutter hace hincapié en la concisión, evitando la verbosidad innecesaria en favor de explicaciones claras y concisas.

```
// MALO:

/// Nota: Es importante
/// tener en cuenta que, en
/// ausencia de un valor explícito,
/// esta propiedad toma por defecto el valor 2.

// BUENO:

/// El valor predeterminado es 2.
```

En línea con la filosofía de documentación de Flutter, se debe hacer hincapié en la claridad, la exhaustividad y la precisión. Un buen documento debe incorporar estos principios para ser verdaderamente informativo, fiable y fácil de usar.

6.1.1. El enfoque de las 5 W

A la hora de crear una buena documentación, es fundamental responder a las 5 preguntas básicas. Aunque la mayoría de la documentación suele responder a la pregunta «cómo», si observamos ejemplos de diversos programas informáticos, como Flutter, queda claro que es necesario adoptar un enfoque integral.

- **Quién (Who):** identifique el público destinatario. Comprender quién utilizará el documento ayuda a adaptar el contenido a sus conocimientos y necesidades.
- **Qué (What):** definir claramente de qué trata el documento. Esto incluye el alcance, las funcionalidades y las características del software o código que se documenta.
- **Dónde (Where):** proporcionar el contexto en el que se aplica el documento o su información. Puede tratarse de entornos, sistemas o escenarios específicos en los que se utiliza el software.
- **Cuándo (When):** describa el momento o las condiciones adecuadas para utilizar la información. Esto podría incluir historiales de versiones, calendarios de actualización o plazos relevantes.
- **Por qué (Why):** explique el propósito o la razón de ser del tema de la documentación. Esto incluye el razonamiento detrás de las decisiones de diseño, la lógica del código y los objetivos generales del software.

Busque documentos completos, meticulosamente precisos y articulados con claridad para comunicar su mensaje y atraer a su público de manera eficaz.

6.1.2. DartDoc

Flutter utiliza principalmente **dartDoc** para generar su documentación. Aunque existen otras herramientas disponibles, **dartDoc** es muy recomendable tanto para aplicaciones como para bibliotecas en el ecosistema Flutter. DartDoc es una herramienta robusta que facilita la creación de documentación de alta calidad. Para mejorar tus habilidades de documentación, lee [DartDoc](#) en pub.dev y considera leer [Effective Dart: Documentation](#).

DartDoc es compatible con Markdown a través del paquete **markdown**, lo que permite el formato de texto enriquecido en la documentación. Un aspecto interesante de DartDoc en Flutter es su integración con el paquete **snippets**. Este paquete es fundamental para crear y gestionar bloques de código dentro de la documentación de la API de Flutter, ya que proporciona ejemplos útiles y puntos de partida para utilizar las API de Flutter.

Por ejemplo, considera este fragmento de código de la documentación de Flutter:

```

/// {@tool snippet}
///
/// Si el avatar va a tener una imagen,
/// esta debe especificarse en la propiedad
/// [backgroundImage]:
///
/// ```dart
/// CircleAvatar(
///   backgroundImage: NetworkImage(userAvatarUrl),
/// )
/// ```
/// {@end-tool}

```

Este fragmento utiliza una plantilla especial (*snippet.html*), que se encuentra en [dev/snippets/config/skeletons/snippet.html](https://github.com/flutter/flutter/blob/master/dev/snippets/config/skeletons/snippet.html) en el repositorio de Flutter, para generar estos bloques de código informativos.

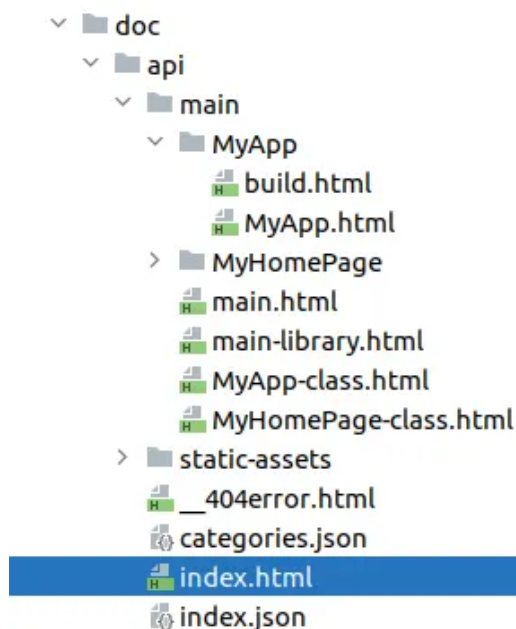
Flutter te proporciona un comando para crear esta documentación para tu código base Dart en solo un minuto. Utiliza estos comandos para generar uno por ti mismo:

```

flutter pub global activate dartdoc
flutter pub global run dartdoc .

```

El punto (.) significa que se debe documentar todo el código. Si deseas un archivo específico, puedes sustituir el punto (.) por el nombre del archivo específico.



Se generará un directorio *doc* en tu código base. Puedes abrir *index.html* y echar un vistazo a la documentación generada.