

# Lumino Docs

La documentación en un proyecto de software es un componente clave que asegura la comprensión, mantenimiento y evolución del sistema a lo largo de su ciclo de vida. Sirve como puente entre los equipos de desarrollo, los “stakeholders” y los usuarios finales, proporcionando una descripción clara de los objetivos, funcionalidades y arquitectura del proyecto. Sin una documentación adecuada, los equipos podrían enfrentarse a desafíos como malentendidos, errores en la implementación y dificultades para dar soporte o realizar actualizaciones futuras.

Existen muchas herramientas para documentar proyectos de software, pero dentro del mundo Python una de las más usadas es [MkDocs](#). En realidad no es más que un generador estático pero que permite estructurar la documentación de forma muy clara y escribirla en formato Markdown.

Uno de los proyectos más interesantes que se han desarrollado sobre este ecosistema es [Material for MkDocs](#). Tiene un aspecto visual muy potente y además ofrece gran cantidad de funcionalidades. Pero existe una secuela de este proyecto que es aún más interesante: [Zensical](#).

La idea es documentar el proyecto *Lumino* (`pypas get lumino`) utilizando [Zensical](#).

## 1. Contenido

El grupo debe incluir en la documentación todos los puntos que se indican a continuación:

### Introducción

- **Propósito del documento:** Explicar los objetivos y el alcance de la documentación.
- **Visión general del proyecto:** Breve descripción del software, su objetivo principal y el problema que soluciona.
- **Audiencia objetivo:** Definir quiénes usarán esta documentación (desarrolladores, testers, usuarios finales, etc.).

### Requisitos del proyecto

- **Requisitos funcionales:** Funcionalidades clave que el sistema debe cumplir.
- **Requisitos no funcionales:** Rendimiento, seguridad, escalabilidad, usabilidad, etc.
- **Restricciones:** Factores técnicos, legales o de negocio que afectan el desarrollo.
- **Casos de uso:** Diagramas o descripciones de los escenarios de interacción con el sistema.

## Diseño del sistema

- **Arquitectura del sistema:** Diagrama y descripción de los componentes principales.
- **Modelo de datos:** Esquema de base de datos y relaciones entre entidades.
- **Diagramas:** Diagramas de clases, de secuencia, de actividades...
- **Decisiones de diseño:** Justificación de las tecnologías, patrones y enfoques elegidos.

## Implementación

- **Estructura del código:** Organización del repositorio y convenciones usadas.
- **Tecnologías y herramientas:** Lenguajes, frameworks, bibliotecas, y entornos de desarrollo.
- **Instrucciones de configuración:** Pasos para instalar dependencias y configurar el entorno.

## Pruebas

- **Estrategia de pruebas:** Métodos utilizados (unitarias, de integración, de aceptación).
- **Casos de prueba:** Ejemplos específicos y sus resultados esperados.
- **Cobertura de pruebas:** Indicadores de qué partes del código están siendo probadas.
- **Automatización:** Herramientas y scripts utilizados para pruebas automáticas.

## Despliegue

- **Entorno de producción:** Descripción de los servidores, servicios y configuración necesaria.
- **Proceso de despliegue:** Pasos detallados para implementar el software.
- **Planes de recuperación:** Estrategias para manejar errores durante el despliegue.

## Manuales de usuario

- **Instrucciones básicas:** Cómo instalar, configurar y usar el software.
- **Guías avanzadas:** Funcionalidades específicas o tareas complejas.
- **Solución de problemas:** Respuestas a preguntas frecuentes y resolución de errores comunes.

## Mantenimiento y actualización

- **Política de mantenimiento:** Frecuencia de actualizaciones y soporte.
- **Registro de cambios (Changelog):** Historial de versiones y mejoras realizadas.
- **Plan de escalabilidad:** Posibles mejoras para soportar más usuarios o nuevas funcionalidades.

## Conclusiones y futuro

- **Estado actual del proyecto:** Resumen del progreso y alcance cumplido.
- **Futuras actualizaciones:** Funcionalidades o mejoras planificadas.
- **Lecciones aprendidas:** Reflexión sobre el desarrollo y recomendaciones para futuros proyectos.

## 2. Requerimientos

Para que puedas trabajar con normalidad en este ejercicio debes tener instalado en tu máquina:

1. `uv` → Gestor de paquetería y proyectos Python.
2. `just` → Lanzador de comandos como recetas.

## 3. Puesta en marcha

Se proporciona una *receta* `just` para la puesta en marcha del proyecto:

```
just setup
```

¿Qué ha ocurrido?

- Se ha creado un entorno virtual en la carpeta `.venv`
- Se han instalado las dependencias del proyecto.

En la estructura del proyecto es importante identificar lo siguiente:

<code>docs/</code>	Carpeta con ficheros <i>Markdown</i> para documentar el proyecto
<code>zensical.toml</code>	Fichero de configuración principal del proyecto <i>Zensical</i>

## 4. Flujo de trabajo

Para trabajar con el proyecto habrá que ir incorporando contenido en `docs/` y configuraciones en `zensical.toml`.

Se proporciona una *receta* `just` para renderizar el proyecto en desarrollo:

```
just runserver
```

La receta anterior también se puede lanzar directamente con `just` ya que es la receta *por defecto*. Podrás visualizar el proyecto en tu navegador accediendo a: <http://localhost:8000>

Se proporciona una *receta* `just` para construir el proyecto:

```
just build
```

La receta anterior crea una carpeta `site/` donde estarán los ficheros estáticos con el proyecto generado.