Carrera Programador full-stack

Documentación

Debate

Introducción

- La documentación es una parte fundamental del desarrollo de software. Un proyecto sin documentación adecuada puede volverse inentendible, dificultando su mantenimiento y escalabilidad. A través de la documentación, los desarrolladores pueden comprender la arquitectura del sistema, su funcionamiento y la razón detrás de ciertas decisiones de diseño.

Documentación

Carrera Programador full-stack

Documentación Funcional y No Funcional

Diferencia entre Documentación Funcional y No Funcional

- **Documentación Funcional:** Describe lo que hace el software y cómo interactúan los usuarios con él. Ejemplos:
  - Requisitos funcionales
  - Casos de uso
  - Diagramas de flujo
- **Documentación No Funcional:** Explica las características de calidad y restricciones del sistema. Ejemplos:
  - Requisitos de rendimiento
  - Seguridad y cumplimiento
  - Escalabilidad y mantenimiento

Carrera Programador full-stack

Tipos de Documentación en un Proyecto de Software

Tipos de Documentación en un Proyecto de Software

- **1. Documentación del Usuario**
- Guías de usuario
- Manuales de referencia
- Preguntas frecuentes (FAQs)

Tipos de Documentación en un Proyecto de Software

- **2. Documentación Técnica**
- Documentación de la arquitectura
- Diagrama de bases de datos
- Especificaciones de la API (Swagger, OpenAPI)
- Documentación del código (comentarios, generada con herramientas como JSDoc, Sphinx, Doxygen)

Tipos de Documentación en un Proyecto de Software

- **3. Documentación del Proceso**
- Historias de usuario
- Casos de uso
- Planes de prueba y resultados
- Documentación de despliegue (infraestructura, CI/CD)

Tipos de Documentación en un Proyecto de Software

- **4. Documentación de Gestión del Proyecto**
- Roadmaps y planes de desarrollo
- Documentación de decisiones arquitectónicas (ADR, Architecture Decision Records)
- Minutas de reuniones

Tipos de Documentación en un Proyecto de Software

- **5. Diferencia entre Documentación Funcional y No Funcional**
- **Documentación Funcional:** Describe lo que hace el software y cómo interactúan los usuarios con él. Ejemplos:
  - Requisitos funcionales
  - Casos de uso
  - Diagramas de flujo

Carrera Programador full-stack

Archivo Readme.md

README.md

- El **README.md** es el primer punto de contacto con un proyecto de software.
- Debe contener información clara y concisa para ayudar a los desarrolladores y usuarios a comprender el proyecto rápidamente.

¿Para qué sirve un README?

- Facilita la comprensión del proyecto.
- Explica cómo instalar y ejecutar la aplicación.
- Brinda detalles sobre el propósito y funcionalidades.
- Sirve como referencia para colaboradores y usuarios.
- Normalmente en formato Markdown (.md).

¿Por qué es importante?

- Facilita la comprensión del proyecto.
- Atrae colaboradores y usuarios.
- Proporciona instrucciones claras de instalación y uso.

Buenas prácticas

- Usa lenguaje claro y conciso.
- Agrega ejemplos y capturas de pantalla.
- Mantén el README actualizado.
- Incluye enlaces a documentación adicional.

Errores comunes a evitar

- No proporcionar ejemplos de uso.
- README desactualizado o incompleto.
- Falta de instrucciones de instalación.
- No incluir una licencia.

Recursos adicionales

- Guía de Markdown
  - https://www.markdownguide.org
- Ejemplo de README bien estructurado
  - https://github.com/matiassingers/awesome-readme

```
# Project Title

A brief description of what this project does and who it's for


## Demo

Insert gif or link to demo


## FAQ

#### Question 1

Answer 1

#### Question 2

Answer 2


## Feedback

If you have any feedback, please reach out to us at fake@fake.com


# Hi, I'm Katherine! 👋


## Other Common Github Profile Sections
👩‍💻 I'm currently working on...

🧠 I'm currently learning...

👯 I'm looking to collaborate on...

🤔 I'm looking for help with...

💬 Ask me about...

📫 How to reach me...

😄 Pronouns...

⚡ Fun fact...
```

Project Title

A brief description of what this project does and who it's for

Demo

Insert gif or link to demo

FAQ

Question 1

Answer 1

Question 2

Answer 2

Feedback

If you have any feedback, please reach out to us at fake@fake.com

Hi, I'm Katherine! 👋

Other Common Github Profile Sections

👩‍💻 I'm currently working on...

🧠 I'm currently learning...

👯 I'm looking to collaborate on...

🤔 I'm looking for help with...

💬 Ask me about...

📫 How to reach me...

😄 Pronouns...

⚡ Fun fact...

¿Que debe tener?

- **Título**:
  - Nombre del proyecto.
- **Descripción**:
  - Resumen del propósito y características.
- **Instalación**:
  - Pasos para instalar dependencias.
- **Uso**
  - Ejemplo de cómo ejecutar y usar la aplicación.
- **Contribuir**
  - Instrucciones para contribuir al desarrollo del proyecto.
- **Licencia**
  - Información sobre la licencia del software.

Tipos de licencias

- Al publicar un repositorio, es importante elegir una licencia que defina cómo otros pueden usar, modificar y distribuir el código. Algunos tipos comunes de licencias incluyen:
- **MIT**: Permite el uso, modificación y distribución con pocas restricciones, solo requiere atribución.
- **GPL (*General Public License*)**: Obliga a que cualquier modificación del código también se distribuya bajo la misma licencia.
- **Apache 2.0**: Similar a MIT, pero incluye una cláusula de protección de patentes.
- **BSD (*Berkeley Software Distribution*)** : Existen variantes de 2 y 3 cláusulas; permiten redistribución con atribución.
- **Creative Commons (*CC*)**: Se usa más para documentación y contenido que para código.
- **Proprietaria**: Restrictiva, limita el uso y modificación del código.
- Es recomendable incluir un archivo `LICENSE` en el repositorio para especificar claramente los términos de uso.

Carrera Programador full-stack

Toma de requerimientos

Introducción

- Para definir correctamente los requerimientos de un proyecto, es fundamental diferenciar si se trata de un desarrollo desde cero o una migración de un sistema existente.

Proyecto desde cero

1. ¿Cuál es el objetivo principal del sistema?
2. ¿Quiénes serán los usuarios finales y cuáles son sus necesidades?
3. ¿Cuáles son los requisitos funcionales clave?
4. ¿Existen requisitos de seguridad específicos?
5. ¿Cuáles son las plataformas de destino (web, móvil, escritorio)?
6. ¿Cuáles son las tecnologías preferidas?
7. ¿Se requiere integración con otras herramientas o servicios?
8. ¿Cuál es el presupuesto y el tiempo estimado de desarrollo?
9. ¿Cómo se manejará el despliegue y la infraestructura?
10. ¿Se necesita soporte y mantenimiento post-lanzamiento?

Migración

1. ¿Cuál es la razón principal de la migración?
2. ¿Qué sistema se está reemplazando y cuáles son sus limitaciones?
3. ¿Cuáles son las funcionalidades críticas que deben preservarse?
4. ¿Existen dependencias con otros sistemas que deban considerarse?
5. ¿Qué tecnologías usa el sistema actual y cuáles se desean utilizar en la nueva versión?
6. ¿Es necesario mantener compatibilidad con versiones anteriores?
7. ¿Cómo se manejarán los datos existentes en la migración?
8. ¿Existen riesgos conocidos asociados con la migración?
9. ¿Cuáles son los requerimientos de pruebas y validación?
10. ¿Cómo se planificará la transición para minimizar el impacto en los usuarios?

Conclusión

- Documentar un proyecto de software no es una tarea opcional, sino una necesidad.
- Facilita la colaboración, reduce la fricción en el mantenimiento y mejora la adopción del software por parte de otros desarrolladores y usuarios.
- Un README bien estructurado es la base mínima de documentación y debe complementarse con otros tipos de documentos técnicos y funcionales para garantizar un desarrollo eficiente y escalable.