

Live-Wiki

Ingeniería de Software aplicada a la
Narrativa: Un ecosistema "Text-First"

> *Lore-as-Code*

Jonathan Pineda Chacón

TFM: Máster en Desarrollo con IA

```
1 GEMINI.md file | 6 skills                                YOLO mode (ctrl + y to toggle)
* crea un personaje mouredev que es el director y porfesor del master de desarrollo
  con IA y es profesor de Jonathan
~/repos/live-wiki (main*)                                no sandbox                                Auto (Gemini 3) /model
```



Motivación y Problemática

Este proyecto nace de una necesidad personal para organizar elementos complejos de worldbuilding:

*"Necesitaba una herramienta flexible potenciada por IA para escribir mundos complejos, pero con la **seguridad técnica** de que la narrativa fuera coherente."*

⚠ El Problema Actual

Las herramientas tradicionales fallan en dos puntos críticos para un Game Dev:

- **Falta de Validación:** Permiten errores lógicos (ej. morir antes de nacer) que rompen la inmersión.
- **Fricción con la IA:** Bases de datos cerradas impiden que los Agentes lean/escriban lore estructurado.

✅ La Solución: Live-Wiki

Un ecosistema donde la narrativa se trata como código (**Lore-as-Code**):

- **Flexibilidad Total:** Markdown plano editable por humanos o IAs.
- **Seguridad Narrativa:** Tests automatizados aseguran la coherencia del mundo antes de cada "deploy".

Arquitectura: "Lore-as-Code"

1. Contexto Rico (Markdown)



Texto estructurado y legible por humanos y máquinas.

2. Comprensión Nativa (IA)



Los LLMs entienden la narrativa sin necesidad de esquemas SQL complejos.

3. Integridad Garantizada (Validación)



El sistema valida referencias y coherencia como si fuera código compilable.

Funcionalidades Principales

1. Gestión Estructurada

7 Tipos de Entidades (Character, Event, Place...) con **Frontmatter Estricto** para validar metadatos.

2. Lore Linter (Suite de tests propia)

Integridad Referencial: Detecta enlaces rotos.

Coherencia Narrativa: Impide paradojas temporales en la narrativa y diferentes agujeros de guión.

3. Experiencia "Wiki"

Generación automática de "Backlinks" y navegación SPA-like instantánea.

```
jowny@Jowny:~/repos/live-wiki$ npx vitest tests/lore-linter  
  
DEV v3.2.4 /home/jowny/repos/live-wiki  
  
✓ tests/lore-linter.spec.ts (2 tests) 109ms  
✓ tests/lore-linter-broken-refs.spec.ts (2 tests) 130ms  
✓ tests/lore-linter-schema.spec.ts (17 tests) 182ms  
  
Test Files 3 passed (3)  
Tests 21 passed (21)  
Start at 23:47:26  
Duration 842ms (transform 412ms, setup 0ms, collect 367ms,  
pass 163ms)  
  
PASS Waiting for file changes...  
press h to show help, press q to quit
```

Calidad y Testing



Estrategia en Tres Capas

1. Unitarios (Vitest):

Validan la lógica del *Lore Linter* y parsers.

2. E2E (Playwright):

Simulan navegación real, filtros y búsquedas.

3. Static Analysis (ESLint):

Garantizan código limpio y tipos seguros.



Pipeline de Seguridad

La rama main está blindada. No se puede desplegar si un solo test falla.



Sanitización

Prevención estricta de XSS en el renderizado de Markdown para evitar inyección de scripts maliciosos.

Potencial "Infinito": Desarrollo de UI Asistido

Estrategia y Ejecución

La IA no "inventa", **acelera** siguiendo patrones estrictos definidos en la **skill ui-developer**.

Ejemplo Real: "Vista de Personajes"

1. **Lógica (Business Logic):** Solicito una función TypeScript que filtre y agrupe los datos.
2. **Composición (Views):** Solicito el componente .astro visual usando la lógica anterior.

Supervisión: "Human-in-the-loop"

La IA propone, el ingeniero dispone. Nada se integra sin supervisión:

Rol Junior Dev:

Trato a la IA como un desarrollador junior; genera el borrador y el "boilerplate".

Code Review Humano:

Reviso, refactorizo y valido manualmente cada línea antes del commit.

Safety Net:

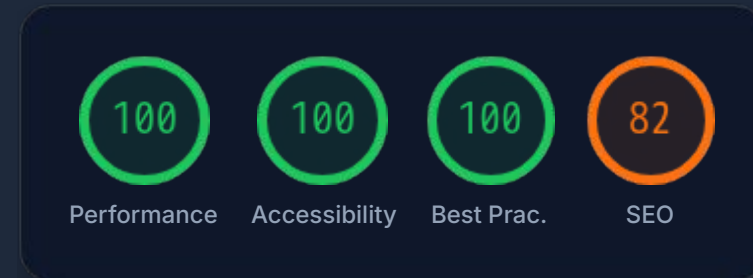
El compilador de TypeScript y los Tests E2E bloquearán cualquier fallo.

| Retos Superados y Resultados

🏆 Reto Técnico: Performance

Problema: Necesidad de interactividad sin sacrificar velocidad.

Solución: Arquitectura de *Islas de Astro*. Solo se hidrata el JS necesario.



[Verificar Test en PageSpeed](#)

🧠 Reto Lógico: Ambigüedad

Problema: Validar datos narrativamente inciertos.

Solución: Esquemas polimórficos (place: vs unknown).

Rendimiento, Accesibilidad y

Buenas Prácticas

Stack Tecnológico



Core & UI



Astro



TypeScript



Tailwind CSS

Renderizado estático con tipado estricto y sistema de diseño atómico.



Quality Assurance



Vitest



Playwright

Validación automática de coherencia narrativa (Linter propio) y tests E2E.



DevOps & Infra



GitHub Actions



Cloudflare Pages

Despliegue continuo en el Edge global sin mantenimiento de servidores.



IA Ready: Prompts y skills versionados como código en /skills.

| Demo y Código



Ver Demo

live-wiki.pages.dev/es



Repositorio

github.com/JownyDev/live-wiki

Conclusiones

1. El Verdadero Valor Diferencial

La arquitectura **Text-First** permite una **iteración exponencial** gracias a la comprensión nativa de la IA, mientras que el **Lore Linter** actúa como red de seguridad.

2. Sinergia Humano-IA

El texto plano es la interfaz definitiva para la colaboración con IA. "Lore-as-Code" permite que los agentes entiendan el contexto.

3. Escalabilidad

Sistema preparado para crecer indefinidamente sin costes de infraestructura (Jamstack + Cloudflare).

¡Muchas Gracias!

[¿Preguntas?](#)

 JownyDev

\$ close presentation

TFM - Máster en
Desarrollo con IA