Implementación de Event-Driven en PHP

Enlace del repositorio:

HTTPS://GITHUB.COM/DIEGOGALANCG20/DSS_INVESTIGACION2.GIT

Integrantes:

- Caleb Alejandro Peñate Deras – PD230166
- Diego Alberto Canizalez Galán – CG232300
- Camila Elizabeth Castillo Joya – CJ220498
- Daniel Adonay García Aguilar – GA232128

¿QUÉ ES EVENT-DRIVEN?

Programación donde el flujo depende de eventos (clics, señales, mensajes)

Usa callbacks y event handlers para reaccionar

Importancia en web:

- Interacción en tiempo real
- Escalabilidad
- Base de tecnologías modernas como Node.js, WebSockets, React

PRINCIPIOS DEL PARADIGMA

EVENT-DRIVEN

- Asincronía y No bloqueo: no detiene ejecución mientras espera
- Event Loop: bucle que atiende eventos cuando hay recursos disponibles
- Emisión y escucha de eventos: objetos emiten y escuchan eventos
- Escalabilidad y eficiencia: ideal para apps con muchas conexiones

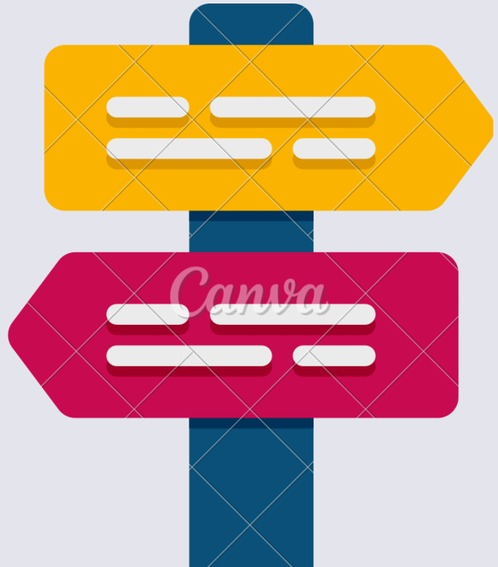
ESTADO ACTUAL DE PHP

Modelo síncrono tradicional: cada solicitud es un nuevo proceso
Desventajas:

- Alto consumo de recursos
- Bloqueo de E/S
- No apto para conexiones persistentes (como chats)
- Escalabilidad vertical (más costosa)

ALTERNATIVAS PARA MEJORAR PHP

- Swoole: agrega event loop y corrutinas; maneja miles de conexiones concurrentes
- ReactPHP: PHP puro, programación asíncrona con promesas
- Arquitecturas híbridas: uso de colas (RabbitMQ, Redis) y workers



FRAMEWORKS Y LIBRERÍAS
EVENT-DRIVEN

Swoole:

- -MOTOR EVENT-DRIVEN Y CORRUTINAS
- SERVIDOR HTTP/WEBSOCKET INTEGRADO
- PROGRAMACIÓN ASÍNCRONA PARA E/S
- ALTO RENDIMIENTO

ReactPHP

- EVENT LOOP BASADO EN LIBEVENT
- COMPONENTES PARA E/S NO BLOQUEANTE
- INTEROPERABILIDAD CON LIBRERÍAS PHP EXISTENTES

COMPARACIÓN CON MODELOS BASADOS EN HILOS

FUNDAMENTOS CONCEPTUALES

El modelo event-driven opera mediante bucles de eventos mientras que el modelo thread-based utiliza multiples hilos o procesos para manejar las solicitudes concurrentes

ESCALABILIDAD

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

ANALISIS DE RENDIMIENTO

Los sistemas event-driven muestran ventajas significativas gracias a que no son bloqueantes y los sistemas thread-based presentan mayor latencia.

COMPLEJIDAD DE DESARROLLO

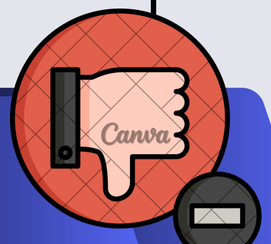
Para event-driver los flujos son asíncronos, el debugging requiere herramientas especiales, mientras que thread-based sus flujos son lineales, con mejor compatibilidad y más facil de comprender para desarrolladores junior

DESVENTAJAS EVENT Y THREAD

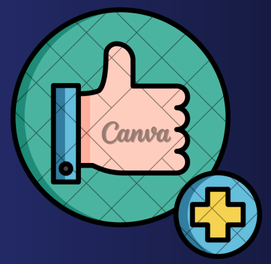
Complejidad en el código en event y problemas de altas cargas con thread

Flujos asíncronos con event y limitación en concurrencia con thread

Problemas con estados compartidos con thread y flujos asíncronos que complican el rastreo de errores con Event



VENTAJAS EVENT Y THREAD

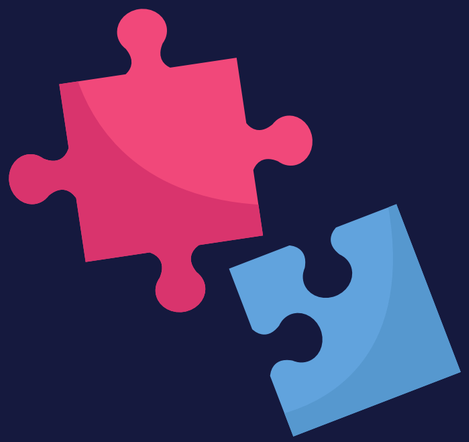


Alto rendimiento en desarrollo con event-driver y simplicidad de desarrollo con Thread

Mejor manejo de la CPU con thread y mejor eficiencia en recursos con Event

Escalabilidad horizontal con Event y Debugging más fácil con thread

DESAFÍOS AL IMPLEMENTAR EVENT-DRIVEN EN PHP



⚠Compatibilidad con código existente

- PHP está diseñado tradicionalmente para ejecución síncrona.
- Requiere refactorizar código legacy para adaptarlo al modelo basado en eventos.

🎓 Curva de aprendizaje

- Conceptos como event loops, callbacks y promesas no son comunes en desarrollo PHP clásico.
- Cambia el enfoque de ejecución lineal a flujo asincrónico y no predecible.

🧱 Limitaciones del entorno de ejecución

- Muchos servidores compartidos no permiten procesos persistentes.
- Requiere infraestructura moderna: contenedores, servidores dedicados o cloud.

🐞 Depuración compleja

- Difícil seguir la lógica entre múltiples eventos y callbacks.
- Errores intermitentes y asincrónicos pueden ser difíciles de detectar y reproducir.

MEJORES PRÁCTICAS Y CONSIDERACIONES DE SEGURIDAD:

🛑 Control de errores asíncronos
- Manejar errores dentro de callbacks/promesas.
- Usar logs y herramientas de monitoreo para detectar fallos ocultos.

🧩Modularización del sistema
- Diseñar eventos y listeners como componentes independientes.
- Facilita mantenimiento y escalabilidad.

🔄 Gestión de recursos
- Controlar uso de memoria, conexiones activas y tiempo de ejecución.
- Evita sobrecarga del servidor.
- bilidad.

🔍 Validación en listeners
- Cada listener debe validar sus propios datos, incluso si ya fueron validados antes.
- Previene errores y vulnerabilidades.

✅ Uso de librerías probadas
- Preferir herramientas como ReactPHP, Amp, Symfony EventDispatcher.
- Evita reinventar la rueda y mejora la confiabilidad del sistema.