UNIVERSIDAD
DON BOSCO

Universidad Don Bosco

Desarrollo de Aplicaciones Web con Software Interpretado en el Servidor

Grupo: G02T

Integrantes:

Daniel Enrique Flores Lino - FL220294

Daniel Alexander Girón Cornejo - GC221469

Eduardo Josué Ortiz Orellana - OO172577

Josué Adrián García Juárez - GJ242648

Docente: Ing. Kevin Jiménez

Actividad: Investigación aplicada 1

Repositorio:

https://github.com/JjayY4/investigacion-aplicada1-dss.git

Introducción

Esta investigación explora el funcionamiento de la programación server-side, sus componentes principales, el rol que desempeña dentro del modelo cliente-servidor y las tecnologías que permiten implementar aplicaciones robustas. Además, se presenta un cuadro comparativo que analiza las características de los lenguajes más utilizados en este ámbito, entre ellos PHP, Node.js, Python, Java, Ruby, ASP.NET y .NET Core. Con ello, se busca ofrecer una visión integral que facilite la selección adecuada de tecnologías según las necesidades específicas de un proyecto, ya que de esto depende el diseño de aplicaciones eficientes, escalables y seguras.

Programación del lado del servidor

Un lenguaje del lado del servidor (server-side) es un lenguaje que se ejecuta en un servidor web inmediatamente antes de que el sitio web se envíe a través de Internet al usuario.

Los navegadores web se comunican con servidores a través de HTTP (HyperText Transfer Protocol). Cuando un usuario hace click en un link, sube un formulario o realiza una búsqueda, el navegador manda una solicitud HTTP al servidor web de destino. Los servidores esperan solicitudes de clientes, y cuando las reciben, las procesan y utilizan la información contenida en las solicitudes para mandar la respuesta HTTP apropiada de vuelta al navegador.
Para poder tener páginas web dinámicas, donde parte del contenido que se manda en la respuesta varía en base a la necesidad, la mayor parte del código debe ejecutarse en el servidor. Escribir este código se conoce entonces como **programación server-side**.

Server-side se refiere, por tanto, a todo lo que, en una aplicación web, ocurre en el servidor. Aunque en el pasado la mayoría de la lógica empresarial se ejecutaba en el servidor, actualmente se limita su función, ya que depender del servidor para cada proceso que se desee realizar implica mandar una solicitud desde el cliente al

servidor cada vez; esto puede sobrecargar al servidor si hay bastantes usuarios concurrentes, y causar latencia alta.

Por otro lado, **client-side** se refiere a todo lo que se muestra o sucede en el cliente (esto es, el dispositivo del usuario final), en lugar del servidor. Esto incluye la interfaz de usuario y acciones que suceden en el navegador. HTML, CSS y JavaScript son lenguajes que se interpretan en el cliente. Utilizar estos lenguajes permite estilizar las páginas y responder a eventos sin necesidad de realizar una solicitud al servidor.

¿Dónde se ejecuta el código server-side?

La programación del lado del servidor "implica el diseño e implementación de programas para ser ejecutados en un servidor", como procesos, ya sea en una *máquina física, máquina virtual o infraestructura en la nube*.
Esto significa que el servidor puede estar ubicado en:

- Centros de datos universitarios o empresariales
- Servidores virtuales (VPS)
- Plataformas en la nube (Azure, AWS, Google Cloud)
- Contenedores (Docker, Kubernetes)

Además, estos servidores pueden ejecutar varias aplicaciones simultáneamente, cada una recibiendo y atendiendo solicitudes de distintos clientes.

¿Cómo se ejecuta el código server-side?

El funcionamiento web se basa en el **modelo cliente-servidor**, donde:

- El cliente envía una **solicitud** (*HTTP request*).
- El servidor **escucha**, procesa esa solicitud y envía la **respuesta** (*HTTP response*).

Detalle del ciclo dentro del núcleo del procesamiento server-side.
Una aplicación server-side se divide en tres capas principales :

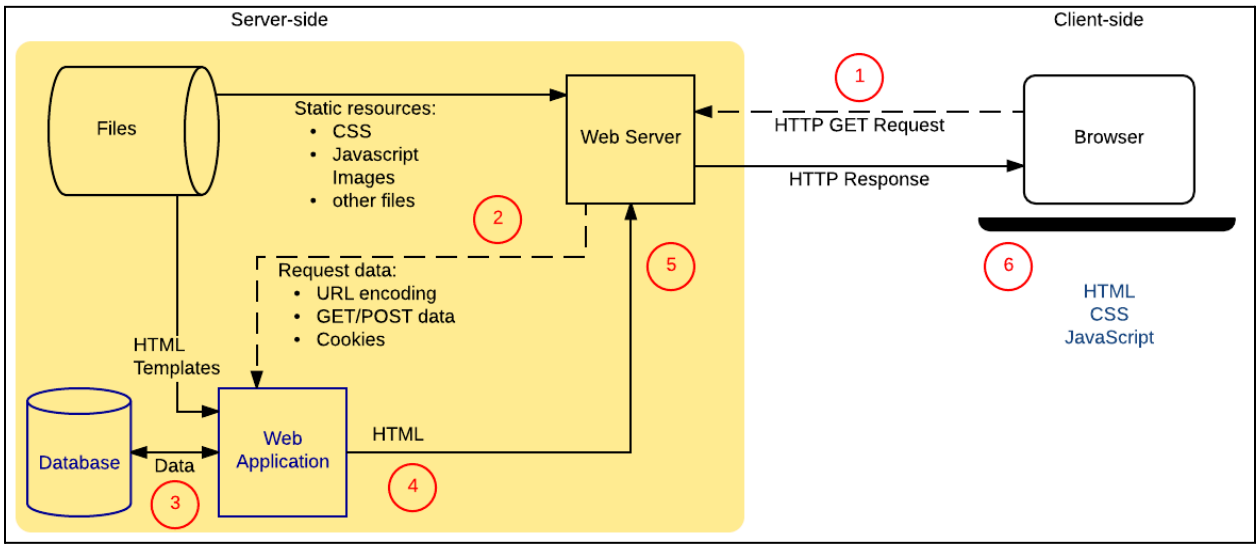
1) *Capa de presentación*
   Genera contenido interpretable por el usuario (HTML, JSON, etc.).

2) *Capa de lógica de negocio*
   Procesa comandos, aplica reglas, toma decisiones y ejecuta cálculos.

3) *Capa de datos (resource layer)*
   Interactúa con bases de datos, sistemas de archivos u otros servicios persistentes.

Al ejecutarse una solicitud HTTP:

1) El servidor recibe parámetros (*query strings, formularios, headers*).
2) Pasa la información a la *lógica de negocio*.
3) Realiza consultas a *bases de datos* o *recursos*.
4) Construye dinámicamente la respuesta.
5) Devuelve el resultado al navegador.



Rol del servidor en aplicaciones web

Básicamente *procesar solicitudes*, *ejecutar la lógica interna de la aplicación* y *devolver una respuesta al cliente*. Funciona como un intermediario entre el Cliente y la Aplicación web.

Función	Descripción
Procesar solicitudes (HTTP Request)	El servidor escucha continuamente peticiones enviadas por los clientes y responde a ellas. Recibe parámetros enviados como los formularios Gestiona conexiones mediante sockets
Ejecutar lógica de negocio	Ejecutar reglas internas de la aplicación: Procesar comandos, tomar decisiones y ejecutar cálculos. Ej: Autenticar usuario, procesar pago, validar datos recibidos
Acceder a bases de datos	Lee y escribe datos de forma persistente. El servidor se comunica con bases de datos, archivos, APIs externas, etc. para organizar el almacenamiento y recuperación de datos mediante gestores de BD o sistema de archivos.
Generar contenido dinámico	Generar contenido dinámico en el servidor como HTML, CSS, JS antes de enviarlo al cliente, en otras palabras: Responder con HTML generado por plantillas Renderizar vistas personalizadas
Enviar respuestas (HTTP Response)	Los servidores envían respuestas de vuelta al cliente siguiendo el protocolo HTTP.
Manejar seguridad	Manejo de sesiones Validación de credenciales Control de acceso

Cuadro comparativo con las características de los principales lenguajes: PHP,NODE.JS,PYTHON,JAVA,RUBY,ASP.NET Y .NET CORE.

Caracteristica	PHP	Node.js	Python	Java	Ruby	ASP.NET	.NET Core
Aislamiento y Concurrencia	Cada request levanta y destruye su propio estado. Excelente aislamiento natural.	Asíncrono (Event Loop). Si un hilo falla, el proceso cae. Poco aislamiento interno.	Concurrencia real limitada. Requiere múltiples procesos del sistema operativo.	Hilos de SO o *Virtual Threads* (Loom). Aislamiento basado en memoria del heap.	Similar a Python. Concurrencia real exige levantar múltiples procesos pesados en memoria.	Gestionado por IIS (*Worker Processes*). Aislamiento basado en *App Domains*.	Multi-hilo altamente optimizado e independiente de la plataforma (Kestrel).
Perfil de Latencia	Limitado por el tiempo de inicialización constante, aunque OPcache lo mitiga.	Muy rápido para red. **Alta en CPU:** Cualquier cálculo matemático bloquea el ciclo.	La sobrecarga del intérprete y el recolector de basura generan latencia impredecible.	El JIT y recolectores modernos (ZGC) reducen las pausas a sub-milisegundos.	El consumo de memoria y el recolector de basura provocan las pausas más largas del grupo.	Penalizaciones heredadas del framework antiguo y acoplamiento estricto a Windows.	Optimizaciones agresivas en asignación de memoria. Tiempos de respuesta extremadamente planos.
Determinismo (Tipado y Ejecución)	Tipado dinámico. Mucha flexibilidad en tiempo de ejecución, reduciendo la predictibilidad estricta.	Dinámico por naturaleza (requiere TypeScript para forzar contratos rígidos).	*Duck typing*. Difícil garantizar la validación de interfaces estrictas sin ejecutar el código.	Tipado estático fuerte, contratos de interfaces rígidos y compilación predictiva.	Basado en metaprogramación. Prioriza la agilidad del desarrollador sobre el determinismo puro.	Tipado estático (C#) con reglas de compilación empresariales estrictas.	Tipado fuerte, soporte de inmutabilidad nativa y compilación AOT (Ahead-of-Time).
Arquitecturas Modulares	Escala horizontalmente fácil, pero no soporta procesos persistentes o *streaming* masivo.	Perfecto para enrutamiento ligero, pero ineficiente como núcleo de procesamiento pesado.	Requiere delegar el trabajo a infraestructuras externas (ej. RabbitMQ + Celery workers).	El estándar de la industria para orquestar millones de transacciones simultáneas de forma predecible.	Escalar requiere multiplicar exponencialmente la infraestructura subyacente de servidores.	Diseñado para monolitos cerrados. Difícil de dividir y casi imposible de containerizar bien.	Diseñado específicamente para microservicios *cloud-native* ligeros y altamente distribuidos.

Conclusión

La programación del lado del servidor continúa siendo un componente esencial en el desarrollo web contemporáneo, encargándose del procesamiento de solicitudes, la ejecución de la lógica de negocio y la interacción segura con bases de datos y otros recursos. A pesar del crecimiento de las tecnologías del lado del cliente, el servidor sigue siendo indispensable para garantizar integridad, seguridad, escalabilidad y personalización en cada aplicación.

El análisis comparativo de los lenguajes estudiados demuestra que cada tecnología posee fortalezas y escenarios ideales de uso. Mientras PHP ofrece simplicidad y amplia adopción, Node.js destaca por su rendimiento no bloqueante; Python por su legibilidad y versatilidad; Java por su robustez empresarial; Ruby por su enfoque en la productividad del desarrollador; y ASP.NET/.NET Core por su alto rendimiento y soporte multiplataforma. La elección del lenguaje y la plataforma adecuada dependerá de los objetivos del proyecto, los requisitos de escalabilidad, el ecosistema disponible y la experiencia del equipo.

En síntesis, comprender la arquitectura server-side y las características de estas tecnologías permite tomar decisiones informadas y desarrollar aplicaciones web más eficientes, seguras y alineadas con las demandas actuales del entorno digital.

Bibliografía

1. MDN, “Introduction to the server side - Learn web development | MDN,” MDN Web Docs, Dec. 19, 2024. https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/First_steps/Introduction
2. CloudFlare, “What Do Client-Side and Server-Side Mean? | Client Side vs. Server Side | Cloudflare,” Cloudflare, 2024. Available: https://www.cloudflare.com/learning/serverless/glossary/client-side-vs-server-side/
3. Tripleten, “Lenguajes de programación web: Guía completa 2025,” TripleTen, Nov. 21, 2025. https://tripleten.mx/blog/lenguajes-de-programacion-web-guia-completa/
4. Aula Creactiva, “Los 4 lenguajes de Programación Web más usados | Aula Creactiva,” Aula Creactiva, Nov. 25, 2024. https://www.aulacreactiva.com/lenguajes-programacion-web/ (accessed Feb. 26, 2026).
5. Hott, R. (2024). *Server-side processing: Introduction + cgi-bin*. CS4640: Programming Languages for Web Applications. University of Virginia. https://cs4640.cs.virginia.edu/spring2024/lectures/11-Server-Side.pdf
6. Ciubotaru, B., & Muntean, G.-M. (2013). Server-side network programming. En *Advanced Network Programming – Principles and Techniques* (pp. 157–191). Springer.https://doi.org/10.1007/978-1-4471-5292-7_8