TALLER ARQUITECTURA DE APLICACIONES WEB

Nombres:
Yenderson Josué Rangel Martínez
Wilber Santiago Barajas Cordero

La arquitectura de aplicaciones web es fundamental para diseñar sistemas eficientes y escalables. Permite que los distintos componentes de una aplicación interactúen de manera coherente.

1. ¿Qué es la Arquitectura de las Aplicaciones Web?

Es el diseño de cómo se organizan y se comunican los componentes de una app web (cliente, servidor, base de datos, APIs, CDN, etc.) para entregar funcionalidades al usuario de forma segura, rápida y escalable. En esencia, define qué piezas existen y cómo interactúan (capas, servicios, protocolos, flujos). Una app típica combina front-end (HTML/CSS/JS en el navegador), back-end (lógica del servidor), y persistencia (BD).

2. ¿Por qué es importante la Arquitectura de las Aplicaciones Web?

- Escalabilidad: crecer sin reescribirlo todo.
- Mantenibilidad: cambios localizados, menor deuda técnica.
- Rendimiento: tiempos de respuesta bajos (CDN, caché, colas).
- Seguridad: separación de responsabilidades y controles por capa.
- Disponibilidad: tolerancia a fallos y despliegues sin caída.

3. ¿Cuáles son los Componentes de la Aplicación Web?

- Cliente (Front-end): interfaz en el navegador (HTML, CSS, JS).
- Servidor (Back-end): procesa peticiones HTTP/HTTPS, orquesta lógica.
- Base de datos: relacional o NoSQL para persistencia.
- API layer / microservicios: acceso a lógica y datos vía HTTP/JSON.
- Web server / App server: sirve estáticos; ejecuta app (Node, .NET, etc.).
- CDN: distribuye contenido estático cerca del usuario.
- Balanceador: reparte tráfico entre instancias.
- Seguridad y monitoreo: WAF, logs, métricas, alertas. ()

4. Mencione las Mejores Prácticas para el desarrollo web
    - Diseño por capas & separación de responsabilidades.
    - API-first y contratos claros (versionado, errores, límites).

- Seguridad por defecto: HTTPS en todo, gestión de secretos, auth robusta. ()
- Cachés y CDN; compresión, lazy loading, paginado.
- Observabilidad: logs estructurados, trazas, métricas, health checks.
- Infra como código y CI/CD con revisiones y despliegues automáticos.
- Testing (unitario/integración/e2e) y linters.
- Escalado horizontal y límites de recursos (timeouts, circuit breakers).

5. Escriba y esplique brevemente los Tipos de Arquitectura de Aplicaciones Web, sus ventajas y desventajas

MONOLÍTICA

Ventajas: Simpler para empezar; despliegue único.

Desventajas: Escala como “bloque”; cambios pueden afectar todo; ciclos largos.

N-CAPAS / N-TIERS (PRESENTACIÓN–LÓGICA–DATOS)

Ventajas: Orden, mantenibilidad, reemplazo por capa.

Desventajas: Puede volverse rígida; orquestación compleja a gran escala. ()

MICROSERVICIOS

Ventajas: Escalan y evolucionan de forma independiente; equipos autónomos.

Desventajas: Complejidad operativa (red, observabilidad, consistencia, DevOps).

SERVERLESS (FAAS + BAAS)

Ventajas: Escalado automático, pago por uso, menos gestión de servidores.

Desventajas: Frío de arranque, límites por proveedor, depuración local distinta.

JAMSTACK (PRE-RENDER + APIS + JS)

Ventajas: Rendimiento y seguridad altos (estáticos en CDN), gran escalabilidad.

Desventajas: Dinámica compleja requiere funciones/API externas. ()

WEB-QUEUE-WORKER (PATRÓN)

Ventajas: Aísla tareas pesadas en workers; mejora respuesta del front.

Desventajas: Añade colas y monitoreo extra. ()

6. Escriba Ejemplos de Tecnologías aplicada a:

- **FRONTEND:** React, Angular, Vue, Next.js, Svelte.
- **BACKEND:** Node.js (Express/Nest), ASP.NET Core, Spring Boot, Django/FastAPI, Laravel.

- **BD RELACIONALES:** PostgreSQL, MySQL/MariaDB, SQL Server.
- **BD NOSQL:** MongoDB, Redis, DynamoDB, Cassandra.
- **APIS & MENSAJERÍA:** REST/JSON, GraphQL, gRPC; RabbitMQ, Kafka.
- **INFRA/DEVOPS:** Docker, Kubernetes, Terraform, GitHub Actions, Jenkins.
- **CDN/EDGE:** Cloudflare, Fastly, Akamai.
- **AUTH:** OAuth 2.0 / OIDC (Keycloak, Auth0).
- **OBSERVABILIDAD:** Prometheus/Grafana, ELK/Opensearch, OpenTelemetry.

7. Que es un protocolo de comunicación

Un conjunto de reglas que definen cómo dos partes (cliente/servidor) formatean, envían, reciben y validan mensajes a través de una red. En la Web, el principal es HTTP y su variante segura HTTPS. ()

8. Explique los protocolos http, https

- HTTP (HyperText Transfer Protocol): protocolo de la Web para solicitar y transferir recursos (HTML, imágenes, JSON, etc.) siguiendo un modelo cliente–servidor y métodos como GET/POST/PUT/DELETE. ()
- HTTPS: HTTP cifrado con TLS, que protege confidencialidad e integridad y autentica el servidor (y opcionalmente el cliente). ()

9. Que es hosting, investigue los tipos de hosting y haga una tabla comparativa de mínimo 4 proveedores de este servicio, elija uno apropiado para su proyecto

Tipos de hosting más comunes:

Tipo	Descripción
Compartido	Económico; recursos compartidos; ideal para sitios pequeños.
VPS/Cloud	Servidor virtual; mayor control; escalable y flexible.
PaaS/Managed	Despliegues fáciles; autoescalado; integración con Git.
Dedicado	Servidor físico exclusivo; máximo control; alto costo.

Comparativa de proveedores:

Proveedor	Tipo principal	Puntos fuertes	Cuándo usar
Vercel	PaaS (Front/SSR)	CDN global, deploy desde Git, funciones serverless	Front React/Next.js, landing de alto tráfico
Netlify	PaaS (JAMstack)	Sitios estáticos, Forms, Identity	Webs JAMstack con forms/auth simple
Render	PaaS (Full-stack)	Auto-deploy, Postgres gestionado	Backends + BD + jobs programados
DigitalOcean	VPS/Cloud	Control total, flexible, buen costo/beneficio	Arquitecturas personalizadas, contenedores

10. Que es un servidor de Dominio, escojan un dominio para su proyecto e investiga si está disponible, agregue capturas para comprobar su investigación

Un “servidor de dominio” se refiere a un DNS (Domain Name System) que traduce nombres como tu-dominio.com a direcciones IP. Los proveedores de hosting o registradores suelen ofrecer DNS gestionado.

Dominio sugerido para el proyecto: jocatoshoes.store