FUNDAMENTOS DE FULL STACK DEVELOPMENT Y ARQUITECTURAS BASE_ PARTE 1

Concepto de Full Stack Development

El **Full Stack Development** se refiere al desarrollo integral de aplicaciones web, abarcando tanto el **lado del cliente (frontend)** como el **lado del servidor (backend)**, incluyendo:

- La gestión de datos
- La lógica del negocio
- La comunicación entre componentes.

Un desarrollador Full Stack comprende el sistema como un todo y puede intervenir en múltiples capas sin perder la coherencia arquitectónica.

Diferencias y responsabilidades entre Frontend y Backend

Frontend

- Interfaz gráfica y experiencia de usuario (UI/UX).
- Interacción con el usuario (formularios, botones, vistas).
- Consumo de APIs.
- Validaciones básicas y control del flujo visual.

Backend

- Lógica de negocio.
- Gestión de usuarios, autenticación y autorización.
- Procesamiento de datos.
- Comunicación con bases de datos y servicios externos.
- Exposición de APIs seguras.

Ambos trabajan de forma **independiente pero coordinada**, bajo contratos claros (endpoints, formatos JSON, reglas de negocio).

Vamos a responder desde lo general:

1. ¿Qué entendemos por contrato?
2. ¿A qué asociamos las reglas de negocio y cómo las definimos?

Flujo de datos en aplicaciones web modernas

En una aplicación moderna, el flujo de datos sigue este patrón general:

1. El usuario interactúa con la interfaz (frontend).
2. El frontend envía una solicitud HTTP (GET, POST, PUT, DELETE).
3. El backend procesa la solicitud.
4. Se consulta o actualiza la base de datos.
5. El backend responde con datos estructurados (JSON).
6. El frontend actualiza la vista sin recargar la página.

Este flujo favorece **interactividad**, **escalabilidad** y **separación de responsabilidades**.

3. ¿Cómo funciona la separación de responsabilidades?

Arquitectura Cliente–Servidor

La arquitectura **Cliente–Servidor** establece una separación clara entre:

- **Cliente**: solicita recursos y presenta información.
- **Servidor**: procesa solicitudes, aplica reglas y gestiona datos.

Ventajas clave:

- Centralización de la lógica.
- Mayor control de seguridad.
- Posibilidad de múltiples clientes (*web, móvil, escritorio*).

Arquitectura por Capas

La **arquitectura por capas** organiza el sistema en niveles con responsabilidades bien definidas, comúnmente:

- ✓ Capa de presentación
- ✓ Capa de lógica de negocio
- ✓ Capa de acceso a datos
- ✓ Capa de persistencia

Este enfoque mejora:

- ✓ Mantenibilidad
- ✓ Escalabilidad
- ✓ Reutilización de código
- ✓ Facilidad de pruebas

4. ¿Cómo podemos deducir la aplicación de la capa de persistencia?

Mirada rápida a la arquitectura web desacoplada

En una arquitectura **desacoplada**, el frontend y el backend se desarrollan y despliegan de manera independiente.

Características:

- ✓ Comunicación mediante APIs REST o GraphQL.
- ✓ El frontend no conoce la implementación interna del backend.
- ✓ Permite cambiar tecnologías sin afectar todo el sistema.

Este modelo es la base de aplicaciones modernas y microservicios.

5. ¿En qué se diferencia de REST?

Principio SPA (Single Page Application) como base del frontend moderno

Una **SPA** carga una sola página HTML inicial y actualiza dinámicamente el contenido sin recargar el navegador.

Beneficios:

- Experiencia de usuario fluida.
- Menor consumo de red.
- Mayor control del estado de la aplicación.
- Integración natural con APIs.

Las SPAs representan el estándar actual del desarrollo frontend.

6. Para que tipos de aplicaciones puedo hacer uso de SPA

Recordemos...

Relación entre frontend, backend y datos en sistemas integrados

En sistemas integrados:

- El **frontend** gestiona la interacción y presentación.
- El **backend** actúa como intermediario y garante de reglas.
- La **base de datos** almacena información estructurada y persistente.

La clave está en mantener **bajo acoplamiento y alta cohesión** entre componentes.

Visión del sistema como un conjunto coherente de componentes

El desarrollo Full Stack no se trata solo de usar muchas tecnologías, sino de pensar el sistema como una arquitectura coherente, donde:

- Cada componente tiene una función clara.
- Las interfaces entre módulos están bien definidas.
- El sistema es escalable, seguro y mantenible, facilitando su crecimiento y adaptación progresiva a nuevas necesidades.

Esta visión sistémica es fundamental para proyectos profesionales.

Antes de llegar a los ejemplos, partimos de la contextualización de Vanilla para trabajar un FE inicial - puro

7. ¿Qué es Vanilla?

Los ejemplos

De acuerdo con los ejemplos expuestos, explicar cómo funciona el js (los bloques de partes) de cada uno de estos frente al archivo base (html).

1. Hospital Docs

Gestión Documental (SPA)

Explicación del SPA: ...

2. Control de Asistencia

SPA (sin recargas)

Explicación del SPA: ...

3. Control de Asistencia (Local Storage)

SPA (sin recargas)

Explicación del SPA: ...

4. Asistencia Full Stack

SPA con Vite (vanilla)

Explicación del SPA: ...