educación IT
CODE YOUR FUTURE

Backend vs Frontend

Arquitectura Cliente-Servidor

Los sistemas fueron evolucionando desde aplicaciones monolíticas hasta la arquitectura *cliente-servidor*, un **modelo de computación distribuida** en el que existen:

- ***Front-End*** (clientes).
- ***Backend*** (servidores).

En un mundo donde la complejidad de los sistemas crece, **la programación orientada a objetos permite organizar y mantener el código de manera más eficiente,** al adaptar sus principios de diseño a la **distribución de responsabilidades en sistemas distribuidos.**

Front-End
(clientes)

Solicitantes de servicios que se encuentran utilizando la interfaz de usuario y las interacciones directas con el usuario final.

Backend
(servidores)

Proveedores de servicios, lógica de negocio, procesamiento de datos y **gestión de la información** de manera estructurada y cohesiva.

Frontend

Es la parte de la aplicación que **interactúa con el usuario.**

Utiliza principios de POO para estructurar componentes que **gestionan la interfaz de usuario y las interacciones** con el usuario final de manera **modular y reutilizable.**

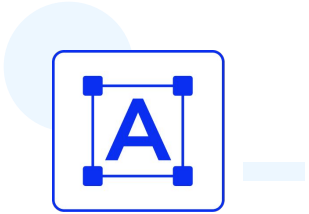
Tecnologías comunes

HTML

CSS

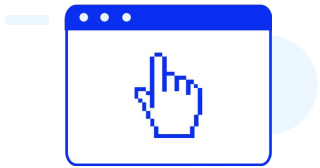
JavaScript

Estructura y gestión de la interacción con el usuario



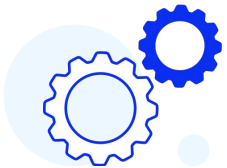
Interfaz de Usuario (UI)

Presentar la interfaz gráfica al usuario final mediante componentes como **objetos reutilizables**. Esto facilita la modularidad y mantenibilidad del código.



Interacción con el usuario

Manejar la interacción directa con el usuario final a través de **métodos y eventos asociados** a los componentes de la interfaz. Esto asegura una respuesta coherente y eficiente.



Lógica de presentación

Definir cómo se presenta la información al usuario utilizando **clases y métodos que estructuran el diseño visual** y la navegación de la interfaz. Esto cumple con los requerimientos de usabilidad del sistema.

Backend

Es la parte de la aplicación que **gestiona la lógica del servidor**, bases de datos, autenticación, y otros recursos.

Aplica conceptos de programación orientada a objetos para **manejar la lógica de negocio, procesar datos y administrar recursos** de manera eficiente y escalable, garantizando la cohesión y el encapsulamiento de la información.

Tecnologías comunes

Node.js

Python

Java

Ruby

Componentes del *backend*

Lógica de la aplicación

Implementar la lógica de negocio de la aplicación mediante **clases de negocio que encapsulan reglas de negocio** complejas y procesos de cálculo. Esto asegura la cohesión y facilita la escalabilidad del sistema.

Gestión de datos

Almacenar y gestionar los datos de la aplicación con **clases de datos y métodos** que aseguran la integridad y consistencia de los datos almacenados en la base de datos. Esto garantiza que se cumplan los requerimientos de persistencia del sistema.

Seguridad y autenticación

Proteger los datos sensibles y gestionar el acceso seguro a los recursos con **clases y métodos que manejan la autenticación** de usuarios y la autorización de acciones, utilizando técnicas de protección para mitigar riesgos de seguridad.

Beneficios y retos de la arquitectura de Cliente-Servidor

Ventajas

- **Escalabilidad:**
  - La separación de responsabilidades permite una mejor gestión de recursos.
  - Los servidores pueden manejar múltiples solicitudes de clientes.
- **Mantenimiento:**
  - Las actualizaciones del servidor no requieren cambios en los clientes.
  - Facilita la administración de aplicaciones complejas.
- **Flexibilidad:**
  - Soporta diferentes tipos de clientes (navegadores, aplicaciones móviles).

Desafíos

- **Complejidad en la integración:**
  - Necesidad de manejar diferentes versiones de cliente y servidor.
  - Requiere protocolos y métodos de autenticación robustos.
- **Dependencia de la red:**
  - Conexión de red necesaria para la comunicación.
  - Latencia y fallos de red pueden afectar el rendimiento.
- **Seguridad:**
  - Necesidad de proteger la comunicación y datos entre cliente y servidor.
  - Autenticación, autorización, y cifrado son esenciales.

Evolución histórica

Aplicaciones de escritorio monolíticas (1980s-1990s)

Aplicaciones que se ejecutan **en un solo equipo** con toda la lógica de negocio y datos integrados en una única unidad.

Ejemplos: Microsoft Word, Lotus 1-2-3.

Aplicaciones Cliente-Servidor distribuidas (1990s)

- Separación de la **lógica de presentación/negocio.**
- Cliente (interfaz de usuario) se **comunica con servidor** (lógica de negocio).
- Mejor **escalabilidad y distribución** que las monolíticas.
- Aparición de **protocolos** como RPC, CORBA.

Primeras aplicaciones web HTTP (Mediados 1990s)

- Aplicaciones **accesibles a través de navegadores** web que emplean HTTP.
- HTTP/0.9: Solicitudes básicas **`GET`**, sin cabeceras.
- HTTP/1.0: Introducción de **cabeceras HTTP,** soporte para **`POST`** y **`HEAD`**.

  Ejemplos: Primeros portales web y formularios en línea.

Aplicaciones web modernas (2010s - Presente)

- Aplicaciones de **una sola página (SPA).**
- ***APIs RESTful:*** Comunicación estandarizada entre frontend y backend.
- Arquitecturas modernas: **Microservicios, JAMstack.**

  Ejemplos: React, Angular, Vue.js.
- Experiencia de usuario **rápida y responsiva.**
- **Separación clara entre *frontend* y *backend.***

Resumen de la evolución de los sistemas

educación IT
CODE YOUR FUTURE

HyperText Transfer Protocol (HTTP)

La comunicación entre *backend* y *frontend*

El Protocolo de Transferencia de Hipertexto, se utiliza para la **transferencia de datos en la web.**

Permite la **comunicación** entre navegadores (clientes) y servidores web.

Gracias a HTTP, es posible **generar aplicaciones** distribuidas y escalables. Además, proporciona un **estándar para la comunicación** entre el *backend* y el *frontend.*

Antes de la aparición de HTTP, existían problemas significativos como **incompatibilidades de comunicación y restricciones de *firewall,*** debido a la falta de un estándar universalmente aceptado.

Solicitud HTTP

Es un **texto** y está compuesta por:

Línea de solicitud

Contiene el método HTTP (GET, POST, y otros), la ruta del recurso y la versión del protocolo.

Cuerpo *(Body)*

Opcionalmente, contiene datos enviados al servidor, como en formularios POST.

```
GET /pagina.html HTTP/1.1
Host: www.ejemplo.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Accept-Language: es-ES,es;q=0.9,en;q=0.8
Content-Type: application/json
Content-Length: 33

{
  "parametro1": "valor1",
  "parametro2": "valor2"
}
```

Encabezados *(Headers)*

Información adicional sobre la solicitud, como tipo de contenido aceptado, *cookies*, y otros datos.

Acciones HTTP

Son **operaciones** definidas por el protocolo HTTP que definen las acciones que el **cliente desea realizar** sobre un recurso del servidor.

	GET	*POST*	*PUT*	*DELETE*
Descripción	Solicita la **representación de un recurso** específico. Los datos se envían en la **URL.**	**Envía datos al servidor para crear recurso.** Los datos se envían en el **cuerpo** de la solicitud.	**Reemplaza todas las representaciones** actuales de un recurso **en el servidor** con los datos proporcionados.	**Elimina** un recurso específico **del servidor.**
Uso	**Obtener datos de un servidor** (por ejemplo, una página web, un archivo).	**Enviar datos** de formularios, **cargar** archivos.	**Actualizar datos** existentes en el servidor.	**Borrar datos** o recursos en el servidor.

Status Code HTTP

Los códigos de estado HTTP son **números que indican el resultado de una solicitud realizada por un cliente a un servidor web.**

Proporcionan información sobre si la solicitud fue exitosa, redirigida, o si ocurrió algún tipo de error, ya sea del cliente o del servidor.

Informacionales (100-199)	Indican que la solicitud ha sido **recibida** y el servidor continúa **procesándola.**
Éxito (200-299)	Indican que la solicitud ha sido recibida, comprendida y **aceptada** correctamente.
Redirecciones (300-399)	Indican que se deben tomar más **acciones para completar** la solicitud.
Errores del Cliente (400-499)	Indican que hubo un **error** por parte del **cliente al enviar** la solicitud.
Errores del Servidor (500-599)	Indican que hubo un **error** por parte del **servidor al procesar** la solicitud.

Ejemplo de interacción

Aquí se puede observar un ejemplo de **cómo funciona** el protocolo HTTP en el contexto de una interacción sencilla, entre el cliente y el servidor:

1. El cliente envía una **solicitud `GET`** al servidor para obtener una página con un formulario.
2. El servidor **devuelve la página** al cliente, que la completa con datos.
3. El cliente envía una **solicitud `POST`** al servidor con los datos completados en el formulario.
4. El servidor **procesa** la solicitud `POST` y envía una **confirmación** o realiza acciones adicionales según los datos recibidos.