UNIDAD 1.

Introducción al desarrollo web.

(El estudiante identificará las herramientas y elementos para la estructura de un documento HTML.)

Tema II. Entornos de desarrollo web

Evidencia de aprendizaje Tema II –RESUMEN

Temas	Saber	Saber hacer	Resultado de Aprendizaje
Entornos de desarrollo web	Identificar las ventajas y desventajas de las herramientas del lado del cliente en el desarrollo Web. Identificar las ventajas y desventajas de las herramientas del lado del servidor en el desarrollo Web.	Instalar y configurar el ambiente de desarrollo Web.	Los estudiantes identifican las herramientas relacionadas al desarrollo web.
Evidencia de aprendizaje			
Examen de Opción Múltiple, Tema: Conceptos y Configuración de un Ambiente de Desarrollo Web. Este examen evalúa los conocimientos sobre la **instalación, configuración y verificación de un entorno de desarrollo web**. El estudiante deberá identificar conceptos básicos, reconocer el software necesario, comprobar su correcta instalación y mostrar los programas instalados en su PC relacionados con el entorno web.			

Información sobre el profesor

Profesor	Correo	Días de Clase
Bernardo Prado Díaz	Tenor_prado@yahoo.com.mx	Lunes y Martes

Entornos de Desarrollo Web con Python y Django

1 Dimensión Conceptual – Ventajas y Desventajas de Herramientas en Desarrollo Web

Objetivo: Comprender las características, ventajas y desventajas de las herramientas utilizadas en el desarrollo web del lado del cliente y del lado del servidor, con enfoque en Python y Django.

A) Herramientas del Lado del Cliente

Definición: Tecnologías que se ejecutan en el navegador del usuario, encargadas de la parte visual y la interacción directa.

Ejemplos: HTML, CSS, JavaScript.

Ventajas:

- Experiencia de usuario interactiva y dinámica.

- Reducción de carga en el servidor.

- Actualizaciones inmediatas sin recargar toda la página (AJAX).

- Compatibilidad con múltiples navegadores y dispositivos.

Desventajas:

- Dependencia del rendimiento del dispositivo del usuario.

- Mayor exposición del código a vulnerabilidades si no se implementa seguridad.

- Compatibilidad limitada si no se respetan estándares.

- Sin acceso directo a bases de datos.

B) Herramientas del Lado del Servidor

Definición: Tecnologías que se ejecutan en el servidor, procesando datos, realizando cálculos y gestionando la lógica del negocio.

Ejemplos: Python, Django, Node.js, PHP.

Ventajas:

- Mayor seguridad en el manejo de datos.

- Procesamiento potente sin depender del dispositivo del usuario.

- Integración directa con bases de datos.

- Posibilidad de manejar sesiones y autenticación.

Desventajas:

- Mayor carga de trabajo en el servidor.

- Tiempo de respuesta más lento si el servidor es limitado.

- Requiere conexión a Internet para la mayoría de funciones.

- Escalabilidad depende de la infraestructura.

2 Ejemplos Prácticos con Python y Django

Caso: Aplicación de registro de usuarios.

• Lado del Cliente: Formulario HTML con validación en JavaScript para verificar que los campos estén completos antes de enviarlos.

• Lado del Servidor: Django recibe la información, valida en el back-end y guarda los datos en la base de datos MySQL.

Ejemplo de vista en Django (views.py):

```
from django.shortcuts import render, redirect
from .models import Usuario

def registrar_usuario(request):
    if request.method == 'POST':
        nombre = request.POST['nombre']
        correo = request.POST['correo']
        Usuario.objects.create(nombre=nombre, correo=correo)
        return redirect('lista_usuarios')
    return render(request, 'registro.html')
```

3 Glosario

- HTML → Lenguaje de marcado para estructurar páginas web.

- CSS → Lenguaje de estilos para dar formato visual a las páginas.

- JavaScript → Lenguaje de programación que añade interactividad en el lado del cliente.

- Python → Lenguaje de programación versátil usado en el back-end.

- Django → Framework de Python para desarrollo web rápido y seguro.

- Servidor → Computadora que procesa y entrega datos a los clientes.

- Base de datos → Sistema para almacenar y organizar datos.

- Front-End → Parte visual e interactiva de una aplicación web.

- Back-End → Lógica, procesamiento y gestión de datos en una aplicación web.

Entornos de Desarrollo Web con Python y Django

1. Dimensión Conceptual: Herramientas en Desarrollo Web

El desarrollo de aplicaciones web es una sinergia de tecnologías. Se divide en dos grandes áreas, cada una con un rol crucial para el funcionamiento de cualquier sitio moderno. Comprender sus responsabilidades es el primer paso para dominar el desarrollo web.

A) Herramientas del Lado del Cliente (Front-end)

Definición: El **front-end** son las tecnologías que se ejecutan directamente en el **navegador del usuario**. Se encargan de la parte visual y la interacción directa, es decir, todo lo que un usuario ve y utiliza.

Ejemplos:

- **HTML (HyperText Markup Language):** Proporciona la estructura y el contenido de la página.
- **CSS (Cascading Style Sheets):** Define el diseño, los colores y la tipografía.
- **JavaScript:** Agrega interactividad y dinamismo.

Ventajas:

- **Experiencia de Usuario Interactiva:** La interfaz es fluida y responde de inmediato a las acciones del usuario, sin necesidad de esperar al servidor.
- **Reducción de Carga del Servidor:** Al procesar parte de la lógica en el dispositivo del cliente, se minimiza la carga de trabajo del servidor.

- **Actualizaciones Inmediatas:** Permite que ciertas partes de una página se actualicen sin recargarla por completo (usando **AJAX**).

Desventajas:

- **Dependencia del Dispositivo:** El rendimiento puede variar según la capacidad del dispositivo del usuario.
- **Mayor Exposición del Código:** El código es visible para el usuario, lo que puede suponer un riesgo si no se aplican medidas de seguridad.

B) Herramientas del Lado del Servidor (Back-end)

Definición: El **back-end** son las tecnologías que se ejecutan en el **servidor**. Se encargan de la lógica interna, el procesamiento de datos y la conexión con bases de datos. Es la "maquinaria" invisible detrás de la aplicación.

Ejemplos:

- **Lenguajes: Python**, Node.js, PHP, Ruby.
- **Frameworks: Django** (para Python), Flask, Express.js, Laravel.
- **Bases de Datos:** MySQL, PostgreSQL, MongoDB.

Ventajas:

- **Mayor Seguridad:** La lógica y los datos sensibles se gestionan en el servidor, lejos del acceso directo del usuario.
- **Procesamiento Potente:** La capacidad de cálculo no depende del dispositivo del usuario.
- **Integración con Bases de Datos:** Permite almacenar, recuperar y gestionar grandes volúmenes de información de manera segura.

Desventajas:

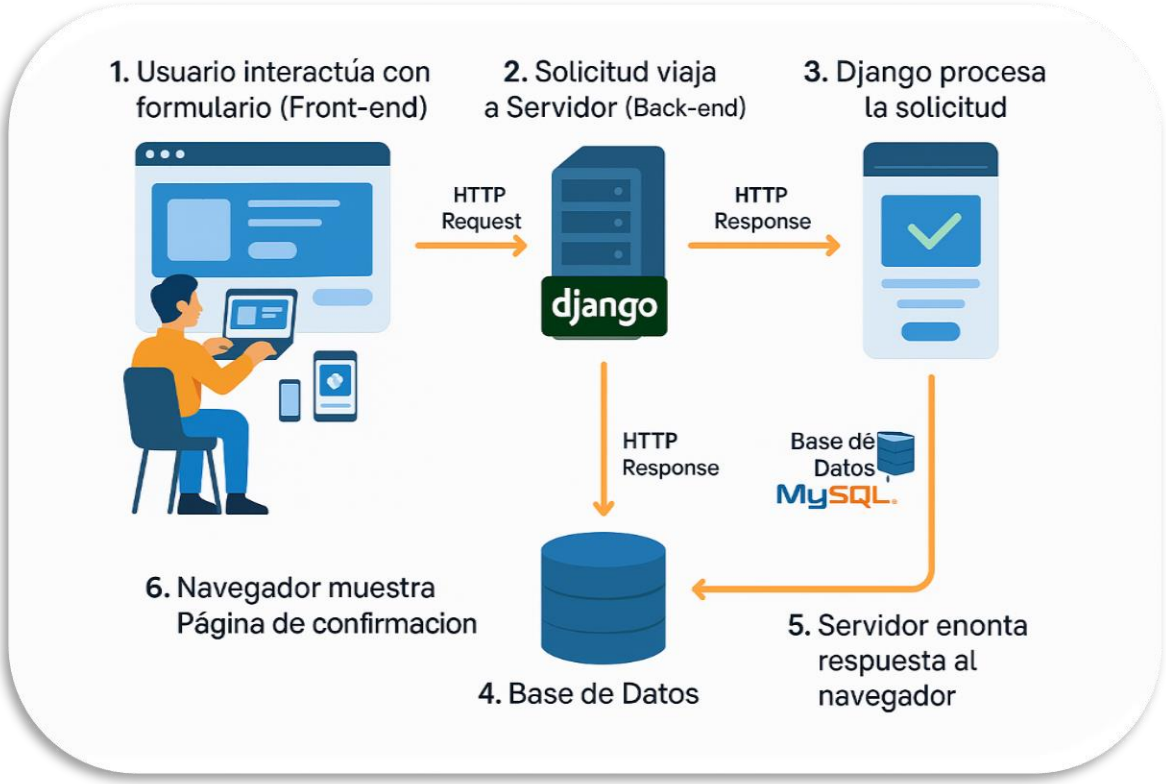
- **Mayor Carga en el Servidor:** Todo el trabajo pesado de lógica de negocio y gestión de datos recae sobre el servidor.
- **Requiere Conexión a Internet:** Las funciones principales no pueden ejecutarse sin una conexión activa.

2. Ejemplo Práctico con Python y Django: Aplicación de Registro

Para entender la interacción entre el front-end y el back-end, consideremos el caso de una aplicación de registro de usuarios.

Un diagrama de flujo que ilustra el ciclo de vida de una solicitud.

1. Un usuario interactúa con un formulario en su **navegador (Front-end)**.
2. La solicitud viaja a un **servidor (Back-end)**.
3. En el servidor, se muestra el logo de **Django**, que procesa la solicitud.
4. Django se conecta a una **base de datos (MySQL)** para guardar la información.
5. El servidor envía una respuesta de vuelta al navegador.
6. El navegador muestra una página de confirmación.



Qué estás viendo

- **Izquierda:** el **navegador** con un **formulario** (Front-End).
- **Centro:** el **servidor** con **Django** (Back-End).
- **Derecha:** la **base de datos MySQL**.
- **Flechas numeradas:** el flujo de ida y vuelta de la solicitud (request) y la respuesta (response).

1) Usuario completa un formulario (Front-End)

- El alumno/usuario escribe datos (ej. *nombre, correo, carrera*).
- El navegador prepara una **solicitud HTTP**.
    - Método típico: **POST** (envía datos).
    - Alternativa: **GET** (consulta sin modificar datos).
- Si tu template usa {% csrf_token %}, se incluye un **token CSRF** para proteger contra ataques Cross-Site Request Forgery.
- **Validación en el cliente (opcional):** HTML5 (required, type="email") y/o JavaScript para feedback inmediato.
  Ojo: esto **no sustituye** la validación en el servidor.

2) La solicitud viaja al servidor (Back-End)

- El navegador envía la petición a https://tu-dominio/....
- **HTTP/HTTPS:** con HTTPS, antes hay un **handshake TLS** (cifrado).
- La solicitud contiene:
    - **Línea de petición:** POST /alumnos/ HTTP/1.1
    - **Cabeceras:** Host, User-Agent, Cookie (sesión), etc.
    - **Cuerpo (body):** datos del formulario (nombre=...&correo=...).

3) Django recibe y procesa

Dentro del servidor suceden varias cosas:

3.1 URL Routing

- urls.py decide qué **vista** (view) maneja /alumnos/.

3.2 Middlewares (antes y después)

- Seguridad, sesión, CSRF, mensajes, etc.

3.3 Vista (View)

- Si es **POST**: la vista (p. ej. registro_alumnos) lee request.POST.
- **Validación del servidor** (inevitable):
    - Con forms.ModelForm o validaciones manuales.
    - Rechaza datos inválidos y **devuelve errores** al template si hace falta.

3.4 Lógica de negocio

- Si todo está OK, la vista **crea o modifica** objetos (ej. Alumno.objects.create(...)).
- Se usa el **ORM de Django** (capa que traduce Python ⇄ SQL).

3.5 Transacción

- Django envuelve operaciones en **transacciones**.
    - Si hay error (ej. duplicado de correo), **rollback**: nada cambia en BD.

4) Django ↔ MySQL (persistencia)

- El ORM ejecuta el **SQL** contra **MySQL**:

    - INSERT INTO alumnos_alumno (...) VALUES (...);
- La BD valida tipos, unicidad (unique=True), claves, etc.
- Responde al servidor con el resultado (éxito o error).

5) El servidor devuelve una respuesta al navegador

- Tipos comunes de respuesta:
    - **Redirect (302)** → patrón **PRG (Post/Redirect/Get)** para evitar repeticiones al recargar.
    - **HTML renderizado (200)** → muestra la página ya con la nueva lista/confirmación.
    - **JSON (200)** → si es una API.
- Incluye **cabeceras**:
    - Content-Type: text/html; charset=utf-8 (o application/json)
    - Set-Cookie para sesión, si aplica.
- **Códigos de estado** típicos:
    - **2xx** éxito, **3xx** redirección, **4xx** error del cliente (p. ej. 400/403/404), **5xx** error del servidor.

6) El navegador muestra la confirmación

- Si hubo **PRG**: el navegador sigue el 302 con un **GET** a la página de éxito/listado.
- Se renderiza un **mensaje de confirmación** (p. ej. “Alumno registrado”).
- Si hubo errores de validación, el usuario ve el **formulario con mensajes** junto a los campos.
- En apps modernas, este paso puede ser un **update parcial** con **Fetch/AJAX** sin recargar toda la página.

Cosas importantes que “no se ven” pero importan

- **Sesión y autenticación:** cookies de sesión; request.user en Django.
- **CSRF:** obligatorio en formularios POST; en AJAX se manda el token en header.
- **Logs/monitorización:** registrar errores y tiempos de respuesta.
- **Seguridad:** sanitización, permisos, límites de tamaño de archivo, CORS en APIs.

- **Caché/CDN:** para estáticos y vistas pesadas (mejora rendimiento).

Análisis Detallado del Diagrama de Flujo: Cómo funciona una Aplicación Web con Django

La imagen anterior muestra un diagrama de flujo que ilustra el proceso completo de interacción entre un usuario, un navegador, un servidor y una base de datos, usando **Django** como el cerebro del servidor.

1. El Comienzo: Interacción del Usuario (Front-end)

- **Lo que muestra el diagrama:** Un usuario sentado frente a una computadora portátil y un teléfono móvil, interactuando con una interfaz. El texto indica: "1. Usuario interactúa con formulario (Front-end)".
- **Explicación sencilla:** Esto representa el primer paso de cualquier aplicación web. El "front-end" es la parte visible de la página, la que tú ves y con la que interactúas. Imagina que es un formulario de registro, donde pones tu nombre y correo electrónico. Todo lo que está en la pantalla —los campos de texto, los botones, los colores— es parte del front-end, construido con tecnologías como **HTML, CSS y JavaScript**.

2. La Solicitud de Viaje: Al Servidor (Back-end)

- **Lo que muestra el diagrama:** Una flecha que sale del navegador con la etiqueta "HTTP Request" y se dirige a un servidor. El texto dice: "2. Solicitud viaja a Servidor (Back-end)". El servidor tiene un logo de "django".
- **Explicación sencilla:** Cuando llenas el formulario y haces clic en "Enviar", tu navegador no lo guarda en tu computadora; lo empaca en un mensaje llamado "HTTP Request" (solicitud HTTP) y lo envía a un servidor a través de Internet. El "back-end" es el lugar donde está el servidor, y es el cerebro que procesará tu información. En este caso, el servidor está utilizando el framework **Django** para recibir y entender esa solicitud.

3. El Cerebro del Proceso: Django Procesa la Solicitud

- **Lo que muestra el diagrama:** El servidor con el logo de Django, y una flecha que se dirige a la base de datos. El texto dice: "3. Django procesa la solicitud".
- **Explicación sencilla:** Una vez que la solicitud llega al servidor, **Django** la recibe y actúa. Es el "motor" de la aplicación. En este punto, Django lee los datos que enviaste (tu nombre y correo) y ejecuta la lógica de negocio. Por ejemplo, podría validar si el correo electrónico tiene el formato correcto, o si ese usuario ya existe. Si todo está bien, pasará al siguiente paso.

4. El Almacén de Datos: La Base de Datos

- **Lo que muestra el diagrama:** Una flecha que conecta el proceso de Django con una caja que representa la base de datos, con el logo de MySQL. El texto dice: "4. Base de Datos".
- **Explicación sencilla:** La base de datos es como un enorme archivador digital, donde la aplicación guarda toda la información de manera organizada y permanente. El código de Django le dice a la base de datos que guarde el nuevo usuario con el nombre y correo que acabas de enviar. En el diagrama, se usa **MySQL**, que es una de las bases de datos más populares para este fin.

5. La Respuesta del Servidor al Navegador

- **Lo que muestra el diagrama:** Una flecha etiquetada "HTTP Response" que sale del servidor y regresa hacia el navegador. El texto dice: "5. Servidor envía respuesta al navegador".
- **Explicación sencilla:** Después de procesar la solicitud y guardar los datos en la base de datos, el trabajo del servidor no ha terminado. Necesita confirmarle al navegador que todo salió bien. Para ello, crea un mensaje de respuesta (un "HTTP Response") y lo envía de regreso. Esta respuesta puede ser una página de "¡Registro Exitoso!" o simplemente una confirmación de que la información se guardó correctamente.

6. El Fin del Viaje: Navegador Muestra la Confirmación

- **Lo que muestra el diagrama:** El navegador, ahora mostrando una nueva pantalla, quizás con un ícono de un visto bueno. El texto dice: "6. Navegador muestra Página de confirmación".
- **Explicación sencilla:** Tu navegador recibe la respuesta del servidor y, finalmente, actualiza la pantalla. Si todo fue exitoso, verás una página de confirmación. Este es el resultado final de todo el proceso: el usuario percibe un cambio y sabe que su acción fue procesada correctamente.

En resumen:

El proceso completo es un ciclo. Un **usuario** envía una solicitud desde el **navegador (front-end)**, el **servidor (back-end con Django)** la recibe, la procesa, interactúa con la **base de datos**, y finalmente, envía una respuesta de vuelta para que el navegador se la muestre al usuario. Este es el modelo de **cliente-servidor** en su máxima expresión.

- **Lado del Cliente (Front-end):** El usuario ve un formulario HTML simple con campos para nombre y correo. Un script de JavaScript valida en tiempo real que los campos estén completos, ofreciendo una respuesta inmediata.
- **Lado del Servidor (Back-end con Django):** Una vez que el usuario envía el formulario, Django recibe la información, la valida de forma segura y utiliza el **ORM (Object-Relational Mapper)** para guardar los datos en la base de datos MySQL.

Ejemplo de Código en Django (views.py)

Este código de Python es la "vista" que procesa la lógica del registro de usuarios en el servidor:

Python

```
from django.shortcuts import render, redirect

from .models import Usuario


def registrar_usuario(request):
    if request.method == 'POST':
        nombre = request.POST['nombre']
        correo = request.POST['correo']

        # El ORM de Django se conecta a la base de datos y crea un nuevo registro
        Usuario.objects.create(nombre=nombre, correo=correo)

        # Redirige al usuario a otra página
        return redirect('lista_usuarios')

    return render(request, 'registro.html')
```

Conclusión

El desarrollo web moderno es una orquesta de tecnologías donde el **front-end** y el **back-end** trabajan en conjunto. El front-end se enfoca en la experiencia del usuario, mientras que el back-end, con herramientas robustas como **Python y Django**, gestiona de forma segura la lógica, la seguridad y los datos. Comprender la sinergia entre estos dos mundos es esencial para construir aplicaciones web eficientes y escalables.