

Ingeniería en Sistemas de Información	
<b>Cátedra:</b> Paradigmas y lenguajes de programación III	<b>Profesor:</b> Mgter. Ing. Agustín Encina
<b>Alumno:</b> Salazar, Facundo	<b>Fecha:</b> 29/10/2025

Duración máxima: 2.30 horas 

### Instrucciones Generales:

- Este examen es interactivo y se compone de varias decisiones que tomarás a lo largo del camino.
- Siga las instrucciones cuidadosamente en cada punto de decisión.
  - La puntuación total se basará en las decisiones tomadas y en la implementación de las tareas relacionadas con cada opción.
  - **No se permiten consultas en línea ni colaboración con otros estudiantes ni con un transformador generativo preentrenado.**



### NARRATIVA DE LA AVENTURA

Has sido contratado por una startup tecnológica que necesita urgentemente un proyecto web funcional. Tu misión es demostrar tus habilidades como desarrollador full-stack navegando por diferentes desafíos. Cada decisión que tomes definirá tu camino y las tecnologías que dominarás.

**¡Tu reputación como desarrollador está en juego!** 

### PARTE 1: DESAFÍOS TEÓRICOS (20 puntos)



### ELECCIÓN DE MISIÓN INICIAL

Antes de comenzar tu proyecto, el equipo técnico necesita evaluar tus conocimientos fundamentales. Elige tu ruta de especialización:

Elige tu Proyecto (*tildar la opción que vas a desarrollar*):

☒ Ruta A: desarrolla el grupo A de preguntas.

☐ Ruta B: desarrolla el grupo B de preguntas.

## **RUTA A: "El Arquitecto Web" (Fundamentos y Estructura)**

### **Desafío 1 - Arquitectura de la Web (5 puntos)**

*El CTO te pregunta durante la reunión inicial...*

Dibuja y explica detalladamente la arquitectura Cliente-Servidor. Incluye:

- Componentes principales
- Flujo de comunicación
- Protocolos involucrados
- Ejemplo práctico con un caso real

### **Desafío 2 - Maestría en CSS (5 puntos)**

*El diseñador UX necesita claridad en la nomenclatura...*

Explica la diferencia entre selectores de clase y selectores de ID en CSS:

- ¿Cuándo usar cada uno?
- Nivel de especificidad
- Proporciona 2 ejemplos prácticos de cada uno aplicados a una interfaz real

### **Desafío 3 - Fundamentos de JavaScript (5 puntos)**

*El líder técnico evalúa tu comprensión de JS...*

Explica el concepto de variables en JavaScript:

- Propósito y utilidad
- Diferencias entre *var*, *let* y *const*
- Proporciona 3 ejemplos mostrando scope y hoisting

#### **Desafío 4 - Introducción a PHP (5 puntos)**

*El backend developer senior te hace una pregunta clave...*

¿Qué es PHP y cuál es su rol en el desarrollo web moderno?

- Características principales
- Diferencias con lenguajes frontend
- Ejemplo de código PHP integrado en HTML (procesamiento de formulario)

#### **RUTA B: "El Innovador Técnico" (Evolución y Arquitectura)**

#### **Desafío 1 - Evolución del HTML (5 puntos)**

*El product manager pregunta sobre tecnologías modernas...*

Explica las diferencias clave entre HTML y HTML5:

- Nuevas etiquetas semánticas
- Mejoras en accesibilidad y SEO
- ¿Cómo HTML5 revolucionó el desarrollo web?

## **Desafío 2 - Arquitectura CSS Avanzada (5 puntos)**

*El tech lead quiere saber si conoces buenas prácticas...*

Explica la diferencia entre arquitectura y metodología en CSS:

- Menciona al menos UNA arquitectura (ej: ITCSS, SMACSS, Atomic)
- Menciona al menos UNA metodología (ej: BEM, OOCSS, SUIT)
- ¿Por qué son importantes en proyectos grandes?

## **Desafío 3 - JavaScript vs PHP (5 puntos)**

*El arquitecto de software evalúa tu visión técnica...*

Compara y contrasta JavaScript y PHP:

- Diferencias fundamentales (ejecución, tipado, uso)
- 3 escenarios donde JavaScript es más apropiado
- 3 escenarios donde PHP es más apropiado
- Ejemplo de código de cada uno

## **Desafío 4 - Conexión a Bases de Datos (5 puntos)**

*El DBA necesita confirmar tus conocimientos de persistencia...*

Describe los conceptos fundamentales para conectar PHP con una Base de Datos:








- Métodos de conexión (MySQLi vs PDO)
- Pasos para establecer conexión
- Manejo de errores
- Ejemplo de código con consulta preparada

## PARTE 2: PROYECTO PRÁCTICO (80 puntos - *distribuidos en 4 niveles*)

### Nivel 1 : ELECCIÓN DE PROYECTO BASE (20 puntos)

⚠ **REGLA CRÍTICA DE NOMENCLATURA:** Todos los archivos, carpetas, clases, funciones, tablas, etc., deben usar como prefijo tus iniciales.

*Ejemplo: Si eres María González López (MGL):*

-  Carpeta: `mgl_assets/`
-  CSS: `mgl_estilos.css`
-  Base de datos: `mgl_parcial_plp3`
-  Tabla: `mgl_usuarios`
-  Función: `function mgl_validar()`
-  Imagen: `mgl_logo.png`
-  Clase CSS: `.mgl-header`

 **MISIÓN PRINCIPAL - Elige tu Proyecto (*tildar la opción que vas a desarrollar*):**

- ☐ Opción A: "MusicStream" - Plataforma de Música Online
- ☐ Opción B: "FoodExpress" - Sistema de Pedidos Online.
- ☒ Opción C: "QuizMaster" - Plataforma de Trivia.

## **PROYECTO A: "MusicStream" - Plataforma de Música Online**

*Una discográfica indie quiere su propia plataforma de streaming*

### **Requisitos Funcionales:**

- Catálogo de álbumes/canciones con reproductor básico (mínimo 8 items)
- Sistema de búsqueda por artista, género o álbum
- Formulario de suscripción con validación
- Listas de reproducción o favoritos (almacenadas en BD)
- Panel para agregar/editar canciones (CRUD)

### **Requisitos No Funcionales:**

- Mínimo 3 secciones distintas (header, galería, formulario)
- Diseño responsive (3 breakpoints)
- Navegación intuitiva y accesible
- Código comentado y estructura modular

## **PROYECTO B: "FoodExpress" - Sistema de Pedidos Online**

*Un restaurante local necesita digitalizar sus pedidos*

### **Requisitos Funcionales:**

- Menú de productos con categorías (mínimo 10 productos)
- Carrito de compras dinámico con subtotales

- Formulario de pedido que guarda en BD
- Sistema de filtrado por categoría
- Panel administrativo para gestionar productos

#### **Requisitos No Funcionales:**

- Mínimo 3 secciones (menú, carrito, checkout)
- Responsive design con mobile-first
- Feedback visual en todas las interacciones
- Tiempo de carga optimizado



#### **PROYECTO C: "QuizMaster" - Plataforma de Trivia**

*Una institución educativa quiere gamificar el aprendizaje*

#### **Requisitos Funcionales:**

- Sistema de preguntas con múltiple opción (mínimo 15 preguntas)
- Validación de respuestas en tiempo real
- Sistema de puntuación y temporizador
- Tabla de mejores puntajes (stored en BD)
- Categorías temáticas con dificultad variable

#### **Requisitos No Funcionales:**

- Mínimo 3 secciones (inicio, juego, resultados)
- Animaciones fluidas y feedback inmediato

- Diseño responsive
- Interfaz intuitiva sin instrucciones complejas

## ⚡ NIVEL 2: Interactividad con JavaScript (20 puntos)



**Documenta:** Comenta la funcionalidad al inicio del archivo JS (3-5 líneas).

### Para PROYECTO A (MusicStream):

Implementar: Reproductor Interactivo con Playlist

- Play/Pause/Skip con controles visuales
- Barra de progreso funcional
- Lista de reproducción dinámica
  - Almacenar última canción

reproducida **Para PROYECTO B**

**(FoodExpress):** Implementar: Carrito de Compras Dinámico

- Agregar/eliminar productos sin recargar
- Cálculo automático de subtotales
- Validación de cantidades
  - Mostrar contador de items en el

carrito **Para PROYECTO C**

**(QuizMaster):** Implementar: Lógica de Juego Completa

- Algoritmo de validación de respuestas
- Sistema de puntuación progresiva



- Temporizador con penalización
- Feedback visual inmediato (correcto/incorrecto)



### **NIVEL 3: Backend con PHP (20 puntos)**

⚠ **OBLIGATORIO:** Conexión e interacción con Base de Datos MySQL



**Documenta:** Comenta la funcionalidad PHP implementada.

Implementación Requerida:

#### **Para MusicStream:**

- CRUD de canciones/álbumes
- Sistema de favoritos persistente
- Búsqueda con queries SQL

#### **Para FoodExpress:**

- CRUD de productos
- Registro de pedidos en BD
- Cálculo de totales en servidor

#### **Para QuizMaster:**

- Banco de preguntas desde BD
- Sistema de ranking persistente
- Registro de partidas jugadas



#### **Base de Datos:**

**Crear con mínimo 2 tablas relacionadas:**

- Claves primarias y foráneas
- Datos de prueba (mínimo 10 registros)

### Exportar:

- [iniciales]\_estructura.sql
- [iniciales]\_datos.sql



## NIVEL 4: Diseño y Experiencia Visual (20 puntos)



**Documenta:** Explica tus decisiones de diseño (paleta, tipografía, layout).

### Requisitos Funcionales:

- Paleta de colores coherente (4-5 colores)
- Tipografía consistente (jerarquía clara)
- Responsive: 3 breakpoints mínimo
- Menú adaptativo (hamburguesa en mobile)

### Requisitos No Funcionales:

- Transiciones suaves (hover, focus)
- Loading states visibles
- Contraste adecuado (accesibilidad)
- Espaciado uniforme (grid/flexbox)



**ENTREGA FINAL**

## Estructura del Proyecto:

```
[INICIALES]_Parcial_PLP3/
├── index.html (o index.php)
├── css/
│   └── [iniciales]_estilos.css
├── js/
│   └── [iniciales]_script.js
├── includes/
│   └── [iniciales]_conexion.php
├── [iniciales]_assets/
│   └── images/
├── database/
│   ├── [iniciales]_estructura.sql
│   └── [iniciales]_datos.sql
├── docs/
│   └── [APELLIDO]_[NOMBRE]_Parcial.pdf
└── README.md
```



## Método de Entrega:

1. Archivo ZIP: [\[APELLIDO\]\\_\[NOMBRE\]\\_PLP3.zip](#)
2. Repositorio GIT con commits descriptivos
3. Subir a aula virtual dentro del tiempo del examen



## EVALUACIÓN

### Puntos Bonus (+10 máximo):

- ✨ Creatividad excepcional (+3)
- 🔒 Seguridad (prepared statements) (+2)
- ♿ Accesibilidad (ARIA, semántica) (+2)

- 📱 Features avanzadas (+3)

### Penalizaciones:

- ❌ Sin nomenclatura de prefijos: -5 pts
- ❌ Código sin comentarios: -3 pts
- ❌ No funciona: -10 pts
- ❌ Plagio: 0 en el parcial

### 🎯 CHECKLIST FINAL

- ☐ Teoría completa
- ☐ Nomenclatura con prefijo
- ☐ Proyecto funcional
- ☐ BD exportada
- ☐ CSS  
responsive
- ☐ JavaScript comentado
- ☐ PHP con BD funcionando
- ☐ README.md claro
- ☐ GIT con commits
- ☐ ZIP correctamente nombrado

🚀 ¡ÉXITO, DESARROLLADOR!

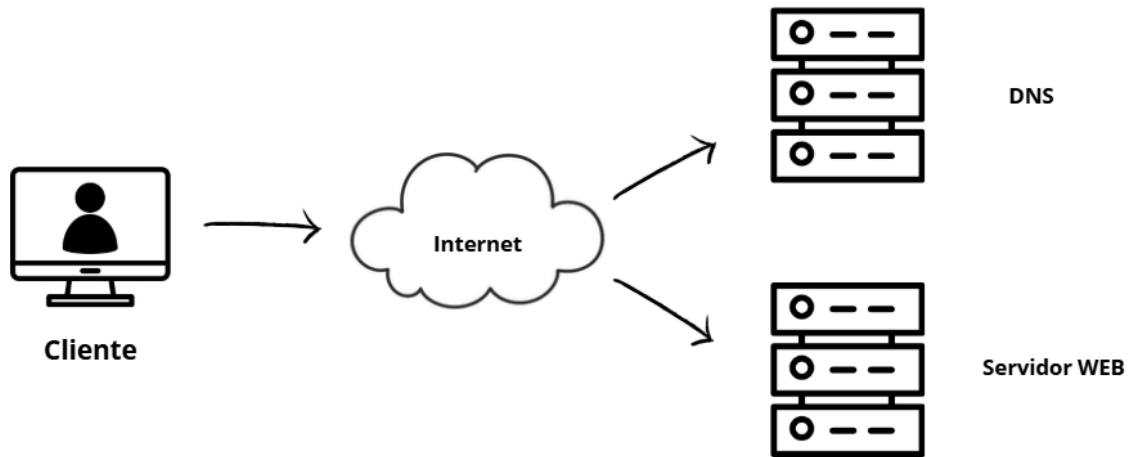
👉 ¡Que la fuerza del código esté contigo! 🎃

Estructura a utilizar

## Desarrollo

### *RUTA A: "El Arquitecto Web" (Fundamentos y Estructura)*

#### Desafío 1



A nivel conceptual, la arquitectura Cliente-Servidor describe una relación asimétrica en la que un cliente inicia solicitudes y un servidor responde con recursos o servicios. En un escenario típico de aplicación web, el cliente es el navegador que renderiza la interfaz y ejecuta JavaScript, mientras que el servidor aloja la lógica de negocio, expone endpoints y gestiona el acceso a datos.

El flujo de comunicación se inicia cuando el usuario interactúa con la interfaz; el navegador construye una petición HTTP(s) que viaja a través de la red, puede pasar por balanceadores o pasarelas, llega a un servidor de aplicaciones que valida, procesa y eventualmente consulta a una base de datos; la respuesta vuelve al cliente en forma de HTML, JSON o archivos estáticos y se representa en pantalla. Los protocolos más habituales incluyen HTTP/HTTPS para el transporte, TLS para el cifrado, HTTP/2 o HTTP/3 para multiplexación y menor latencia, WebSocket para comunicación bidireccional de baja latencia, y DNS para la resolución de nombres.

Un ejemplo real sería un sistema de turnos de salud: el usuario abre la SPA en su navegador, busca disponibilidad y envía una solicitud HTTPS a `/api/turnos`; el servidor valida credenciales con un proveedor OAuth, consulta una base de datos transaccional para horarios y un caché en memoria para reglas de cupos, compone la respuesta en JSON y el navegador la usa para pintar la grilla de turnos, manteniendo además un canal WebSocket para reflejar en tiempo real cancelaciones o nuevas disponibilidades.

#### Desafío 2

En hojas de estilo, los selectores de clase y los de id cumplen roles distintos en cuanto a semántica y especificidad. Una clase debe emplearse para definir estilos reutilizables que puedan aplicarse a múltiples elementos, favoreciendo la coherencia visual y el mantenimiento; un id, en cambio, identifica un elemento único dentro del documento y conviene reservarlo para anclas, referencias de JavaScript o estilos excepcionalmente específicos que no se repetirán.

En términos de especificidad, un id tiene mayor peso que una clase, por lo que su abuso dificulta la sobreescritura y termina generando reglas frágiles. En una interfaz de portal académico, por ejemplo, podría definirse `.tarjeta-curso` para todas las tarjetas del catálogo y `.boton-primario` para CTA reutilizables, mientras que el encabezado del usuario autenticado podría tener `#panel-perfil` como identificador único para posicionarlo o asociarlo a un menú contextual. En otro caso, una página de pagos puede utilizar `.campo-formulario` y `.estado-error` para estilizar inputs y validaciones a lo largo de todo el formulario, y reservar `#resumen-compra` para el bloque único que muestra totales y tasas. A la hora de escribir las reglas, conviene priorizar clases atómicas o semánticas y mantener los id para casos de unicidad comprobada y para ganchos de accesibilidad o scripting.

### Desafío 3

En JavaScript, una variable es un identificador asociado a un espacio de memoria que permite almacenar y manipular valores durante la ejecución del programa; posibilita representar estado, parametrizar funciones y coordinar flujos de control. Las diferencias entre `var`, `let` y `const` se centran en su alcance y comportamiento ante la redeclaración y la reasignación. `var` tiene alcance de función y se eleva (hoisting) con inicialización diferida, lo que puede producir resultados inesperados; `let` y `const` poseen alcance de bloque y también se elevan, pero quedan en la “zona muerta temporal” hasta su inicialización, evitando usos prematuros. `let` permite reasignación, mientras que `const` prohíbe cambiar la referencia, aunque el contenido interno de objetos o arreglos referenciados por un `const` sí puede mutar. Estas distinciones se observan en ejemplos como los siguientes:

// Hoisting con var: la declaración se eleva, la asignación no

```
function calculaTotal() {  
  console.log(total); // undefined, la declaración 'var total' existe pero aún no se asignó  
  var total = 120;  
  return total;  
}
```

// Alcance de bloque con let: protege variables del exterior

```
for (let i = 0; i < 3; i++) {  
  // i solo existe dentro del bloque del for  
}  
console.log(typeof i); // "undefined"
```

// const inmuta la referencia, no necesariamente el contenido

```
const configuracion = { modo: 'claro', pagina: 1 };  
// configuracion = {} // Error: no se puede reasignar  
configuracion.pagina = 2; // Válido: cambia una propiedad del objeto
```

### Desafío 4

En el ecosistema del servidor, PHP es un lenguaje interpretado de propósito general, especialmente orientado al desarrollo web por su integración nativa con servidores HTTP y su sintaxis diseñada para incrustarse en HTML. En el desarrollo moderno, PHP actúa como motor de renderizado del lado servidor y como capa de API, gestionando sesiones, autenticación, validación, acceso a bases de datos mediante PDO u ORM, colas de trabajo y servicios REST o GraphQL, tanto en arquitecturas monolíticas bien organizadas como en

microservicios. A diferencia de los lenguajes que se ejecutan en el navegador, como JavaScript del lado del cliente, PHP se ejecuta en el servidor y produce documentos o respuestas que llegan al cliente ya procesadas; el frontend se enfoca en la experiencia y la interacción, mientras que PHP resuelve persistencia, seguridad y reglas de negocio. Sus rasgos distintivos incluyen integración sencilla con servidores como Nginx o Apache, un ecosistema maduro de frameworks (por ejemplo, Laravel o Symfony), soporte amplio para extensiones y una curva de aprendizaje favorable para equipos mixtos.

Ejemplo:

```
<?php
// procesar.php
$errores = [];
$nombre = "";
$email = "";

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $nombre = trim($_POST['nombre'] ?? "");
    $email = trim($_POST['email'] ?? "");

    if ($nombre === "") { $errores[] = 'El nombre es obligatorio.'; }
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) { $errores[] = 'El email no es válido.'; }

    if (!$errores) {
        // Ejemplo: persistir con PDO
        // $pdo->prepare('INSERT INTO contactos (nombre,email) VALUES
        (? ,?)')->execute([$nombre,$email]);
        $mensajeExito = 'Gracias, registramos tu consulta.';
    }
}
?>
<!doctype html>
<html lang="es">
<head>
    <meta charset="utf-8">
    <title>Contacto</title>
</head>
<body>
    <?php if (!empty($mensajeExito)): ?>
        <p><?= htmlspecialchars($mensajeExito) ?></p>
    <?php else: ?>
        <?php if ($errores): ?>
            <div>
                <?php foreach ($errores as $e): ?>
                    <p><?= htmlspecialchars($e) ?></p>
                <?php endforeach; ?>
            </div>
        <?php endif; ?>
        <form method="post" action="procesar.php">
```

```
<label>Nombre
  <input type="text" name="nombre" value="<?= htmlspecialchars($nombre) ?>">
</label>
<label>Email
  <input type="email" name="email" value="<?= htmlspecialchars($email) ?>">
</label>
<button type="submit">Enviar</button>
</form>
<?php endif; ?>
</body>
</html>
```

Este patrón evidencia la separación de responsabilidades: la plantilla define la estructura, PHP valida y decide qué contenido renderizar, y, de ser necesario, una llamada posterior desde el cliente con fetch puede enriquecer la experiencia sin duplicar reglas de negocio. Con esta base, los cuatro ejes —arquitectura, CSS, JavaScript y PHP— quedan articulados en un flujo coherente apto para un parcial: una interfaz mantenible, un modelo de comunicación seguro y eficiente, un control preciso del estado en el navegador y una capa de servidor responsable de la integridad de los datos y de la lógica de negocio.