UNIVERSIDAD TECNOLÓGICA DE PANAMÁ
FACULTAD DE INGENIERÍA DE SISTEMAS COMPUTACIONALES
DEPARTAMENTO DE ARQUITECTURA Y REDES DE COMPUTADORA
Investigación N° 3

Facilitador(a): Hilda L. Quirós M. Asignatura: Diseño Web
Estudiante: Simon Vega **Fecha:** 23/10/24 **Grupo:** 6S3121

A. TÍTULO DE LA EXPERIENCIA: INVESTIGACIÓN LENGUAJE DE PROGRAMACIÓN PHP

B. TEMAS: FORMULARIOS

Procesamiento del Formulario con PHP.

- Formularios (action/name)
- Manejo de los datos de un formulario a través de las variables PHP
- Seguridad en el manejo del formulario
- Seguridad en el envío de datos (cifrado de datos)

Validación de los datos de un formulario, a través de Excepciones

- Esquema de funcionamiento (try-catch, throw)
- Manejo de Excepciones en los datos del formulario

C. OBJETIVO(S):

- Integrar los conceptos sobre diseño Web, programación PHP y manejo de datos en el desarrollo de una aplicación Web.

D. METODOLOGÍA SUGERIDA:

- Lectura y análisis del material suministrado.
- Investigación.
- Discusión del tema en clases.
- Discusión de la asignación y puesta en común.
- Portafolio Estudiantil.

E. ENUNCIADO:

1. **Investiga cada uno de los temas asignados y genera un informe** en el que expliques de manera detallada cada concepto.

2. **Crea** un formulario que permita a los usuarios ingresar su nombre y correo electrónico.

    - Asegúrate de que el formulario envíe los datos mediante el método POST a un archivo PHP (en el atributo action). ○ Define el atributo name para cada campo de entrada. **Punto de investigación**:

- ¿Qué significan los atributos action y name en un formulario? ○ ¿Qué métodos HTTP pueden ser utilizados en un formulario? ○ En el archivo PHP al que se envían los datos (especificado en action), usa $_POST para capturar los valores del nombre y correo electrónico ingresados.
- Muestra los datos capturados en la pantalla.
  - **Seguridad en el manejo del formulario**
  - Implementa la función htmlspecialchars() para evitar ataques XSS (Cross-site Scripting).
  - Incluye una validación básica para evitar la inyección de código en los campos del formulario.

Punto de investigación:

- ¿Qué es un ataque XSS y cómo se puede prevenir en PHP?
- ¿Qué otras medidas de seguridad se deben tener en cuenta al procesar formularios en PHP?
- **Seguridad en el envío de datos (cifrado de datos)**

- Investiga cómo se puede cifrar una contraseña en PHP usando la función password_hash() antes de almacenarla en una base de datos.

Punto de investigación:

- ¿Qué métodos de cifrado son recomendados para proteger datos sensibles?
- ¿Cómo se implementa el cifrado de contraseñas en PHP?

- **Validación de los Datos del Formulario, a través de Excepciones**

.1 Esquema de funcionamiento (try-catch, throw)

- Investiga el manejo de excepciones en PHP con las estructuras try, catch y throw.

Punto de investigación:

- ¿Cómo funcionan las excepciones en PHP?
- ¿Cuál es la sintaxis correcta para usar try, catch, y throw?
- Busca un ejemplo o Implementa un esquema de validación de datos en tu formulario que use un bloque try-catch para controlar errores comunes, como si el nombre está vacío o si el correo tiene un formato incorrecto.

- Lanza una excepción (throw) si los datos no cumplen con los requisitos. Busca ejmeplo o implementa en tu formulario.

.2 Manejo de Excepciones en los datos del formulario ○ ¿Cómo se

crean y lanzan excepciones personalizadas en PHP?

- ¿Qué ventajas tiene el uso de excepciones en comparación con otros métodos de validación?
- Busca un ejemplo o implementa en tu formulario como si el nombre del usuario está vacío o el correo no tiene un formato válido, lanza una excepción personalizada que muestre un mensaje de error.

RESULTADOS

2. Crea un formulario que permita a los usuarios ingresar su nombre y correo electrónico.

- Asegúrate de que el formulario envíe los datos mediante el método POST a un archivo PHP (en el atributo action).
- Define el atributo name para cada campo de entrada. **Punto de investigación:**
- **¿Qué significan los atributos action y name en un formulario?**

El atributo **name** permite a un script acceder a su contenido. Es preferible utilizar los dos atributos con el mismo identificador, por motivos de compatibilidad.

El atributo **action** indica la página a la que se envían los datos del formulario. Si este atributo está vacío, es la página que contiene el formulario la que se recargará con los datos como parámetros.

El atributo **onSubmit** permite asociar una función de test en el formulario. Si la función retorna falso, les datos del formulario no se envían, quedan en la página.

- **¿Qué métodos HTTP pueden ser utilizados en un formulario?**

GET: Envía los datos del formulario como parte de la URL, lo que los hace visibles en la barra de direcciones y en el historial del navegador. Se usa para solicitudes donde los datos no son sensibles.

HTML

```html
<form action="formget.php" method="get">
    Nombre: <input type="text" name="nombre"><br>
    Email: <input type="text" name="email"><br>
    <input type="submit" value="Enviar">
</form>
```

PHP

```php
<?php
if ($_SERVER["REQUEST_METHOD"] == "GET") {
   $nombre = htmlspecialchars($_GET['nombre']);
   $email = htmlspecialchars($_GET['email']);
   echo "Nombre: " . $nombre . "<br>";
   echo "Email: " . $email;
}
?>
```

URL

```
GET http://ejemplo.com/formget.php?nombre=pepe&email=pepe%40ejemplo.com
```

POST: Envía los datos en el cuerpo de la solicitud HTTP, ocultándolos de la URL, lo que lo hace más seguro y adecuado para enviar datos sensibles o formularios grandes.

HTML

```html
<form action="formpost.php" method="post">
    Nombre: <input type="text" name="nombre"><br>
    Email: <input type="text" name="email"><br>
    <input type="submit" value="Enviar">
</form>
```

PHP

```php
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
   $nombre = htmlspecialchars($_POST['nombre']);
   $email = htmlspecialchars($_POST['email']);
   echo "Nombre: " . $nombre . "<br>";
   echo "Email: " . $email;
}
?>
```

```
POST http://ejemplo.com/formpost.php
```

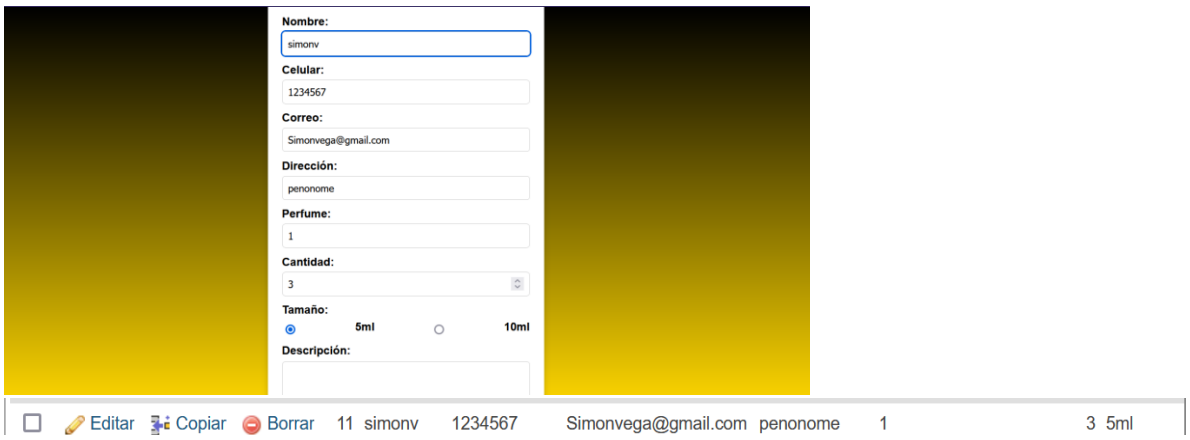
- En el archivo PHP al que se envían los datos (especificado en action), usa $_POST para capturar los valores del nombre y correo electrónico ingresados.
- Muestra los datos capturados en la pantalla.

3. Seguridad en el manejo del formulario

- Implementa la función htmlspecialchars() para evitar ataques XSS (Crosssite Scripting).
- Incluye una validación básica para evitar la inyección de código en los campos del formulario.

```
<body>

    <div id="pedido-container">
        <h2>Realiza tu Pedido</h2>

        <form action="procesar_pedido.php" method="POST">
            <label for="nombre">Nombre:</label>z
            <input type="text" id="nombre" name="nombre" required>

            <label for="celular">Celular:</label>
            <input type="tel" id="celular" name="celular" required>

            <label for="correo">Correo:</label>
            <input type="email" id="correo" name="correo" required>

            <label for="direccion">Dirección:</label>
            <input type="text" id="direccion" name="direccion" required>

            <label for="tipo_perfume">Perfume:</label>
            <input type="text" id="tipo_perfume" name="tipo_perfume" required>

            <label for="cantidad">Cantidad:</label>
            <input type="number" id="cantidad" name="cantidad" min="1" required>

            <label for="tamano">Tamaño:</label>
            <div class="radio-group">
                <input type="radio" id="5ml" name="tamano" value="5ml" required>
                <label for="5ml">5ml</label>
                <input type="radio" id="10ml" name="tamano" value="10ml" required>
                <label for="10ml">10ml</label>
            </div>
```

```
    // Preparar la consulta SQL para insertar los datos
    $stmt = $conn->prepare("INSERT INTO pedidos (nombre, celular, correo, direccion, tipo_perfume, cantida
                            VALUES (:nombre, :celular, :correo, :direccion, :tipo_perfume, :cantidad, :tam

    // Asignar los valores a cada parámetro
    $stmt->bindParam(':nombre', $_POST['nombre']);
    $stmt->bindParam(':celular', $_POST['celular']);
    $stmt->bindParam(':correo', $_POST['correo']);
    $stmt->bindParam(':direccion', $_POST['direccion']);
    $stmt->bindParam(':tipo_perfume', $_POST['tipo_perfume']);
    $stmt->bindParam(':cantidad', $_POST['cantidad']);
    $stmt->bindParam(':tamano', $_POST['tamano']);
    $stmt->bindParam(':descripcion', $_POST['descripcion']);

    // Ejecutar la consulta
    $stmt->execute();

    // Mostrar mensaje de éxito
    echo "Tu pedido ha sido realizado con éxito.";

} catch (PDOException $e) {
    // Mostrar error si no se puede conectar
    echo "Error: " . $e->getMessage();
}

// Cerrar la conexión
$conn = null;
?>
```



- **¿Qué es un ataque XSS y cómo se puede prevenir en PHP?**

Ataques XSS

Los ataques XSS ocurren cuando un atacante logra inyectar scripts maliciosos en páginas web vistas por otros usuarios. Estos scripts pueden ser ejecutados por los navegadores y realizar acciones indebidas como robo de cookies, manipulación de la sesión del usuario, redireccionamientos no autorizados y otros actos dañinos. En la práctica, el XSS explota la confianza que un usuario tiene en un sitio web determinado.

Prevención y Mitigación en PHP

- **Validación de Entradas**

Uno de los primeros pasos para prevenir un ataque XSS es validar todas las entradas del usuario. Asegúrate de que los datos recibidos coincidan con lo esperado, rechazando aquellos que contengan caracteres potencialmente peligrosos.

- **Escapado de Caracteres Especiales**

Otro mecanismo de defensa es el escapado de caracteres, que consiste en convertir caracteres especiales en entidades HTML para que no se interpreten como código. En PHP, funciones como htmlspecialchars() o htmlentities() pueden ser utilizadas para este propósito:

```
<?php
// Uso seguro de htmlspecialchars
Echo “¡Hola “ . htmlspecialchars($_GET[‘nombre’], ENT_QUOTES,
‘UTF-8’) . “!”;
¿>
```

- **Uso de Content Security Policy (CSP)**
  Imponer una política de seguridad de contenido fuerte puede ayudar a mitigar los ataques XSS. CSP permite especificar desde dónde se puede cargar el contenido y restringir la ejecución de scripts no autorizados.

- **Filtros y Validadores Especializados**
  Usar librerías de terceros que se especializan en la sanitización de datos puede proporcionar una capa adicional de seguridad. Por ejemplo, librerías como HTML Purifier permiten purificar el HTML y eliminar elementos maliciosos.

- **¿Qué otras medidas de seguridad se deben tener en cuenta al procesar formularios en PHP?**

  Implementar tokens CSRF, CAPTCHAs, campos honeypot y validar y sanitizar los datos son técnicas efectivas para mitigar estos riesgos.

4. Seguridad en el envío de datos (cifrado de datos).

- **Investiga cómo se puede cifrar una contraseña en PHP usando la función password_hash() antes de almacenarla en una base de datos.**
  Se puede utilizar la función **password_hash()** para cifrar una contraseña antes de almacenarla en una base de datos. Esta función genera un hash seguro de la contraseña utilizando algoritmos como **bcrypt** (el predeterminado) y **argon2**.

  Pasos para cifrar una contraseña usando password_hash():

  Capturar la contraseña del usuario: Primero, obtienes la contraseña ingresada por el usuario, por ejemplo, desde un formulario.

Cifrar la contraseña: Luego, utilizas la función password_hash() para generar el hash de la contraseña. Esta función toma como parámetros la contraseña sin cifrar y un algoritmo de cifrado (por defecto PASSWORD_DEFAULT que actualmente es bcrypt).

Almacenar el hash en la base de datos: Finalmente, almacenamos el hash generado en la base de datos en lugar de la contraseña en texto claro.

Punto de investigación:

- **¿Qué métodos de cifrado son recomendados para proteger datos sensibles?**

  AES (Advanced Encryption Standard): Uno de los métodos de cifrado más populares y seguros. AES utiliza claves de 128, 192 o 256 bits. Es ampliamente utilizado en aplicaciones de seguridad y es considerado muy eficiente tanto en software como en hardware.

  RSA (Rivest-Shamir-Adleman): Un algoritmo de cifrado asimétrico que utiliza un par de claves (pública y privada). Es ideal para la transmisión segura de datos a través de redes no confiables.

  ChaCha20: Un algoritmo de cifrado de flujo rápido y seguro, utilizado principalmente en aplicaciones móviles y redes de bajo ancho de banda debido a su eficiencia.

  Twofish: Un cifrado simétrico que es muy rápido y adecuado para aplicaciones que requieren tanto velocidad como seguridad.

  Elliptic Curve Cryptography (ECC): Un método de cifrado asimétrico que ofrece alta seguridad con claves más pequeñas, lo que lo hace ideal para dispositivos con recursos limitados como móviles o sistemas embebidos.

- **¿Cómo se implementa el cifrado de contraseñas en PHP?**

  Cifrado de contraseñas en PHP con password_hash():

PHP ofrece una forma nativa y sencilla de implementar el cifrado de contraseñas a través de la función password_hash(), que genera un hash seguro usando algoritmos modernos como bcrypt o Argon2. Aquí tienes una guía paso a paso para implementarlo:

Captura de la contraseña: Primero, se obtiene la contraseña ingresada por el usuario desde un formulario HTML:

```
$password = $_POST['password']; // La contraseña del formulario
```

Generación del hash con password_hash(): Utiliza la función para generar un hash seguro. El algoritmo predeterminado es bcrypt, pero

```
$hashedPassword = password_hash($password, PASSWORD_DEFAULT); // bcrypt por defecto
```

también puedes optar por Argon2 (considerado más seguro y recomendado para aplicaciones de alta seguridad):

Almacenamiento del hash: Almacena el hash generado en la base de datos en lugar de la contraseña en texto plano. En la base de datos, deberías tener una columna diseñada para almacenar cadenas largas (por ejemplo, VARCHAR(255)).

```
$sql = "INSERT INTO users (username, password) VALUES ('$username', '$hashedPassword')";
mysqli_query($conn, $sql);
```

Verificación de la contraseña: Cuando el usuario intente iniciar sesión, debes comparar la contraseña ingresada con el hash almacenado en la base de datos usando la función password_verify().

```
if (password_verify($password, $hashedPasswordFromDB)) {
    // Contraseña correcta
} else {
    // Contraseña incorrecta
}
```

5.Validación de los Datos del Formulario, a través de Excepciones

5.1. Esquema de funcionamiento (try-catch, throw)

- **Investiga el manejo de excepciones en PHP con las estructuras try, catch y throw.**

El manejo de excepciones en PHP permite a los desarrolladores capturar y manejar errores de una manera controlada, evitando que un programa falle abruptamente. Las excepciones son una forma avanzada de manejo de errores, que proporciona un flujo de control más robusto para situaciones inesperadas.

Estructura básica de manejo de excepciones

El manejo de excepciones en PHP sigue el flujo común de las estructuras de excepciones de otros lenguajes. Las palabras clave principales son:

- **Try**: Define un bloque de código que podría lanzar una excepción.
- **Catch**: Captura la excepción lanzada en el bloque try y define cómo manejarla.
- **Throw**: Lanza manualmente una excepción.

```
try {
    // Código que puede generar una excepción
    if (!$dbConnection) {
        throw new Exception("No se pudo conectar a la base de
datos.");
    }
} catch (Exception $e) {
    // Manejo de la excepción
    echo 'Excepción capturada: ' . $e->getMessage();
}
```

Try

El bloque try contiene el código que potencialmente puede lanzar una excepción. Si se produce un error dentro de este bloque, se lanza una excepción (manualmente con throw o por PHP), y se transfiere el control al bloque catch.

Throw

La palabra clave throw se usa para lanzar una excepción manualmente. Se puede crear una nueva instancia de la clase Exception o de una clase que extienda Exception para encapsular un mensaje de error o información adicional.

Catch

El bloque catch captura la excepción lanzada. La excepción capturada se pasa como un objeto, lo que permite acceder a métodos como getMessage() para obtener el mensaje de error, getCode() para el código de error, getFile() para el archivo donde ocurrió la excepción, y getLine() para obtener la línea donde se produjo.

Punto de investigación:

- **¿Cómo funcionan las excepciones en PHP?**
  Las excepciones en PHP permiten manejar errores de manera controlada y estructurada, en lugar de dejar que el programa falle abruptamente. En esencia, las excepciones son una forma de controlar el flujo de ejecución de un programa cuando ocurre una situación excepcional o un error.

  Cuando una excepción es lanzada (throw), el flujo normal del programa se detiene y PHP intenta encontrar un bloque catch para manejar esa excepción. Si no se encuentra ningún bloque catch, el programa termina con un error fatal. Si se encuentra un bloque catch, el control del programa pasa a ese bloque, donde se puede manejar el error de manera específica.

- **¿Cuál es la sintaxis correcta para usar try, catch, y throw?**

  Sintaxis correcta para usar try, catch, y throw

```
try {
    // Código que puede generar una excepción
    if (/* condición que causa un error */) {
        throw new Exception("Mensaje de error");
    }
} catch (Exception $e) {
    // Manejo de la excepción
    echo 'Error: ' . $e->getMessage();
} finally {
    // (Opcional) Código que siempre se ejecuta, con o sin excepción
    echo 'Bloque finally ejecutado.';
}
```

  Try { }:

- Define un bloque de código que potencialmente puede generar una excepción.

- Si ocurre una excepción dentro del bloque try, se interrumpe la ejecución del bloque y se pasa al bloque catch.

Throw new Exception('mensaje'):

- Lanza manualmente una excepción.
- La palabra clave throw se utiliza para generar una excepción. Se puede crear una instancia de la clase Exception o una de sus subclases para definir un mensaje de error y, opcionalmente, un código de error.

```
if (!$connection) {
    throw new Exception("Error al conectar a la base de datos.");
}
```

catch (Exception $e) { … }:

- Define cómo manejar la excepción lanzada.
- La variable $e es una instancia del objeto Exception, que proporciona métodos como getMessage(), getCode(), getFile(), y getLine() para obtener detalles sobre la excepción.

```
catch (Exception $e) {
    echo 'Error capturado: ' . $e->getMessage();
}
```

finally { … } (opcional):

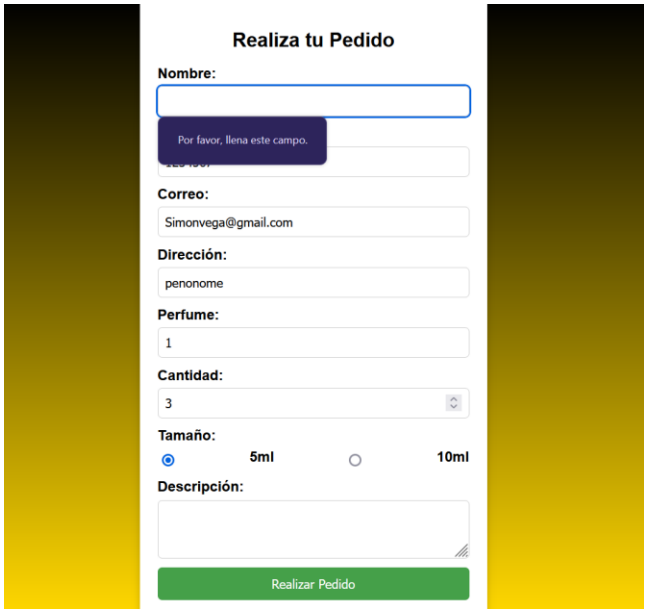
- Este bloque siempre se ejecuta, ya sea que ocurra o no una excepción. Es útil para realizar tareas de limpieza, como cerrar conexiones o liberar recursos.

```
finally {
    echo 'Bloque finally ejecutado.';
}
```

- Busca un ejemplo o Implementa un esquema de validación de datos en tu formulario que use un bloque try-catch para controlar errores comunes, como si el nombre está vacío o si el correo tiene un formato incorrecto.

- Lanza una excepción (throw) si los datos no cumplen con los requisitos.

Busca ejmeplo o implementa en tu formulario.



5.2. Manejo de Excepciones en los datos del formulario

- ¿Cómo se crean y lanzan excepciones personalizadas en PHP? **Pasos para crear y lanzar excepciones personalizadas:**

- **Crear una clase que extienda Exception**:
  Define una nueva clase que extienda la clase Exception. Puedes personalizar esta clase agregando propiedades o métodos adicionales, o simplemente usarla como está.

- **Lanzar la excepción personalizada**:
  Usa la palabra clave throw para lanzar la excepción cuando ocurra una condición que deba ser manejada de manera especial.

- **Capturar la excepción personalizada**:
  Usa un bloque catch para capturar la excepción personalizada y definir cómo manejarla.

  ejemplo completo que muestra cómo crear, lanzar y manejar una excepción personalizada:

```php
<?php
// Creación de la clase de excepción personalizada
class MiExcepcionPersonalizada extends Exception {
    // Constructor para inicializar el mensaje y otros parámetros
    public function __construct($mensaje, $codigo = 0, Exception $previa = null) {
        // Llamamos al constructor de la clase base (Exception)
        parent::__construct($mensaje, $codigo, $previa);
    }

    // Método personalizado que puedes agregar para representar la excepción como cadena
    public function __toString() {
        return __CLASS__ . ": [{$this->code}]: {$this->message}\n";
    }

    // Puedes agregar métodos adicionales si es necesario
    public function obtenerDetalles() {
        return "Detalles de la excepción: [Código: {$this->code}] - Mensaje: {$this->messa
    }
}
```

Captura de datos del formulario:
$username = $_POST['username']; captura el nombre de usuario enviado desde un formulario mediante el método POST. **Validación:**

Se verifica si la variable **$username** está vacía con **empty()**. Si lo está, se lanza una excepción con el mensaje 'El nombre de usuario es obligatorio.' Usando throw.
Manejo de la excepción:
Si se lanza la excepción, el bloque catch la captura, y muestra el mensaje de error con echo '**Error: ' . $e->getMessage();.**

En este ejemplo, capturamos una excepción si el campo de nombre de usuario está vacío. Lanzamos una excepción personalizada con un mensaje descriptivo y lo mostramos al usuario.

- **¿Qué ventajas tiene el uso de excepciones en comparación con otros métodos de validación?**

Separación de la lógica de errores y la lógica principal: Las excepciones permiten mantener la lógica del programa y el manejo de errores separados, lo que mejora la legibilidad y organización del código.

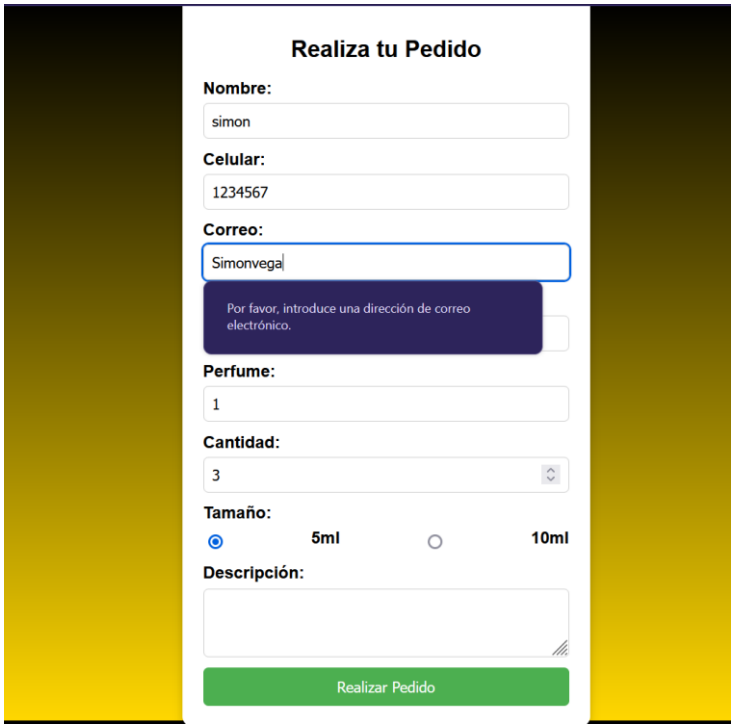
Manejo centralizado de errores: Las excepciones permiten agrupar el manejo de errores en un solo lugar, facilitando un control unificado de los errores en lugar de dispersar la lógica de manejo de errores en múltiples partes del código.

Reutilización del manejo de errores: Al lanzar excepciones personalizadas, puedes reutilizar la lógica de manejo de errores en varias partes del código sin necesidad de duplicar validaciones.

Flujo de control flexible: Las excepciones permiten abandonar el flujo normal del programa inmediatamente cuando ocurre un error, lo que evita continuar con la ejecución de código no necesario después de un fallo.

Escalabilidad y mantenimiento: El uso de excepciones facilita el manejo de errores en aplicaciones grandes o complejas, permitiendo un enfoque más modular y escalable. Los cambios en la lógica de manejo de errores se aplican de manera más centralizada.

• Busca un ejemplo o implementa en tu formulario como si el nombre del usuario está vacío o el correo no tiene un formato válido, lanza una excepción personalizada que muestre un mensaje de error.


Realiza tu Pedido
Nombre:
simon
Celular:
1234567
Correo:
Simonvega
Por favor, introduce una dirección de correo electrónico.
Perfume:
1
Cantidad:
3
Tamaño:
5ml
10ml
Descripción:
Realizar Pedido

CRITERIOS DE EVALUACIÓN

Se tomará como nota de Parcial #3

CRITERIOS A EVALUAR EN EL TALLER	PUNTAJE ASIGNADO	PUNTAJE LOGRADO
Investigación y claridad del informe,	40	
Correcta implementación del formulario en PHP.	30	
Seguridad aplicada en el manejo de los datos.	15	
Validación de datos y manejo de excepciones.	15	

BIBLIOGRAFÍA

• Primeros pasos con PHP – 2015.

✓ Disponible en URL: https://www.freelibros.org/diseno-web/video2brain-primerospasoscon-php-2015

• Desarrollar un sitio Web dinámico e interactivo – Olivier HEURTEL

✓ Disponible en URL: https://www.freelibros.org/programacion/php-5-5-desarrollar-unsitioweb-dinamico-e-interactivo-olivier-heurtel

• Aprender a desarrollar un sitio Web con PHP y MySQL: Ejercicios prácticos y corregidos – Olivier ROLLET.

✓ Disponible en URL: https://www.freelibros.org/programacion/aprender-a-desarrollarunsitio-web-con-php-y-mysql-ejercicios-practicos-y-corregidos-olivier-rollet

• Bartolomé Sintes Marco, ***"Programación web en PHP: reglas de estilo"***

✓ Disponible en URL: http://www.mclibre.org/consultar/php/otros/ot_guiaestilo.html

• https://www.w3schools.com/php/default.asp