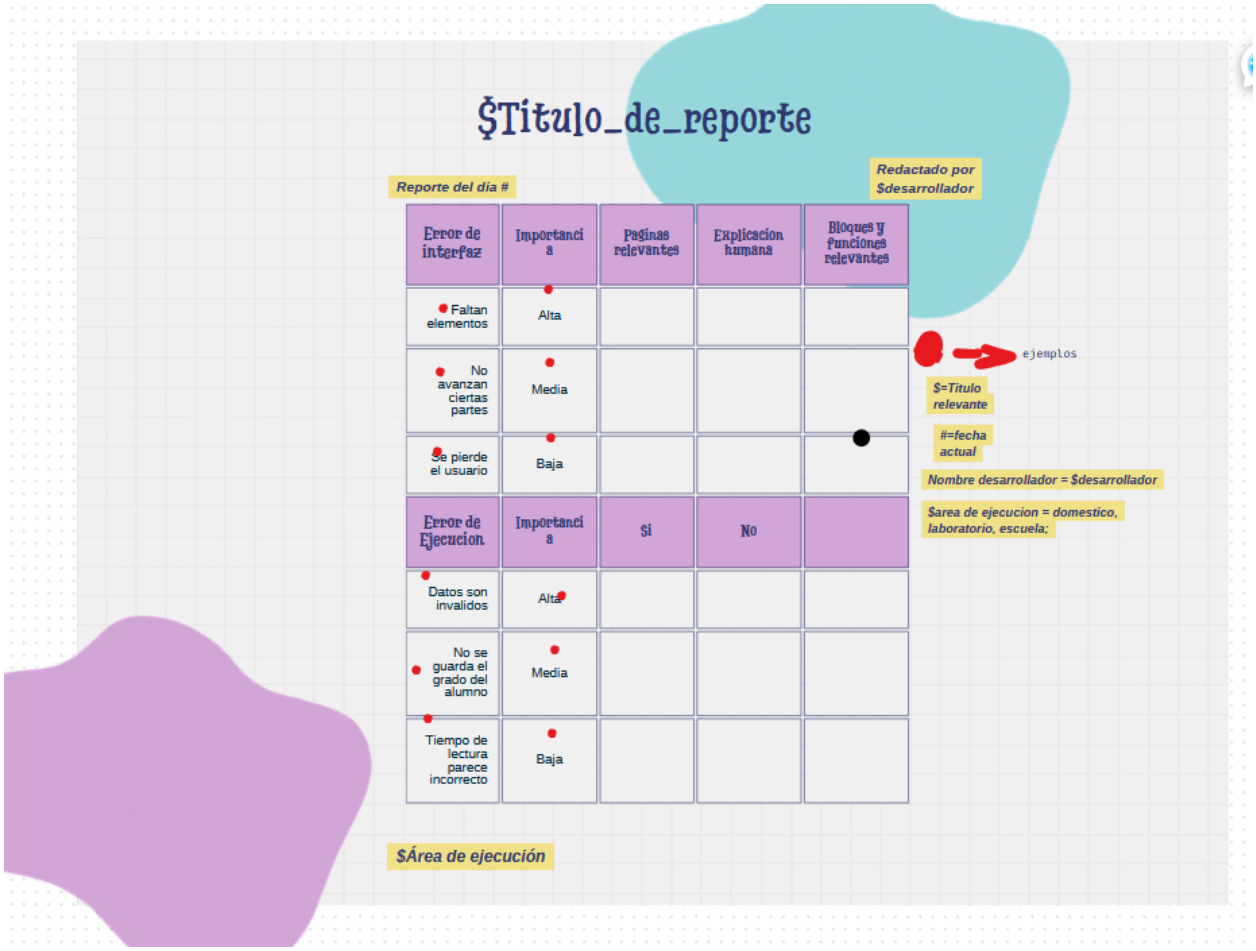
Actividad 1:

Realizar reporte de errores con el nuevo formato (sí, se entrega digital, pero lo presento aquí para propósitos de documentación)



Actividad 2:

Documentar con comentarios los archivos que pertenecen a la aplicación web que están desarrollando (en español, la documentación)

Validación:

script.js

```
// Sistema de validación para Registro de CBNoticias
document.addEventListener('DOMContentLoaded', function() {

  const formulario = document.getElementById('registerForm');
  const botonEnviar = document.getElementById('submitBtn');
  const camposTexto = ['nombre', 'correo', 'usuario', 'clave', 'telefono'];

  const patrones = {
    nombre: /^[a-záéíóúñA-ZÁÉÍÓÚÑ\s]+$/,
    correo: /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/,
    usuario: /^[a-zA-Z0-9_]{3,20}$/,
    clave: /^.{6,}$/,
    telefono: /^[0-9]{10}$/
  };

  // Mensajes de ayuda
  const mensajes = {
    nombre: 'Solo letras y espacios',
    correo: 'usuario@cbtis03.edu',
    usuario: '3-20 caracteres (letras, números, guiones bajos)',
    clave: 'Mínimo 6 caracteres',
    telefono: '10 dígitos (opcional)'
  };

  // Estado de validación de campos
  const estadoValido = {
    nombre: true,
    correo: true,
    usuario: false,
    clave: false,
    telefono: true
  };
```

Nuestra validación empieza con un event listener que le pone atención a los elementos enviados por el usuario, definimos las constantes importantes (los considero las ‘partes que no se mueven’) Y el estado inicial de validación, por razones de debugging puse algunos como true, pero en la versión final deben ser falsos por default.

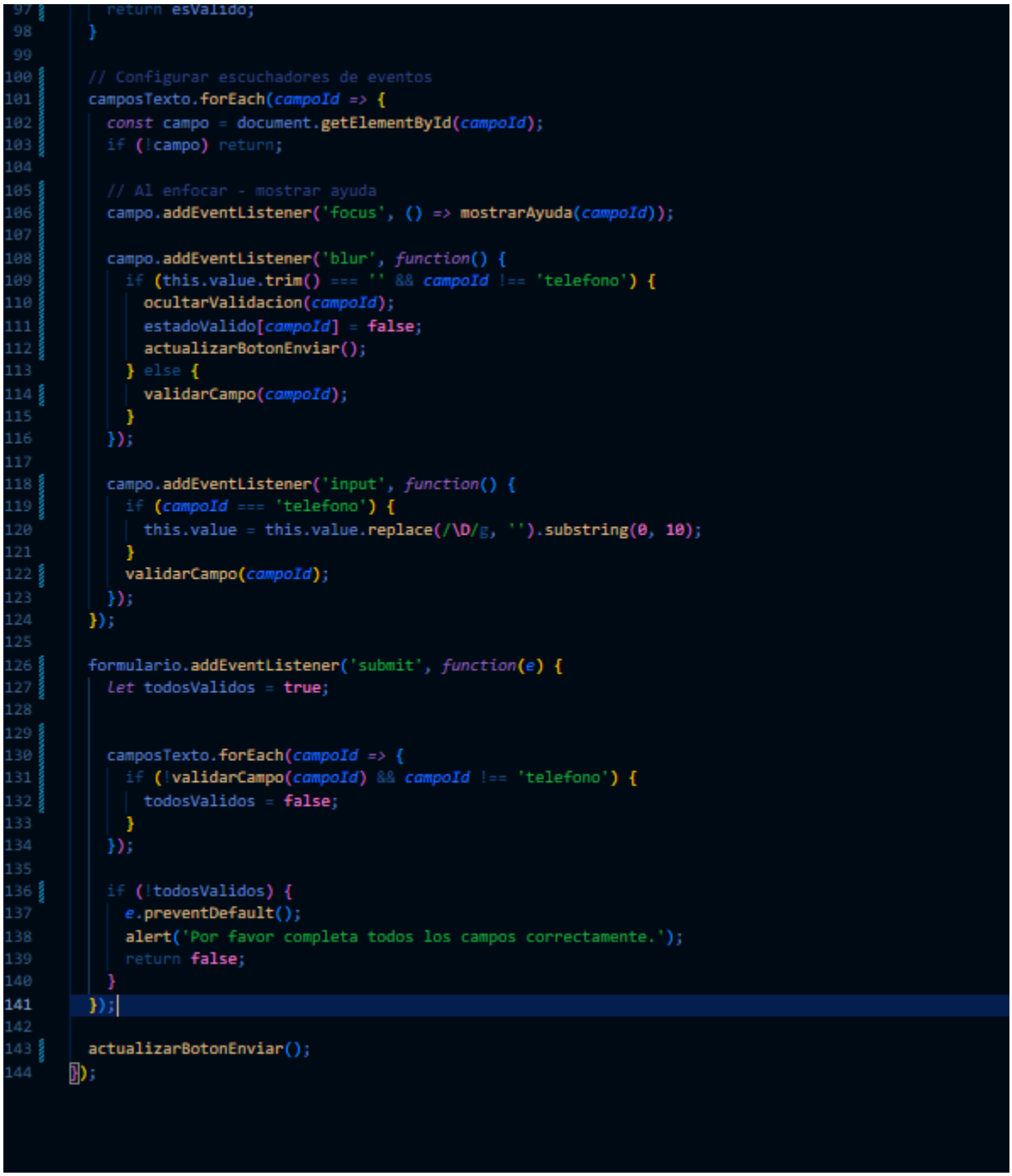

script.js M
script.js > document.addEventListener('DOMContentLoaded') callback > validarCampo
2 document.addEventListener('DOMContentLoaded', function() {
34
35
36 function actualizarBotonEnviar() {
37 const todosValidos = Object.values(estadoValido).every(Boolean);
38 botonEnviar.disabled = !todosValidos;
39 }
40
41 function mostrarAyuda(campoId) {
42 const elementoMsg = document.getElementById(campoId + 'Msg');
43 if (elementoMsg && mensajes[campoId]) {
44 elementoMsg.textContent = mensajes[campoId];
45 elementoMsg.className = 'validation-box show neutral';
46 }
47 }
48
49 function ocultarValidacion(campoId) {
50 const elementoMsg = document.getElementById(campoId + 'Msg');
51 if (elementoMsg) {
52 elementoMsg.className = 'validation-box';
53 elementoMsg.textContent = '';
54 }
55 }
56
57 function validarCampo(campoId) {
58 const campo = document.getElementById(campoId);
59 if (!campo) return false;
60
61 const elementoMsg = document.getElementById(campoId + 'Msg');
62 const valor = campo.value.trim();
63
64 // Manejar campos vacío
65 if (valor === '') {
66 // El teléfono es opcional
67 if (campoId === 'telefono') {
68 ocultarValidacion(campoId);
69 estadoValido[campoId] = true;
70 campo.className = '';
71 actualizarBotonEnviar();
72 return true;
73 } else {
74 ocultarValidacion(campoId);
75 estadoValido[campoId] = false;
76 campo.className = '';
77 actualizarBotonEnviar();
78 return false;
79 }
80 }

Luego, desarrollamos las funciones que vamos a utilizar para la validacion y configuracion de las cajas encargo de demostrar el estado de validacion actual, en base a las constantes establecidas aqui configuramos un poco.

```
    // Validar contra patrón
    const esValido = patrones[campoId].test(valor);
    estadoValido[campoId] = esValido;

    if (esValido) {
        elementoMsg.textContent = '✓ ' + mensajes[campoId];
        elementoMsg.className = 'validation-box show valid';
        campo.className = 'success';
    } else {
        elementoMsg.textContent = '✗ ' + mensajes[campoId];
        elementoMsg.className = 'validation-box show invalid';
        campo.className = 'error';
    }

    actualizarBotonEnviar();
    return esValido;
}
```

Enviamos lo que se necesite y vemos si el botón de enviar aparece en la pantalla del usuario tras ingresar de forma correcta la información.

Ahora agregue unos eventos para incrementar la interacción que siente el usuario. Y finalizar el cheque previo (pues quiero hacer el numero de telefono opcional).



```
    return esValido;
  }

  // Configurar escuchadores de eventos
  camposTexto.forEach(campoId => {
    const campo = document.getElementById(campoId);
    if (!campo) return;

    // Al enfocar - mostrar ayuda
    campo.addEventListener('focus', () => mostrarAyuda(campoId));

    campo.addEventListener('blur', function() {
      if (this.value.trim() === '' && campoId !== 'telefono') {
        ocultarValidacion(campoId);
        estadoValido[campoId] = false;
        actualizarBotonEnviar();
      } else {
        validarCampo(campoId);
      }
    });

    campo.addEventListener('input', function() {
      if (campoId === 'telefono') {
        this.value = this.value.replace(/\D/g, '').substring(0, 10);
      }
      validarCampo(campoId);
    });
  });

  formulario.addEventListener('submit', function(e) {
    let todosValidos = true;


    camposTexto.forEach(campoId => {
      if (!validarCampo(campoId) && campoId !== 'telefono') {
        todosValidos = false;
      }
    });

    if (!todosValidos) {
      e.preventDefault();
      alert('Por favor completa todos los campos correctamente.');
      return false;
    }
  });

  actualizarBotonEnviar();
});
```

registrar.php

Estoy implementando validación PHP muy básica en caso de que el JavaScript se interponga mágicamente (como tanto le encanta hacer):

registrar.php > ...

```php
<?php
require_once 'lib/common.php';
session_start();

$success = false;
$error = '';

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $pdo = getPDO();

    // Get and sanitize input
    $nombre = trim(string: $_POST["nombre"] ?? '');
    $email = trim(string: $_POST["correo"] ?? '');
    $usuario = trim(string: $_POST["usuario"] ?? '');
    $clave = $_POST["clave"] ?? '';
    $telefono = trim(string: $_POST["telefono"] ?? '');
    $genero_lit_fav = $_POST["genero_lit_fav"] ?? '';

    try {
        // Validation
        if (empty($nombre) || empty($email) || empty($usuario) || empty($clave)) {
            $error = "Todos los campos obligatorios deben ser completados.";
        }
        elseif (userExists(pdo: $pdo, usuario: $usuario)) {
            $error = "El usuario ya existe. Por favor elige otro.";
        }
        elseif (emailExists(pdo: $pdo, email: $email)) {
            $error = "El correo ya está registrado.";
        }
        else {
            // Insert new user - NO PASSWORD HASHING (to match login)
            $stmt = $pdo->prepare(query: "
                INSERT INTO user (usuario, nombre, email, clave, fecha_registro, grade, genero_lit_fav)
                VALUES (:usuario, :nombre, :email, :clave, CURRENT_TIMESTAMP, 1, :genero_lit_fav)
            ");

            $result = $stmt->execute(params: [
                ':usuario' => $usuario,
                ':nombre' => $nombre,
                ':email' => $email,
                ':clave' => $clave, // Plain text to match your login.php
                ':genero_lit_fav' => $genero_lit_fav
            ]);

            if ($result) {
                $success = true;
                // Auto-login after registration
                login(usuario: $usuario, nombre: $nombre, genero_lit_fav: $genero_lit_fav);
                header(header: "refresh:1;url=LP.php");
            } else {
                $error = "Error al crear la cuenta. Intenta nuevamente.";
            }
        }
    } catch (PDOException $e) {
        error_log(message: "DB Error: " . $e->getMessage());
        $error = "Error al registrarse: " . $e->getMessage();
    }
```

Es tan simple que explicarla puede ser un poco difícil:

registrar.php > ...

```php
<?php
require_once 'lib/common.php';
session_start();

$success = false;
$error = '';

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $pdo = getPDO();

```

Aquí arriba tenemos la conexión a mi biblioteca de funciones, y el inicio de la sesión (como esta es la página de registro, no requiere que el usuario tenga una sesión iniciada obv). Ademas de contar con 2 variables, establecemos que esta base de datos realmente existe (gracias getPDO()) y estamos usando el metodo POST porque vamos a enviar esta informacion luego al servidor. Session_start se asegura de que sea un post.

```php
$nombre = trim(string: $_POST["nombre"] ?? '');
$email = trim(string: $_POST["correo"] ?? '');
$usuario = trim(string: $_POST["usuario"] ?? '');
$clave = $_POST["clave"] ?? '';
$telefono = trim(string: $_POST["telefono"] ?? '');
$genero_lit_fav = $_POST["genero_lit_fav"] ?? '';

try {
    // Validation
    if (empty($nombre) || empty($email) || empty($usuario) || empty($clave)) {
        $error = "Todos los campos obligatorios deben ser completados.";
    }
    elseif (userExists(pdo: $pdo, usuario: $usuario)) {
        $error = "El usuario ya existe. Por favor elige otro.";
    }
    elseif (emailExists(pdo: $pdo, email: $email)) {
        $error = "El correo ya está registrado.";
    }
    else {
        // Insert new user - NO PASSWORD HASHING (to match login)
        $stmt = $pdo->prepare(query: "
            INSERT INTO user (usuario, nombre, email, clave, fecha_registro, grade, genero_lit_fav)
            VALUES (:usuario, :nombre, :email, :clave, CURRENT_TIMESTAMP, 1, :genero_lit_fav)
        ");

        $result = $stmt->execute(params: [
            ':usuario' => $usuario,
            ':nombre' => $nombre,
            ':email' => $email,
            ':clave' => $clave, // Plain text to match your login.php
            ':genero_lit_fav' => $genero_lit_fav
        ]);
```

Luego, como pescador, lanzamos nuestra carnada (post) hacia el infinito (nuestra base de datos) para recoger la información de nuestro usuario (me perdí la metáfora q estaba haciendo) y limpiarla. Este bloque antes de nuestro “try” es seguridad básica y según reduce la cantidad de espacio que toma la bd.

```php
try {
    // Validation
    if (empty($nombre) || empty($email) || empty($usuario) || empty($clave)) {
        $error = "Todos los campos obligatorios deben ser completados.";
    }
    elseif (userExists(pdo: $pdo, usuario: $usuario)) {
        $error = "El usuario ya existe. Por favor elige otro.";
    }
    elseif (emailExists(pdo: $pdo, email: $email)) {
        $error = "El correo ya está registrado.";
    }
    else {
        // Insert new user - NO PASSWORD HASHING (to match login)
        $stmt = $pdo->prepare(query: "
            INSERT INTO user (usuario, nombre, email, clave, fecha_registro, grade, genero_lit_fav)
            VALUES (:usuario, :nombre, :email, :clave, CURRENT_TIMESTAMP, 1, :genero_lit_fav)
        ");

        $result = $stmt->execute(params: [
            ':usuario' => $usuario,
            ':nombre' => $nombre,
            ':email' => $email,
            ':clave' => $clave, // Plain text to match your login.php
            ':genero_lit_fav' => $genero_lit_fav
        ]);
```

¿Vez donde empieza el if? Pues nuestra “validación” empieza ahíy termina en el siguiente punto y coma. Te dije que era básica jaja, pero igual funciona como respaldo. Másimportante que eso, lo usamos como anclaje para validar si el usuario ya existía o no- cosa que nuestro JavaScript no maneja.

Luego de validar, podemos preparar la inserción de nuestro usuario a la base de datos (sin utilizar hashing porque OMG es difícil ¿okay?)

```php
            ]);

            if ($result) {
                $success = true;
                // Auto-login after registration
                login(usuario: $usuario, nombre: $nombre, genero_lit_fav: $genero_lit_fav);
                header(header: "refresh:1;url=LP.php");
            } else {
                $error = "Error al crear la cuenta. Intenta nuevamente.";
            }
        }
    } catch (PDOException $e) {
        error_log(message: "DB Error: " . $e->getMessage());
        $error = "Error al registrarse: " . $e->getMessage();
    }
}
?>
```

Esta parte es interesante, al parecer es más intuitivo automáticamente iniciar la sesión del usuario y dirigirlos a nuestra página protegida sin pedirles nuevamente su información de ingreso. Aparte de intuitivo, reduce el tamaño local de la sesión, así que es un poco más seguro.

De ahi cabe contar lo que tanto me intriga de PHP:

```php
<div class="container">
  <form action="registrar.php" method="post" class="form" id="registerForm">
    <h2>📝 Crear Cuenta</h2>

    <?php if ($error): ?>
      <div class="alert alert-error">
        <?php echo htmlEscape(html: $error); ?>
      </div>
    <?php endif; ?>

    <?php if ($success): ?>
      <div class="alert alert-success">
        ✅ ¡Cuenta creada exitosamente! Redirigiendo...
      </div>
    <?php endif; ?>

    <div class="input-wrapper">
      <input type="text" name="nombre" id="nombre" placeholder="Nombre completo"
             value="<?php echo isset($_POST['nombre']) ? htmlEscape(html: $_POST['nombre']) : ''; ?>"
             required>
      <div class="validation-box" id="nombreMsg"></div>
    </div>

    <div class="input-wrapper">
      <input type="email" name="correo" id="correo" placeholder="Correo electrónico"
             value="<?php echo isset($_POST['correo']) ? htmlEscape(html: $_POST['correo']) : ''; ?>"
             required>
      <div class="validation-box" id="correoMsg"></div>
    </div>

    <div class="input-wrapper">
      <input type="text" name="usuario" id="usuario" placeholder="Usuario (3-20 caracteres)"
             value="<?php echo isset($_POST['usuario']) ? htmlEscape(html: $_POST['usuario']) : ''; ?>"
             required>
      <div class="validation-box" id="usuarioMsg"></div>
    </div>

    <div class="input-wrapper">
      <input type="password" name="clave" id="clave" placeholder="Contraseña (mín. 6 caracteres)" required>
      <div class="validation-box" id="claveMsg"></div>
    </div>

    <div class="input-wrapper">
      <input type="tel" name="telefono" id="telefono" placeholder="Teléfono (opcional)"
             value="<?php echo isset($_POST['telefono']) ? htmlEscape(html: $_POST['telefono']) : ''; ?>">
      <div class="validation-box" id="telefonoMsg"></div>
    </div>

    <div class="input-wrapper">
      <select name="genero_lit_fav" id="genero_lit_fav">
        <option value="">Selecciona tu género literario favorito (opcional)</option>
        <option value="Ficción" <?php echo (isset($_POST['genero_lit_fav']) && $_POST['genero_lit_fav'] === 'Ficción') ? 'selected' : ''; ?>>Ficción</option>
        <option value="No Ficción" <?php echo (isset($_POST['genero_lit_fav']) && $_POST['genero_lit_fav'] === 'No Ficción') ? 'selected' : ''; ?>>No Ficción</option>
        <option value="Ciencia Ficción" <?php echo (isset($_POST['genero_lit_fav']) && $_POST['genero_lit_fav'] === 'Ciencia Ficción') ? 'selected' : ''; ?>>Ciencia Ficción</option>
        <option value="Romance" <?php echo (isset($_POST['genero_lit_fav']) && $_POST['genero_lit_fav'] === 'Romance') ? 'selected' : ''; ?>>Romance</option>
        <option value="Misterio" <?php echo (isset($_POST['genero_lit_fav']) && $_POST['genero_lit_fav'] === 'Misterio') ? 'selected' : ''; ?>>Misterio</option>
        <option value="Fantasía" <?php echo (isset($_POST['genero_lit_fav']) && $_POST['genero_lit_fav'] === 'Fantasía') ? 'selected' : ''; ?>>Fantasía</option>
        <option value="Horror" <?php echo (isset($_POST['genero_lit_fav']) && $_POST['genero_lit_fav'] === 'Horror') ? 'selected' : ''; ?>>Horror</option>
        <option value="Biografía" <?php echo (isset($_POST['genero_lit_fav']) && $_POST['genero_lit_fav'] === 'Biografía') ? 'selected' : ''; ?>>Biografía</option>
        <option value="Poesía" <?php echo (isset($_POST['genero_lit_fav']) && $_POST['genero_lit_fav'] === 'Poesía') ? 'selected' : ''; ?>>Poesía</option>
        <option value="General" <?php echo (isset($_POST['genero_lit_fav']) && $_POST['genero_lit_fav'] === 'General') ? 'selected' : ''; ?>>General</option>
      </select>
    </div>
```

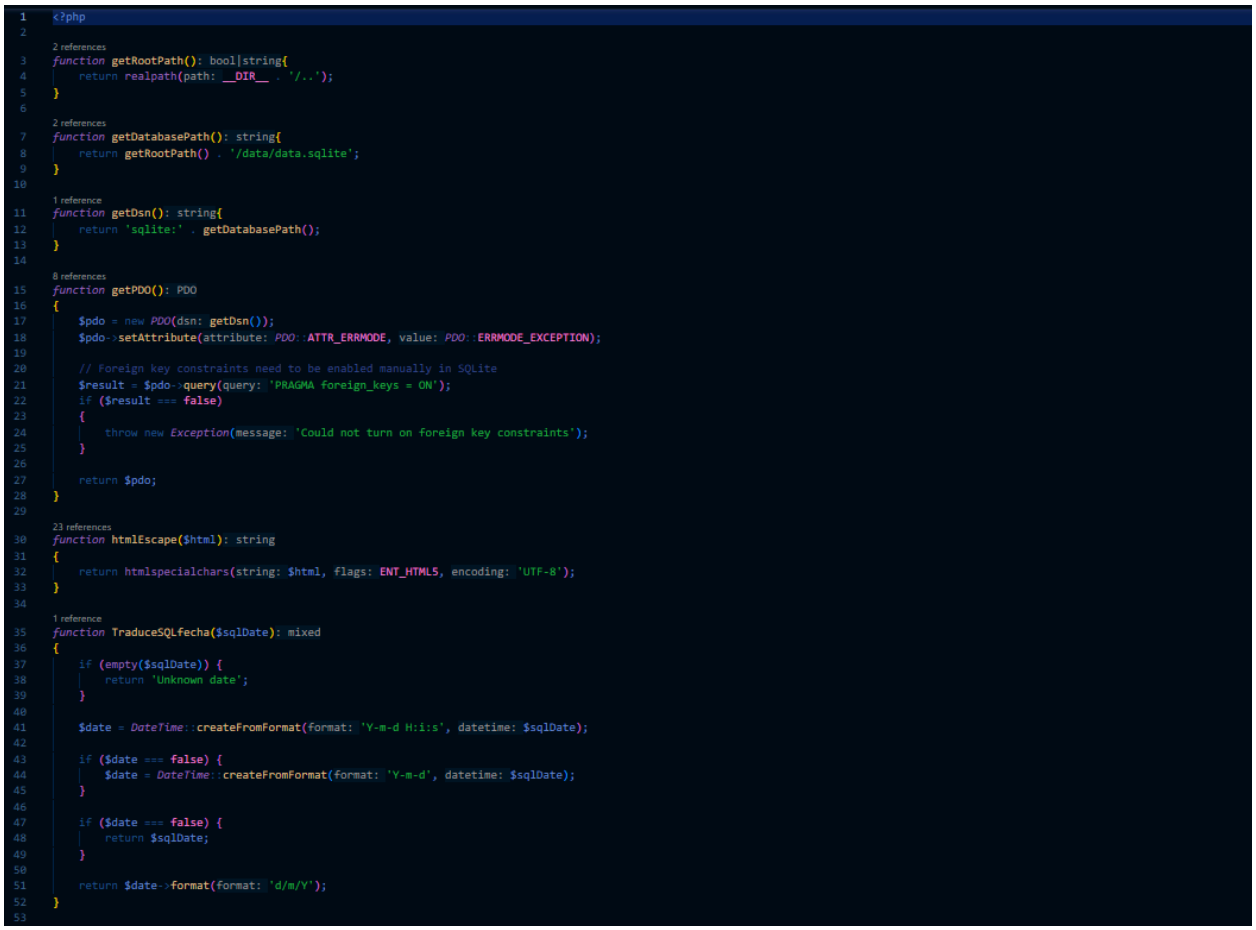
Su relación simbiótica con HTML una vez que desarrollas, es tan hermosa- Es como un remolino entre el agua y cielo, creando nada más que naturaleza predictiva. Lo que está escrito es absoluto, y si PHP o HTML o una de las muchas extensiones de SQL y apache nota que has escrito algo mal, simplemente intentara cumplir su función sin interrumpirte. Lo apreciable de eso es la fortaleza con cuál toman tus errores de ortografía. Lo que sería fácil de ver en un ensayo se ofusca aquí detrás de una capa cual tiene decenas de bibliotecas y recursos educativos.

Está bajo las manos del desarrollador la virtud de corregir y entender su codigo.

Biblioteca

Common.php

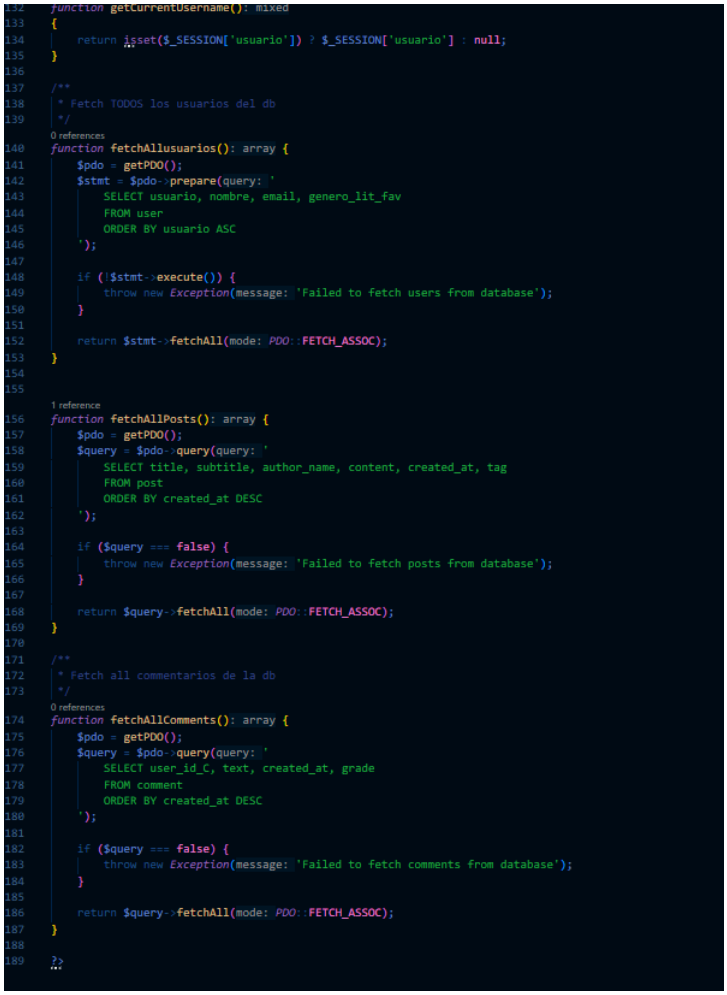
Este codigo es facilmente el cual mas me importa como desarrollador;



```php
<?php

2 references
function getRootPath(): bool|string{
    return realpath(path: __DIR__ . '/..');
}

2 references
function getDatabasePath(): string{
    return getRootPath() . '/data/data.sqlite';
}

1 reference
function getDsn(): string{
    return 'sqlite:' . getDatabasePath();
}

8 references
function getPDO(): PDO
{
    $pdo = new PDO(dsn: getDsn());
    $pdo->setAttribute(attribute: PDO::ATTR_ERRMODE, value: PDO::ERRMODE_EXCEPTION);

    // Foreign key constraints need to be enabled manually in SQLite
    $result = $pdo->query(query: 'PRAGMA foreign_keys = ON');
    if ($result === false)
    {
        throw new Exception(message: 'Could not turn on foreign key constraints');
    }

    return $pdo;
}

23 references
function htmlEscape($html): string
{
    return htmlspecialchars(string: $html, flags: ENT_HTML5, encoding: 'UTF-8');
}

1 reference
function TraduceSQLfecha($sqlDate): mixed
{
    if (empty($sqlDate)) {
        return 'Unknown date';
    }

    $date = DateTime::createFromFormat(format: 'Y-m-d H:i:s', datetime: $sqlDate);

    if ($date === false) {
        $date = DateTime::createFromFormat(format: 'Y-m-d', datetime: $sqlDate);
    }

    if ($date === false) {
        return $sqlDate;
    }

    return $date->format(format: 'd/m/Y');
}
```

Una simple y corta biblioteca cuál visualizo como mi ‘caja de arena’ para probar y compilar todo lo que he visto. La verdad amó esta cosa:

En lo personal, es la primera vez que trabajo con sesiones asi que he tenido diversion descubriendo todos los metodos y varia logica que podria usar para profundizar mi uso de este simple concepto. Se ve que esta lejos de ser perfecto:



```php
function getCurrentUsername(): mixed
{
    return isset($_SESSION['usuario']) ? $_SESSION['usuario'] : null;
}

/**
 * Fetch TODOS los usuarios del db
 */
0 references
function fetchAllusuarios(): array {
    $pdo = getPDO();
    $stmt = $pdo->prepare(query: '
        SELECT usuario, nombre, email, genero_lit_fav
        FROM user
        ORDER BY usuario ASC
    ');

    if (!$stmt->execute()) {
        throw new Exception(message: 'Failed to fetch users from database');
    }

    return $stmt->fetchAll(mode: PDO::FETCH_ASSOC);
}


1 reference
function fetchAllPosts(): array {
    $pdo = getPDO();
    $query = $pdo->query(query: '
        SELECT title, subtitle, author_name, content, created_at, tag
        FROM post
        ORDER BY created_at DESC
    ');

    if ($query === false) {
        throw new Exception(message: 'Failed to fetch posts from database');
    }

    return $query->fetchAll(mode: PDO::FETCH_ASSOC);
}

/**
 * Fetch all commentarios de la db
 */
0 references
function fetchAllComments(): array {
    $pdo = getPDO();
    $query = $pdo->query(query: '
        SELECT user_id_C, text, created_at, grade
        FROM comment
        ORDER BY created_at DESC
    ');

    if ($query === false) {
        throw new Exception(message: 'Failed to fetch comments from database');
    }

    return $query->fetchAll(mode: PDO::FETCH_ASSOC);
}

?>
```

Ha sido una excelente herramienta y lo mejor es que cada línea de codigo que se ve aca, termina siendo como 2 líneas que nunca debo escribir de nuevo. Y si decido ‘refactorizar’ esta biblioteca, logro sentir un profundo miedo al no actualizar todas sus referencias.

Actividad 3:

Documenta con comentarios (los bloques de codigo) que contenga las operaciones y cálculos.

Read.php

```php
<?php
require_once 'lib/common.php';
session_start();
requiereLogin();


$blogs = fetchAllPosts();

// Calculate word count and reading time for each blog
foreach ($blogs as &$blog) {
    $wordCount = str_word_count(string: strip_tags(string: $blog['content']));
    $blog['palabra_count'] = $wordCount;
    $blog['tiempo_lectura'] = max(value: 1, values: ceil(num: $wordCount / 200)); // Assuming 200 words per minute

    $blog['titulo'] = $blog['title'];
    $blog['subtitulo'] = $blog['subtitle'];
    $blog['contenido'] = $blog['content'];
    $blog['autor'] = $blog['author_name'];
    $blog['fecha_creacion'] = $blog['created_at'];
    $blog['id'] = md5(string: $blog['title'] . $blog['created_at']); //
}
unset($blog); // deshacer

?>
<!DOCTYPE html>
```

Aquí empiezo con el bloque que se encarga de obtener **todos** los blogs que existan en la base.

La idea principal es procesar un poco la información de cada uno antes de mostrarla: contar palabras, estimar el tiempo de lectura y generar un ID único para cada entrada.

Primero, hago el **conteo de palabras**. Para eso, limpio el contenido con `strip_tags()` —porque no quiero contar etiquetas HTML— y luego uso `str_word_count()` para saber cuántas palabras reales hay, cuál luego se guarda como ‘palabra_count’
Con eso listo, calculo el **tiempo de lectura aproximado**. Aquí supuse que una persona lee unas **200 palabras por minuto**, así que simplemente divido el total de palabras entre 200 y redondeo hacia arriba con `ceil()`, lo que he notado es que si el texto es demasiado corto (menos de 200 palabras), el tiempo mínimo será **1 minuto**, para que no aparezca “0 min”.

Por último, genero un **ID único** para cada blog.

Y Cierro el ciclo con `unset($blog)`

En si solo ‘ceil($wordcount / 200)’ es una operación matemática, pero cosas como str_word_count y el hecho de utilizar un ciclo nos forzá a utilizar operaciones lógicas y aritméticas aplicadas (eww)

Script.js:

```
// Sistema de Validación para Registro de cbNoticias
document.addEventListener('DOMContentLoaded', function() {

  const formulario = document.getElementById('registerForm');
  const botonEnviar = document.getElementById('submitBtn');
  const camposTexto = ['nombre', 'correo', 'usuario', 'clave', 'telefono'];

  const patrones = {
    nombre: /^[a-záéíóúñA-ZÁÉÍÓÚÑ\s]+$/,
    correo: /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/,
    usuario: /^[a-zA-Z0-9_]{3,20}$/,
    clave: /^.{6,}$/,
    telefono: /^[0-9]{10}$/
  };

  // Mensajes de ayuda
  const mensajes = {
    nombre: 'Solo letras y espacios',
    correo: 'usuario@cbtis03.edu',
    usuario: '3-20 caracteres (letras, números, guiones bajos)',
    clave: 'Mínimo 6 caracteres',
    telefono: '10 dígitos (opcional)'
  };

  // Estado de validación de campos
  const estadoValido = {
    nombre: true,
    correo: true,
    usuario: false,
    clave: false,
    telefono: true
  };
```

Aquí sí hay **operaciones y cálculos**, pero no en el sentido matemático tradicional.
En este contexto, las **expresiones regulares funcionan como operaciones lógicas** que:

- Evalúan la longitud mínima y máxima de los textos (`{3,20}`, `{6,}`, `{10}`), lo que se puede considerar una forma de cálculo de rango.

- Comparan los caracteres introducidos con un patrón permitido (como una especie de “verificación condicional” que decide si el texto pasa o no la prueba).

Lógica, yay.