

## Actividad 1:

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

The image shows a digital form for reporting errors, titled "\$Titulo\_de\_reporte". The form is set against a light gray grid background. It includes several labels and a table structure.

**Labels:**

- Reporte del día #** (top left)
- Redactado por \$desarrollador** (top right)
- \$=Titulo relevante** (middle right, with a red arrow pointing to the table)
- #=fecha actual** (middle right)
- Nombre desarrollador = \$desarrollador** (middle right)
- \$area de ejecucion = domestico, laboratorio, escuela;** (middle right)
- \$Área de ejecución** (bottom left)

**Table Structure:**

Error de interfaz	Importancia	Páginas relevantes	Explicación humana	Bloques y funciones relevantes
Faltan elementos	Alta			
No avanzan ciertas partes	Media			
Se pierde el usuario	Baja			
Error de Ejecucion	Importancia	Sí	No	
Datos son invalidos	Alta			
No se guarda el grado del alumno	Media			
Tiempo de lectura parece incorrecto	Baja			

## 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 conocidas
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-zAÉÍÓÚÑA-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 X
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     }
81 }
```

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 = 'X ' + 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).

```
97     return esValido;
98 }
99
100 // Configurar escuchadores de eventos
101 camposTexto.forEach(campoId => {
102     const campo = document.getElementById(campoId);
103     if (!campo) return;
104
105     // Al enfocar - mostrar ayuda
106     campo.addEventListener('focus', () => mostrarAyuda(campoId));
107
108     campo.addEventListener('blur', function() {
109         if (this.value.trim() === '' && campoId !== 'telefono') {
110             ocultarValidacion(campoId);
111             estadoValido[campoId] = false;
112             actualizarBotonEnviar();
113         } else {
114             validarCampo(campoId);
115         }
116     });
117
118     campo.addEventListener('input', function() {
119         if (campoId === 'telefono') {
120             this.value = this.value.replace(/\D/g, '').substring(0, 10);
121         }
122         validarCampo(campoId);
123     });
124 });
125
126 formulario.addEventListener('submit', function(e) {
127     let todosValidos = true;
128
129     camposTexto.forEach(campoId => {
130         if (!validarCampo(campoId) && campoId !== 'telefono') {
131             todosValidos = false;
132         }
133     });
134
135     if (!todosValidos) {
136         e.preventDefault();
137         alert('Por favor completa todos los campos correctamente.');
```

## registrar.php

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

```
1  <?php
2  require_once 'lib/common.php';
3  session_start();
4
5  $success = false;
6  $error = '';
7
8  if ($_SERVER['REQUEST_METHOD'] === 'POST') {
9      $pdo = getPDO();
10
11      // Get and sanitize input
12      $nombre = trim(string: $_POST["nombre"] ?? '');
13      $email = trim(string: $_POST["correo"] ?? '');
14      $usuario = trim(string: $_POST["usuario"] ?? '');
15      $clave = $_POST["clave"] ?? '';
16      $telefono = trim(string: $_POST["telefono"] ?? '');
17      $genero_lit_fav = $_POST["genero_lit_fav"] ?? '';
18
19      try {
20          // Validation
21          if (empty($nombre) || empty($email) || empty($usuario) || empty($clave)) {
22              $error = "Todos los campos obligatorios deben ser completados.";
23          }
24          elseif (userExists($pdo, $usuario)) {
25              $error = "El usuario ya existe. Por favor elige otro.";
26          }
27          elseif (emailExists($pdo, $email)) {
28              $error = "El correo ya está registrado.";
29          }
30          else {
31              // Insert new user - NO PASSWORD HASHING (to match login)
32              $stmt = $pdo->prepare(query: "
33                  INSERT INTO user (usuario, nombre, email, clave, fecha_registro, grade, genero_lit_fav)
34                  VALUES (:usuario, :nombre, :email, :clave, CURRENT_TIMESTAMP, 1, :genero_lit_fav)
35              ");
36
37              $result = $stmt->execute(params: [
38                  ':usuario' => $usuario,
39                  ':nombre' => $nombre,
40                  ':email' => $email,
41                  ':clave' => $clave, // Plain text to match your login.php
42                  ':genero_lit_fav' => $genero_lit_fav
43              ]);
44
45              if ($result) {
46                  $success = true;
47                  // Auto-login after registration
48                  login(usuario: $usuario, nombre: $nombre, genero_lit_fav: $genero_lit_fav);
49                  header(header: "refresh:1;url=LP.php");
50              } else {
51                  $error = "Error al crear la cuenta. Intenta nuevamente.";
52              }
53          }
54      } catch (PDOException $e) {
55          error_log(message: "DB Error: " . $e->getMessage());
56          $error = "Error al registrarse: " . $e->getMessage();
57      }
58  }
```

Es tan simple que explicarla puede ser un poco difícil:

```

registrarp.php > ...
1  <?php
2  require_once 'lib/common.php';
3  session_start();
4
5  $success = false;
6  $error = '';
7
8  if ($_SERVER['REQUEST_METHOD'] === 'POST') {
9      $pdo = getPDO();
10

```

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). Además de contar con 2 variables, establecemos que esta base de datos realmente existe (gracias getPDO()) y estamos usando el método POST porque vamos a enviar esta información luego al servidor. Session\_start se asegura de que sea un post.

```

    $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, usuario: $usuario)) {
            $error = "El usuario ya existe. Por favor elige otro.";
        }
        elseif (emailExists($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.

```

try {
    // Validation
    if (empty($nombre) || empty($email) || empty($usuario) || empty($clave)) {
        $error = "Todos los campos obligatorios deben ser completados.";
    }
    elseif (userExists($pdo, $usuario)) {
        $error = "El usuario ya existe. Por favor elige otro.";
    }
    elseif (emailExists($pdo, $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 ahí y termina en el siguiente punto y coma. Te dije que era básica jaja, pero igual funciona como respaldo. Más importante 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?)



```

4
5         if ($result) {
6             $success = true;
7             // Auto-login after registration
8             login(usuario: $usuario, nombre: $nombre, genero_lit_fav: $genero_lit_fav);
9             header(header: "refresh:1;url=LP.php");
10        } else {
11            $error = "Error al crear la cuenta. Intenta nuevamente.";
12        }
13    }
14    catch (PDOException $e) {
15        error_log(message: "DB Error: " . $e->getMessage());
16        $error = "Error al registrarse: " . $e->getMessage();
17    }
18 }
19 ?>

```

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 ahí cabe contar lo que tanto me intriga de PHP:

```

87 <div class="container">
88 <form action="registrar.php" method="post" class="form" id="registerForm">
89 <h2> Crear Cuenta</h2>
90
91 <?php if ($error): ?>
92 <div class="alert alert-error">
93 <?php echo htmlspecialchars($error); ?>
94 </div>
95 <?php endif; ?>
96
97 <?php if ($success): ?>
98 <div class="alert alert-success">
99 <input checked="" type="checkbox"/> ¡Cuenta creada exitosamente! Redirigiendo...
100 </div>
101 <?php endif; ?>
102
103 <div class="input-wrapper">
104 <input type="text" name="nombre" id="nombre" placeholder="Nombre completo"
105 value="<?php echo isset($_POST['nombre']) ? htmlspecialchars($_POST['nombre']) : ''; ?>"
106 required>
107 <div class="validation-box" id="nombreMsg"></div>
108 </div>
109
110 <div class="input-wrapper">
111 <input type="email" name="correo" id="correo" placeholder="Correo electrónico"
112 value="<?php echo isset($_POST['correo']) ? htmlspecialchars($_POST['correo']) : ''; ?>"
113 required>
114 <div class="validation-box" id="correoMsg"></div>
115 </div>
116
117 <div class="input-wrapper">
118 <input type="text" name="usuario" id="usuario" placeholder="Usuario (3-20 caracteres)"
119 value="<?php echo isset($_POST['usuario']) ? htmlspecialchars($_POST['usuario']) : ''; ?>"
120 required>
121 <div class="validation-box" id="usuarioMsg"></div>
122 </div>
123
124 <div class="input-wrapper">
125 <input type="password" name="clave" id="clave" placeholder="Contraseña (mín. 6 caracteres)" required>
126 <div class="validation-box" id="claveMsg"></div>
127 </div>
128
129 <div class="input-wrapper">
130 <input type="tel" name="telefono" id="telefono" placeholder="Teléfono (opcional)"
131 value="<?php echo isset($_POST['telefono']) ? htmlspecialchars($_POST['telefono']) : ''; ?>"
132 <div class="validation-box" id="telefonoMsg"></div>
133 </div>
134
135 <div class="input-wrapper">
136 <select name="genero_lit_fav" id="genero_lit_fav">
137 <option value="">Selecciona tu género literario favorito (opcional)</option>
138 <option value="Ficción"><?php echo (isset($_POST['genero_lit_fav']) && $_POST['genero_lit_fav'] === 'Ficción') ? 'selected' : ''; ?></option>
139 <option value="No Ficción"><?php echo (isset($_POST['genero_lit_fav']) && $_POST['genero_lit_fav'] === 'No Ficción') ? 'selected' : ''; ?></option>
140 <option value="Ciencia Ficción"><?php echo (isset($_POST['genero_lit_fav']) && $_POST['genero_lit_fav'] === 'Ciencia Ficción') ? 'selected' : ''; ?></option>
141 <option value="Romance"><?php echo (isset($_POST['genero_lit_fav']) && $_POST['genero_lit_fav'] === 'Romance') ? 'selected' : ''; ?></option>
142 <option value="Misterio"><?php echo (isset($_POST['genero_lit_fav']) && $_POST['genero_lit_fav'] === 'Misterio') ? 'selected' : ''; ?></option>
143 <option value="Fantasía"><?php echo (isset($_POST['genero_lit_fav']) && $_POST['genero_lit_fav'] === 'Fantasía') ? 'selected' : ''; ?></option>
144 <option value="Horror"><?php echo (isset($_POST['genero_lit_fav']) && $_POST['genero_lit_fav'] === 'Horror') ? 'selected' : ''; ?></option>
145 <option value="Biografía"><?php echo (isset($_POST['genero_lit_fav']) && $_POST['genero_lit_fav'] === 'Biografía') ? 'selected' : ''; ?></option>
146 <option value="Poesía"><?php echo (isset($_POST['genero_lit_fav']) && $_POST['genero_lit_fav'] === 'Poesía') ? 'selected' : ''; ?></option>
147 <option value="General"><?php echo (isset($_POST['genero_lit_fav']) && $_POST['genero_lit_fav'] === 'General') ? 'selected' : ''; ?></option>
148 </select>
149 </div>
150

```

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 código.

## Biblioteca

### Common.php

Este código es fácilmente el cual más me importa como desarrollador;

```
1 <?php
2
3 2 references
4 function getRootPath(): bool|string{
5     return realpath(path: __DIR__ . '/../..');
6 }
7
8 2 references
9 function getDatabasePath(): string{
10     return getRootPath() . '/data/data.sqlite';
11 }
12
13 1 reference
14 function getDsn(): string{
15     return 'sqlite:' . getDatabasePath();
16 }
17
18 8 references
19 function getPdo(): PDO
20 {
21     $pdo = new PDO(dsn: getDsn());
22     $pdo->setAttribute(attribute: PDO::ATTR_ERRMODE, value: PDO::ERRMODE_EXCEPTION);
23
24     // Foreign key constraints need to be enabled manually in SQLite
25     $result = $pdo->query(query: 'PRAGMA foreign_keys = ON');
26     if ($result === false)
27     {
28         throw new Exception(message: 'Could not turn on foreign key constraints');
29     }
30
31     return $pdo;
32 }
33
34 23 references
35 function htmlEscape($html): string
36 {
37     return htmlspecialchars(string: $html, flags: ENT_HTML5, encoding: 'UTF-8');
38 }
39
40 1 reference
41 function TraduceSQLfecha($sqlDate): mixed
42 {
43     if (empty($sqlDate)) {
44         return 'Unknown date';
45     }
46
47     $date = DateTime::createFromFormat(format: 'Y-m-d H:i:s', datetime: $sqlDate);
48
49     if ($date === false) {
50         $date = DateTime::createFromFormat(format: 'Y-m-d', datetime: $sqlDate);
51     }
52
53     if ($date === false) {
54         return $sqlDate;
55     }
56
57     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:

```
lib > common.php > ...
52 }
53
54
55 2 references
56 function intentaLogin(PDO $pdo, $usuario, $clave): mixed
57 {
58     $sql = "
59         SELECT
60         id_usr, usuario, nombre, email, clave, genero_lit_fav
61         FROM
62         user
63         WHERE
64         usuario = :usuario
65     ";
66     $stmt = $pdo->prepare(query: $sql);
67     $stmt->execute(params: ['usuario' => $usuario]);
68
69     $user = $stmt->fetch(mode: PDO::FETCH_ASSOC);
70
71     // Direct password comparison (plain text - matching your system)
72     if ($user && $user['clave'] === $clave) {
73         return $user;
74     }
75
76     return false;
77 }
78
79 3 references
80 function login($usuario, $nombre, $genero_lit_fav = null): void
81 {
82     session_regenerate_id(delete_old_session: true);
83     $_SESSION['usuario'] = $usuario;
84     $_SESSION['nombre'] = $nombre;
85     $_SESSION['genero_lit_fav'] = $genero_lit_fav ?? 'General';
86     $_SESSION['logged_in'] = true;
87 }
88
89 3 references
90 function isLoggedIn(): bool
91 {
92     return isset($_SESSION['logged_in']) && $_SESSION['logged_in'] === true;
93 }
94
95 /**
96  * Require login - redirect if not logged in
97  */
98 2 references
99 function requiereLogin(): void {
100     if (!isLoggedIn()) {
101         header(header: 'Location: login.php');
102         exit();
103     }
104 }
105
106 1 reference
107 function userExists(PDO $pdo, $usuario): bool
108 {
109     $stmt = $pdo->prepare(query: "SELECT COUNT(*) FROM user WHERE usuario = :usuario");
110     $stmt->execute(params: [':usuario' => $usuario]);
111     return $stmt->fetchColumn() > 0;
112 }
113
114 /**
115  * Check if an email exists - FIXED :)
116  */
117 1 reference
118 function emailExists(PDO $pdo, $email): bool
119 {
120     $stmt = $pdo->prepare(query: "SELECT COUNT(*) FROM user WHERE email = :email");
121     $stmt->execute(params: [':email' => $email]);
122     return $stmt->fetchColumn() > 0;
123 }
124 }
```

En lo personal, es la primera vez que trabajo con sesiones así que he tenido diversión descubriendo todos los métodos y variada lógica que podría usar para profundizar mi uso de este simple concepto. Se ve que está lejos de ser perfecto:

```
142 function getCurrentUsername(): mixed
143 {
144     return isset($_SESSION['usuario']) ? $_SESSION['usuario'] : null;
145 }
146
147 /**
148  * Fetch TODOS los usuarios del db
149  */
150 0 references
151 function fetchAllUsuarios(): array {
152     $pdo = getPDO();
153     $stmt = $pdo->prepare(query: '
154         SELECT usuario, nombre, email, genero_lit_fav
155         FROM user
156         ORDER BY usuario ASC
157     ');
158
159     if (!$stmt->execute()) {
160         throw new Exception(message: 'Failed to fetch users from database');
161     }
162
163     return $stmt->fetchAll(mode: PDO::FETCH_ASSOC);
164 }
165
166 1 reference
167 function fetchAllPosts(): array {
168     $pdo = getPDO();
169     $query = $pdo->query(query: '
170         SELECT title, subtitle, author_name, content, created_at, tag
171         FROM post
172         ORDER BY created_at DESC
173     ');
174
175     if ($query === false) {
176         throw new Exception(message: 'Failed to fetch posts from database');
177     }
178
179     return $query->fetchAll(mode: PDO::FETCH_ASSOC);
180 }
181
182 /**
183  * Fetch all comentarios de la db
184  */
185 0 references
186 function fetchAllComments(): array {
187     $pdo = getPDO();
188     $query = $pdo->query(query: '
189         SELECT user_id_C, text, created_at, grade
190         FROM comment
191         ORDER BY created_at DESC
192     ');
193
194     if ($query === false) {
195         throw new Exception(message: 'Failed to fetch comments from database');
196     }
197
198     return $query->fetchAll(mode: PDO::FETCH_ASSOC);
199 }
200
201 };
```

Ha sido una excelente herramienta y lo mejor es que cada línea de código que se ve acá, 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 código) que contenga las operaciones y cálculos.

### Read.php

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

$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 CboNoticias
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-zA-ZáéíóúñA-ZÁÉÍÓÚÑ\s]+$/,
        correo: /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/ ,
        usuario: /^[a-zA-Z0-9_]{3,20}$/ ,
        clave: /^[0-9]{6,}$/ ,
        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.