DeneDig

Desarrollo y Estructura de los Archivos del Proyecto

1. cronometro.php (Checador del Colega)

Componente	Cómo se desarrolló
Backend (PHP)	**Gestión de Sesiones y Hora:** Utiliza session_start() para la autenticación y date_default_timezone_set para asegurar la precisión horaria. **Sincronización:** Contiene una lógica que responde a una petición GET específica devolviendo la hora y fecha del servidor en formato JSON (json_encode), lo cual es clave para la sincronización con el frontend. **Registro:** Procesa la petición POST que contiene la acción (iniciar/detener) y la contraseña, validando la identidad del usuario antes de ejecutar el INSERT o UPDATE en la tabla de registros.
Frontend (JavaScript/UI)	**Reloj en Vivo:** Implementa funciones JavaScript (iniciarRelojTiempoReal, updateLiveTimers) para mostrar la hora actual y el tiempo transcurrido desde el último *check-in*. **Sincronización:** Usa la Fetch API para hacer una llamada periódica (cada 5 minutos) al servidor, garantizando que la hora mostrada sea exacta. **Seguridad:** Despliega un modal de Bootstrap (passwordModal) para solicitar la contraseña del usuario antes de enviar la acción de checado.

A. Sincronización de Hora del Servidor (PHP)

Este bloque permite que el frontend obtenga la hora precisa del servidor sin recargar la página, esencial para el cronómetro.

```php
if (isset($_GET['obtener_hora_servidor'])) {
    // 1. Asegura la zona horaria para precisión.
    date_default_timezone_set('America/Mexico_City');
    $hora_servidor = date('H:i:s');
    $fecha_servidor = date('d/m/Y');
    // 2. Devuelve la hora y fecha en formato JSON para JavaScript.
    echo json_encode(['hora' => $hora_servidor, 'fecha' => $fecha_servidor]);
    exit;
}
```

```javascript
setInterval(function() {
    fetch('?obtener_hora_servidor=1') // Llama al bloque PHP anterior
        .then(response => response.json())
        .then(data => {
            // Actualiza la hora mostrada y reinicia el reloj
            document.getElementById('reloj').textContent = data.hora;
            document.getElementById('fecha').textContent = data.fecha;
            iniciarRelojTiempoReal();
        })
        .catch(error => console.error('Error al sincronizar hora:', error));
}, 300000); // 5 minutos
```

2.justificacion.php (Gestión de Registros Inusuales)

Componente	Cómo se desarrolló
Backend (PHP)	**Consulta y Paginación:** Implementa lógica para la consulta de registros con estatus inusuales. Utiliza variables ($registros_por_pagina, $offset) para gestionar la paginación de los resultados. **Búsqueda (AJAX):** El código PHP detecta peticiones AJAX específicas (isset($_GET['sugerencias'])) y, en lugar de renderizar HTML, ejecuta un *query* para buscar nombres de usuarios y los devuelve en formato JSON (json_encode) para el frontend.
Frontend (JavaScript/UI)	**Búsqueda Asíncrona:** El JavaScript escucha el evento input del campo de búsqueda. Usa la Fetch API para enviar la consulta de búsqueda al *script* PHP de forma asíncrona. **Sugerencias Dinámicas:** La función mostrarSugerencias procesa el JSON recibido y manipula el DOM para crear y desplegar la lista de sugerencias dinámicas (sugerencia-item) justo debajo del campo de búsqueda.

A. Detección y Respuesta AJAX para Sugerencias (PHP)

Este fragmento detecta si la petición es una llamada AJAX para devolver datos JSON en lugar de HTML.

```php
// Deteccion si es peticion AJAX y si se piden sugerencias
$is_ajax = !empty($_SERVER['HTTP_X_REQUESTED_WITH']) && strtolower($_SERVER['HTTP_X

if ($is_ajax && isset($_GET['sugerencias'])) {
    $busqueda_sugerencias = isset($_GET['buscar']) ? $_GET['buscar'] : '';
    $param_busqueda_sugerencias = "%{$busqueda_sugerencias}%";

    // ... Ejecuta query para buscar nombres ...

    // Devuelve el array de sugerencias en JSON
    echo json_encode($sugerencias_array);
    exit;
}
```

B. Petición Asíncrona de Sugerencias (JavaScript)

Esta función envía la búsqueda al servidor y espera la respuesta JSON.

```javascript
function buscarSugerencias(query) {
    fetch(`?sugerencias=1&buscar=${encodeURIComponent(query)}`, {
        headers: {
            'X-Requested-With': 'XMLHttpRequest' // Indica al servidor que es
        }
    })
    .then(response => response.json())
    .then(sugerencias => {
        mostrarSugerencias(sugerencias); // Funcion que actualiza el DOM
    })
    .catch(error => console.error('Error fetching suggestions:', error));
}
```

3. historial_colega.php (Consulta y Edición Detallada)

Componente	Cómo se desarrolló
Backend (PHP)	**Seguridad y Acceso:** Requiere el id_usuario a través de la URL ($_GET) y lo valida. Edición: Maneja una petición POST de edición que recibe el id_registro, el nuevo estatus y las nuevas horas de entrada/salida. Utiliza mysqli_real_escape_string para sanitizar todos los datos de entrada antes de ejecutar el *query* de UPDATE en la base de datos.
Frontend (JavaScript/UI)	**Modal de Edición:** Utiliza JavaScript para capturar los datos de la fila de registro seleccionada y pasarlos al modal de Bootstrap (editModal). **Validación de Cambios**: Un *listener* de JavaScript (validarCambios) compara las horas originales con las nuevas horas de entrada/salida para habilitar o deshabilitar el botón de guardar, previniendo envíos innecesarios.

A. Lógica de Edición y Sanitización (PHP)

Bloque que procesa el formulario de edición, demostrando sanitización (mysqli_real_escape_string) antes del UPDATE.

```php
// Código para manejar la edición de registros (si es necesario)
if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['accion_edicion']) && $_POST['accion_edicion'] === 'guardar') {
    $id_registro = mysqli_real_escape_string(mysql: $conexion, string: $_POST['id_registro_editar']);
    $nuevo_estatus = mysqli_real_escape_string(mysql: $conexion, string: $_POST['nuevo_estatus']);
    $nueva_entrada = mysqli_real_escape_string(mysql: $conexion, string: $_POST['nueva_entrada']);
    $nueva_salida = mysqli_real_escape_string(mysql: $conexion, string: $_POST['nueva_salida']);
    $nueva_descripcion = mysqli_real_escape_string(mysql: $conexion, string: $_POST['nueva_descripcion']);

    $query_update = "UPDATE registros SET hora_entrada = '$nueva_entrada', hora_salida = '$nueva_salida',
                    estatus = '$nuevo_estatus', descripcion = '$nueva_descripcion' WHERE id_registro = '$id_registro'";

    // mysqli_query($conexion, $query_update); // Ejecución de la actualizacion
}
```

B. Validación de Cambios (JavaScript)

Función que previene el envío del formulario si no hay cambios reales en los campos de hora.

```javascript
const validarCambios = () => {
    const nuevaEntrada = document.getElementById('nueva_entrada').value;
    const nuevaSalida = document.getElementById('nueva_salida').value;
    const btnGuardar = document.getElementById('btn-guardar');

    // Compara el valor actual con el valor original guardado al abrir el modal
    if (nuevaEntrada !== originalEntrada || nuevaSalida !== originalSalida) {
        btnGuardar.disabled = false; // Habilita el botón
    } else {
        btnGuardar.disabled = true; // Deshabilita el botón
    }
};
// Escucha eventos para entrada y salida
document.getElementById('nueva_entrada').addEventListener('input', validarCambios);
document.getElementById('nueva_salida').addEventListener('input', validarCambios);
```

4. horas_asignadas.php (Administración de Horarios)

Componente	Cómo se desarrolló
Backend (PHP)	**CRUD de Horarios:** La lógica principal es un bloque POST que distingue entre la acción de crear un nuevo horario y la acción de editar/actualizar un horario existente. Convierte el *array* de días seleccionados ($_POST['nombre_dias']) en una cadena de texto para su almacenamiento en un solo campo de la base de datos (implode).
Frontend (JavaScript/UI)	**Cálculo de Horas:** El JavaScript (calcularHorasModal) implementa una lógica de cálculo para determinar las horas totales de un horario basándose en la diferencia entre la hora de inicio y término, y los días seleccionados, mostrando el resultado al usuario antes de guardar. **Pre-llenado de Modal:** Utiliza jQuery para leer los atributos de datos (data-*) de los botones de edición y cargar todos los campos del modal de edición (incluyendo los *checkboxes* de los días de la semana) con la información existente del registro.

A. Manejo de Días Seleccionados (PHP)

El uso de implode es clave para convertir un *array* de selección múltiple (días) en una cadena que puede almacenarse en un solo campo de la base de datos.

```php
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    if (isset($_POST['editar_id'])) {
        // ... Sanitización de otros campos ...

        // Convierte el array de días seleccionados a una cadena para guardar
        $dias_seleccionados = isset($_POST['nombre_dias']) ? $_POST['nombre_dias'] : [];
        $nombre_dias = implode(separator: ", ", array: $dias_seleccionados);

        // ... Ejecucion de query UPDATE usando $nombre_dias ...
    }
    // ...
}
```

B. Lectura de Datos del Modal de Edición (JavaScript/jQuery)

Este fragmento ilustra cómo se utiliza el atributo data-* de HTML para transferir datos de PHP a JavaScript y pre-llenar un modal de edición.

```javascript
$(document).on('click', '.btn-edit', function(e) {
    e.stopPropagation();
    var modalData = $(this).data(); // Captura todos los atributos data-* del boton

    $('#editarHorarioModal').modal('show');
    $('#editar_id').val(modalData.id);
    $('#editar_hora_inicio').val(modalData.horainicio);

    // Lógica para marcar los checkboxes de los días existentes
    if (modalData.dias) {
        var diasArray = modalData.dias.split(', ');
        diasArray.forEach(function(dia) {
            $('#editar_nombre_dia_' + dia.trim()).prop('checked', true);
        });
    }
    // ...
});
```

5.asignacion_horarios.php (Asignación de Horarios)

Componente	Cómo se desarrolló
Backend (PHP)	**Consultas Relacionales:** Ejecuta *queries* complejos con LEFT JOIN para obtener la lista de horarios agrupados por servicio y universidad, lo cual es esencial para la lógica de asignación. **Asignación:** La lógica POST principal se enfoca en actualizar el campo id_horario en la tabla de usuarios o departamentos basándose en la selección del líder.
Frontend (JavaScript/UI)	**Navegación:** Usa JavaScript para gestionar la interacción y navegación dentro del módulo (ej., volverADepartamentos). **Notificaciones:** Implementa un bloque condicional en PHP que, si existe un mensaje de notificación ($mensaje_modal), utiliza JavaScript y jQuery para inyectar y mostrar un modal de Bootstrap dinámico (mensajeModal) al cargar la página.

A. Consulta de Horarios Agrupados (PHP)

Este *query* demuestra la necesidad de relacionar la tabla horario con otras entidades (universidades, servicios) para una presentación organizada al líder.

```php
$query_horarios = "SELECT h.id_horario, h.nombre_dias, h.hora_inicio, h.hora_termino,
                          u.siglas, ts.nombre_servicio, ts.id_servicio, u.id_escuela, h.horas_totales
                   FROM horario h
                   LEFT JOIN universidades u ON h.id_escuela = u.id_escuela
                   LEFT JOIN tipo_servicio ts ON h.id_servicio = ts.id_servicio
                   ORDER BY ts.nombre_servicio, u.siglas, h.nombre_dias ASC";

// El resultado es procesado en PHP para agrupar en $horarios_agrupados
```

B. Generación y Visualización de Modal de Notificación (PHP y JavaScript)

Este bloque de PHP incrusta código JavaScript para mostrar un modal de éxito o error solo si la variable $mensaje_modal está definida.

```php
<?php if (isset($mensaje_modal) && $mensaje_modal != ""): ?>
    var modalHTML = `
        <div class="modal fade" id="mensajeModal" tabindex="-1" aria-hidden="true">
            <div class="modal-body"><?= addslashes($mensaje_modal) ?></div>
        </div>
    `;

    $('body').append(modalHTML);
    var mensajeModal = new bootstrap.Modal(document.getElementById('mensajeModal'));
    mensajeModal.show(); // Muestra el modal

    $('#mensajeModal').on('hidden.bs.modal', function() {
        $(this).remove(); // Elimina el modal del DOM al cerrarse
    });
<?php endif; ?>
```