Repositorio de Libros Virtual: El Rincón de ADSO

Gerardo Andres Ardila Jerez

1097332311

Ficha: 2931558

Servicio Nacional del Aprendizaje SENA

Análisis y Desarrollo de Software ADSO

Vélez, Santander

01 de mayo de 2025

Este proyecto tiene como finalidad la creación de una plataforma web enfocada en apoyar a los aprendices del SENA que se encuentran en proceso de formación en el área de programación. La propuesta busca proporcionar un espacio digital donde los usuarios puedan acceder fácilmente a material de estudio, compartir conocimientos y fortalecer su aprendizaje de manera autónoma, todo mediante una interfaz sencilla y funcional.

La idea nace de la necesidad de centralizar recursos que, aunque disponibles en línea, muchas veces se encuentran dispersos o poco organizados. Por eso, esta herramienta tiene como propósito facilitar el acceso rápido a contenidos relevantes, como libros, manuales, guías y documentos técnicos relacionados con los lenguajes de programación, bases de datos, estructuras de software y metodologías de desarrollo.

Para llevar a cabo este desarrollo, se hizo uso de herramientas y tecnologías actuales, entre ellas el lenguaje PHP, la base de datos PostgreSQL, el entorno de desarrollo Visual Studio Code y el sistema de control de versiones GitHub. El producto final es un repositorio en línea que reúne libros y recursos relevantes para quienes se están iniciando en la programación, organizados de forma que permitan una navegación fluida y un aprendizaje práctico y accesible.

11 Abril del 2025

Creación de base de datos inicia

En colaboración con mi compañero Jhonny, se nos asignó la tarea de diseñar y estructurar la base de datos principal del sistema. El objetivo fue establecer una base sólida para el manejo eficiente de los recursos educativos que se integran en la plataforma.

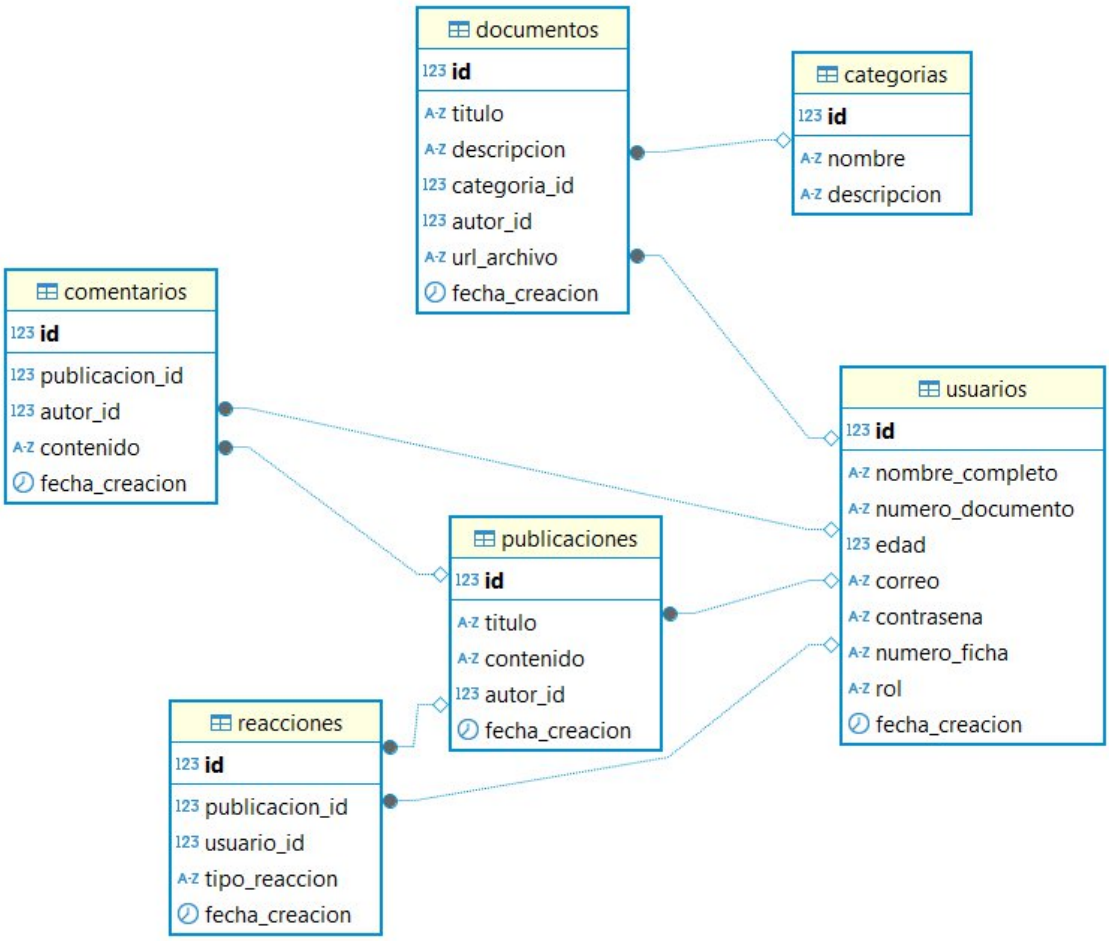
Para ello, realizamos un análisis previo de los elementos fundamentales que requería el sistema, como los documentos, usuarios, categorías, publicaciones, comentarios y reacciones. Con base en este análisis, procedimos a modelar la base de datos utilizando PostgreSQL, enfocándonos en una estructura relacional clara y normalizada que garantizara la integridad de los datos, la escalabilidad del sistema y un acceso rápido a la información.

La estructura final incluye varias tablas interrelacionadas, cada una con campos específicos y claves foráneas que permiten mantener la coherencia de los datos. Este diseño facilita la organización de los contenidos, el control de usuarios y la interacción entre los distintos componentes del sistema.

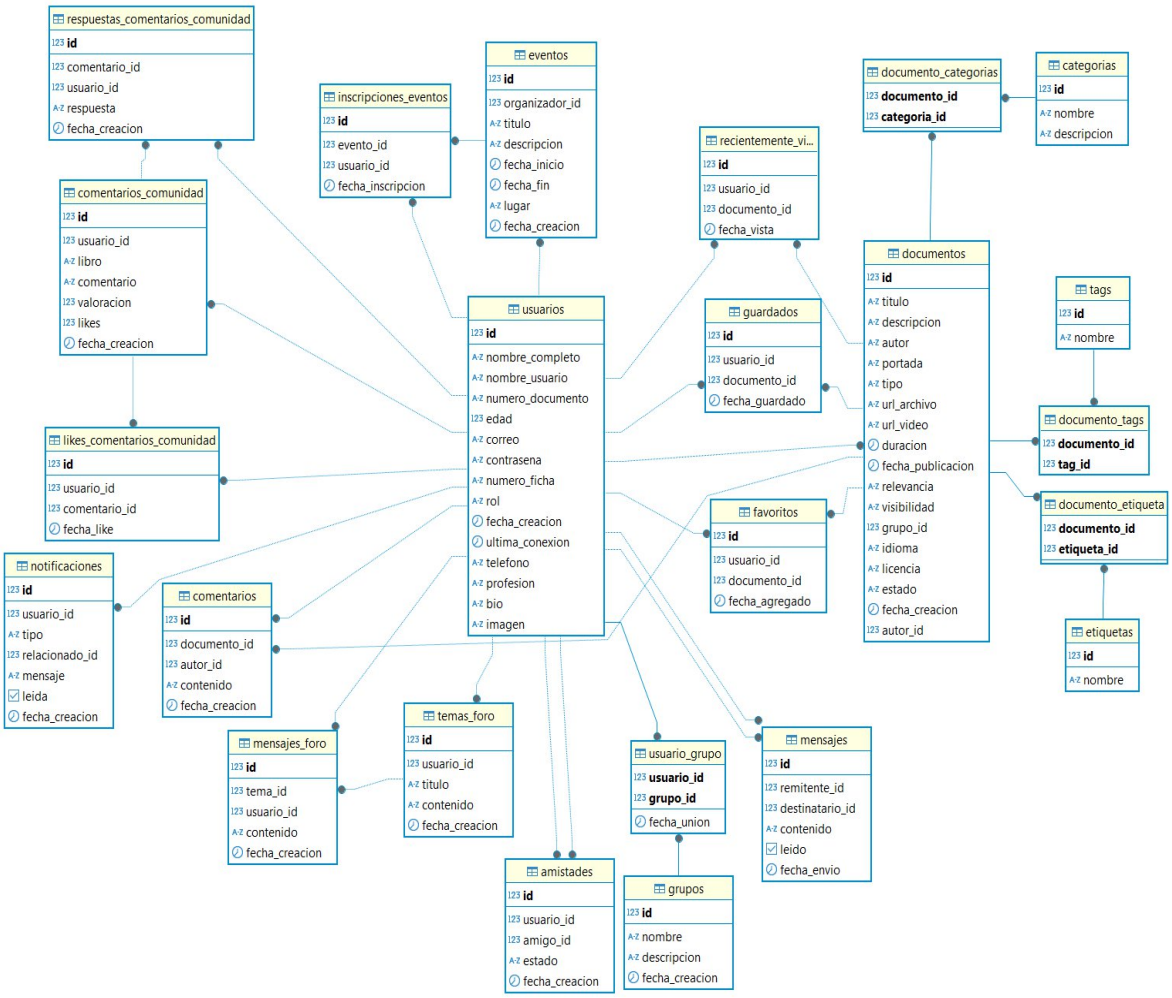
Esta contaba con:

- **documentos:** Almacena información de documentos y artículos con campos como id, título, descripción, categoría_id, autor_id, url_archivo y fecha_creacion.
- **categorías:** Registre las categorías de los recursos con campos id, nombre y descripción.
- **publicaciones:** Contiene datos de publicaciones con id, título, contenido, autor_id y fecha_creación.
- **usuarios:** Gestiona la información de los usuarios con campos id, nombre_completo, numero_documento, edad, correo, contraseña, numero_ficha, rol y fecha_creacion.
- **reacciones:** Registra las interacciones de los usuarios con las publicaciones mediante los campos id, publicacion_id, usuario_id, tipo_reaccion y fecha_creacion.
- **comentarios:** Almacena los comentarios realizados en publicaciones con id, publicacion_id, autor_id, contenido y fecha_creacion.

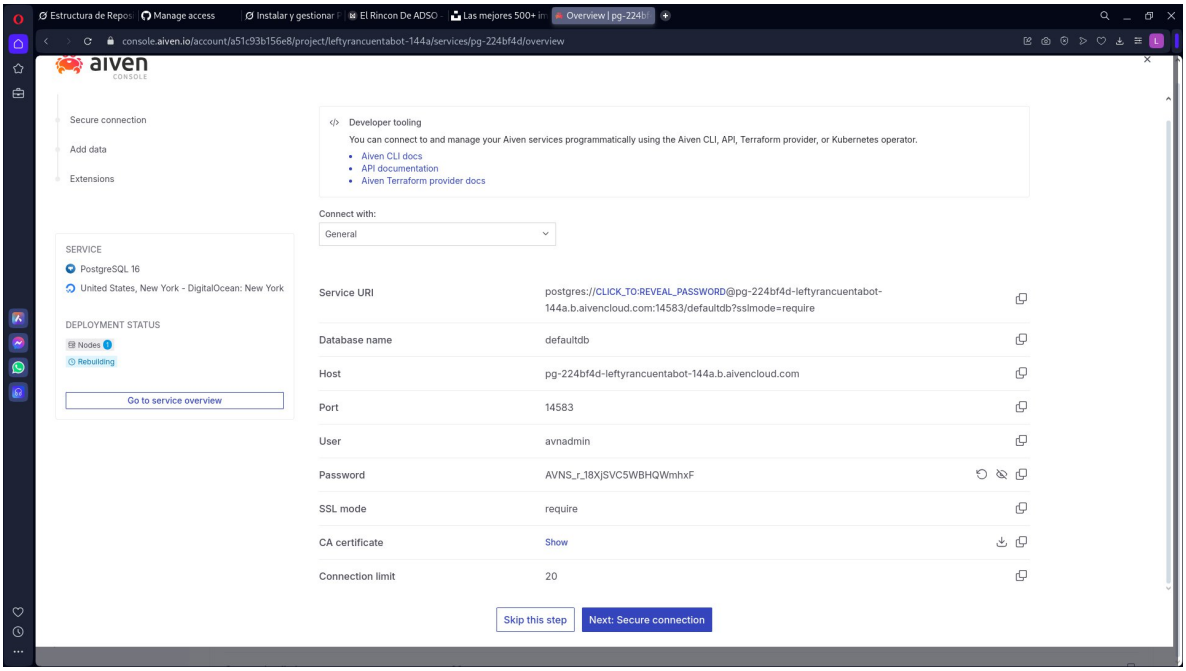
version 1:



Con el paso del tiempo y conforme avanzábamos en el desarrollo, la base de datos fue ampliándose y ajustándose hasta quedar como se muestra en la imagen actual.

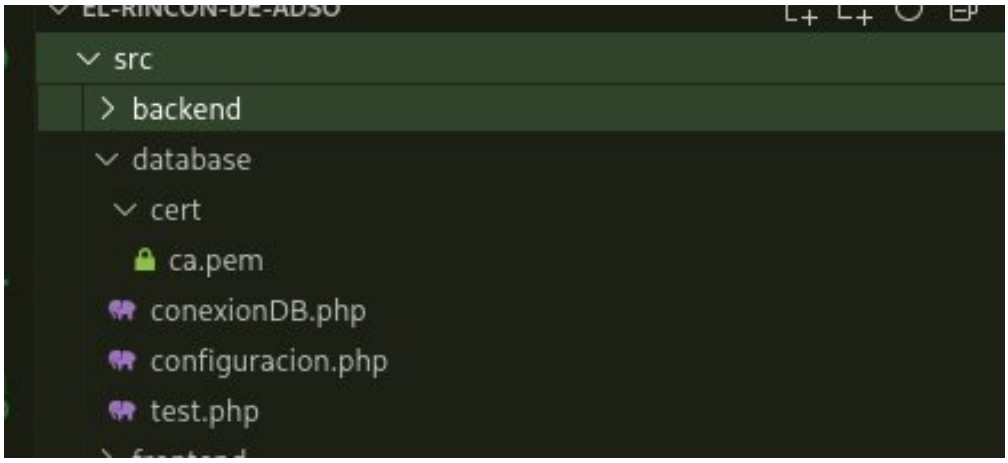


Después de finalizar esa tarea, ese mismo día se me asignó la labor de backend, comenzando con la configuración de la conexión en PHP hacia la base de datos PostgreSQL, la cual estaba alojada temporalmente en el servicio en la nube Aiven, como se observa en la imagen.



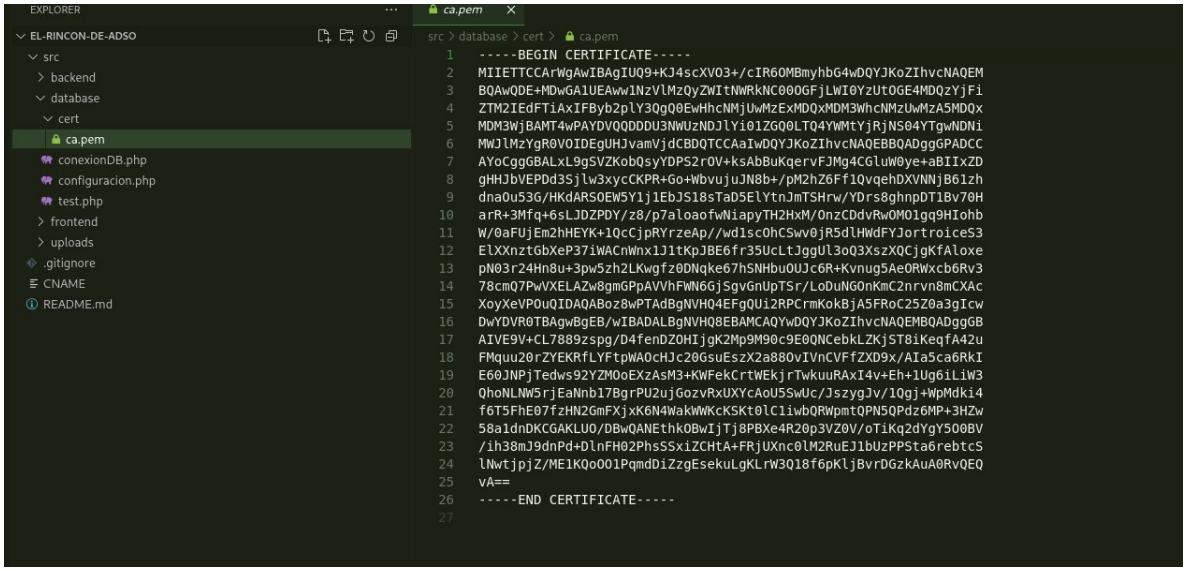
Pasos realizados:

1. **Creación de la Estructura de Archivos**:
    - Se creó una carpeta denominada `database`, que contiene los archivos necesarios para gestionar la conexión:
        - `conexion.php`: Archivo principal para establecer la conexión a la base de datos.
        - `configuracion.php`: Archivo que contiene las credenciales de la base de datos y la configuración necesaria.
        - `test.php`: Archivo de prueba para verificar que la conexión es exitosa y que la base de datos está accesible.



2.**Configuración de la Conexión**:

- En `configuracion.php`, se definieron las constantes con las credenciales necesarias para la conexión: host, puerto, nombre de la base de datos, usuario y contraseña.
- Se especificó la ruta al archivo `ca.pem`, que es el certificado SSL requerido por Aiven para asegurar la conexión.



3.**Implementación de la Conexión en** `conexion.php`:

- La clase `conexionDB` se implementó utilizando el patrón Singleton, lo que garantiza que solo se cree una instancia de la conexión a la base de datos. Esta clase utiliza PDO para la conexión y maneja las excepciones de forma adecuada.

conexionDB.php

src > database > conexionDB.php > ...

```php
<?php
require_once "configuracion.php";

class conexionDB {
    private static $conexion = null;

    public static function getConexion() {
        if (self::$conexion === null) {
            try {
                $dsn = "pgsql:host=" . DB_HOST .
                       ";port=" . DB_PORT .
                       ";dbname=" . DB_NAME .
                       ";sslmode=" . DB_SSLMODE .
                       ";sslrootcert=" . DB_SSLROOTCERT;

                self::$conexion = new PDO($dsn, DB_USER, DB_PASSWORD);
                self::$conexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
            } catch (PDOException $e) {
                die("❌ Error de conexión: " . $e->getMessage());
            }
        }
        return self::$conexion;
    }
}
?>
```

4.**Archivo de Prueba (**`test.php`**):**

- Se creó un archivo de prueba para verificar si la conexión a la base de datos se realizó con éxito. En este archivo se realizó una consulta para obtener la fecha actual y el usuario conectado en la base de datos, confirmando así que la conexión funciona correctamente.

test.php

src > database > test.php > ...

```php


<?php
require_once "conexionDB.php";

try {
    $conn = conexionDB::getConexion();
    $stmt = $conn->query("SELECT current_date, current_user");
    $data = $stmt->fetch(PDO::FETCH_ASSOC);

    echo "✅ Conexión exitosa.<br>";
    echo "📅 Fecha actual: " . $data['current_date'] . "<br>";
    echo "👤 Usuario actual: " . $data['current_user'];
} catch (Exception $e) {
    echo "❌ Fallo en la conexión: " . $e->getMessage();
}

?>

```

Una vez completada la configuración de la conexión a la base de datos, esperé a que el equipo de frontend terminara su parte del proyecto.

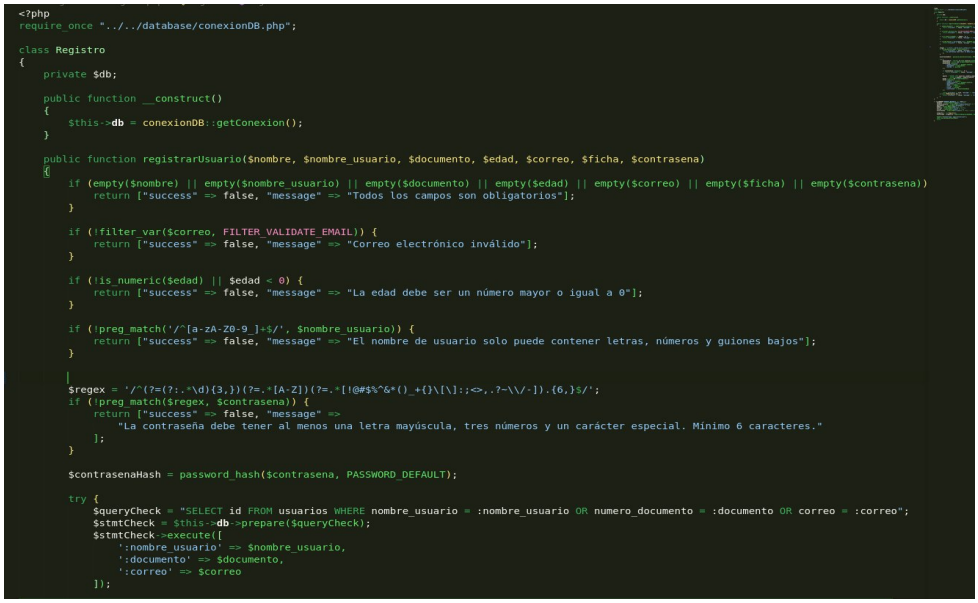
14 Abril del 2025

Logica en el repositorio

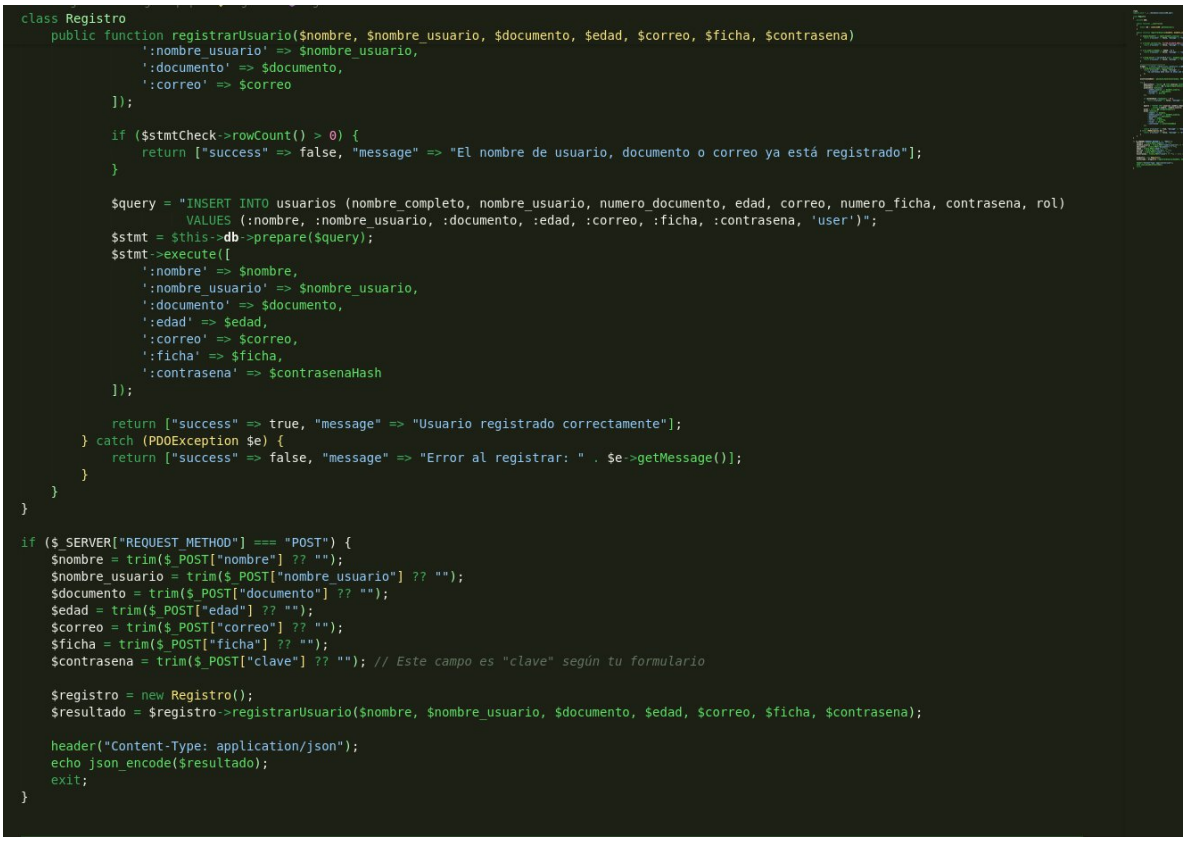
Después de esperar a que el equipo de frontend completara sus respectivas tareas, pude iniciar la implementación de la funcionalidad de registro de usuarios en el backend.
Los objetivos de esta fase eran los siguientes:

- Validar los datos recibidos desde el formulario.
- Asegurar que el nombre de usuario, correo y documento no estuvieran registrados previamente.
- Validar la seguridad de la contraseña.
- Registrar el nuevo usuario en la base de datos con un rol por defecto (`user`)

Dentro del repositorio, en la carpeta `backend`, creé una nueva carpeta llamada `register`, y en ella añadí el archivo `register.php`, que contiene toda la lógica para procesar el registro de un nuevo usuario.



```php
<?php
require_once "../../database/conexionDB.php";

class Registro
{
    private $db;

    public function __construct()
    {
        $this->db = conexionDB::getConexion();
    }

    public function registrarUsuario($nombre, $nombre_usuario, $documento, $edad, $correo, $ficha, $contrasena)
    {
        if (empty($nombre) || empty($nombre_usuario) || empty($documento) || empty($edad) || empty($correo) || empty($ficha) || empty($contrasena))
            return ["success" => false, "message" => "Todos los campos son obligatorios"];
        }

        if (!filter_var($correo, FILTER_VALIDATE_EMAIL)) {
            return ["success" => false, "message" => "Correo electrónico inválido"];
        }

        if (!is_numeric($edad) || $edad < 0) {
            return ["success" => false, "message" => "La edad debe ser un número mayor o igual a 0"];
        }

        if (!preg_match('/^[a-zA-Z0-9_]+$/', $nombre_usuario)) {
            return ["success" => false, "message" => "El nombre de usuario solo puede contener letras, números y guiones bajos"];
        }

        $regex = '/^(?=(?:.*\d){3,})(?=.*[A-Z])(?=.*[!@#$%^&*()_+{}\[\]:;<>,.?~\\/-]).{6,}$/';
        if (!preg_match($regex, $contrasena)) {
            return ["success" => false, "message" =>
                "La contraseña debe tener al menos una letra mayúscula, tres números y un carácter especial. Mínimo 6 caracteres."
            ];
        }

        $contrasenaHash = password_hash($contrasena, PASSWORD_DEFAULT);

        try {
            $queryCheck = "SELECT id FROM usuarios WHERE nombre_usuario = :nombre_usuario OR numero_documento = :documento OR correo = :correo";
            $stmtCheck = $this->db->prepare($queryCheck);
            $stmtCheck->execute([
                ':nombre_usuario' => $nombre_usuario,
                ':documento' => $documento,
                ':correo' => $correo
            ]);
```



```php
class Registro
    public function registrarUsuario($nombre, $nombre_usuario, $documento, $edad, $correo, $ficha, $contrasena)
                ':nombre_usuario' => $nombre_usuario,
                ':documento' => $documento,
                ':correo' => $correo
            ]);

            if ($stmtCheck->rowCount() > 0) {
                return ["success" => false, "message" => "El nombre de usuario, documento o correo ya está registrado"];
            }

            $query = "INSERT INTO usuarios (nombre_completo, nombre_usuario, numero_documento, edad, correo, numero_ficha, contrasena, rol)
                      VALUES (:nombre, :nombre_usuario, :documento, :edad, :correo, :ficha, :contrasena, 'user')";
            $stmt = $this->db->prepare($query);
            $stmt->execute([
                ':nombre' => $nombre,
                ':nombre_usuario' => $nombre_usuario,
                ':documento' => $documento,
                ':edad' => $edad,
                ':correo' => $correo,
                ':ficha' => $ficha,
                ':contrasena' => $contrasenaHash
            ]);

            return ["success" => true, "message" => "Usuario registrado correctamente"];
        } catch (PDOException $e) {
            return ["success" => false, "message" => "Error al registrar: " . $e->getMessage()];
        }
    }
}

if ($_SERVER["REQUEST_METHOD"] === "POST") {
    $nombre = trim($_POST["nombre"] ?? "");
    $nombre_usuario = trim($_POST["nombre_usuario"] ?? "");
    $documento = trim($_POST["documento"] ?? "");
    $edad = trim($_POST["edad"] ?? "");
    $correo = trim($_POST["correo"] ?? "");
    $ficha = trim($_POST["ficha"] ?? "");
    $contrasena = trim($_POST["clave"] ?? ""); // Este campo es "clave" según tu formulario

    $registro = new Registro();
    $resultado = $registro->registrarUsuario($nombre, $nombre_usuario, $documento, $edad, $correo, $ficha, $contrasena);

    header("Content-Type: application/json");
    echo json_encode($resultado);
    exit;
}
```

Con esta implementación, el backend ya contaba con una ruta funcional para registrar nuevos usuarios, validando datos correctamente y protegiendo las contraseñas mediante `password_hash`.

Ya con esta parte del backend funcionando, lo siguiente era esperar nuevamente a que el equipo de frontend finalizara su parte, para así poder continuar implementando más lógica y funcionalidades en el sistema.

22 de abril de 2025
Implementación de funcionalidades: Chat, Amigos, Notificaciones y Ver Perfil

Ese día se me asignó desarrollar cuatro nuevas vistas dentro del panel de usuario ya existente:

- `amigos.php`
- `chat.php`
- `notificaciones.php`
- `ver_perfil.php`

La tarea incluía tanto el frontend (diseño e interfaz) como el backend (lógica y conexión con la base de datos) para cada una de estas secciones, lo cual representó un trabajo de tipo fullstack.
El objetivo principal fue permitir a los usuarios ver su lista de amigos, enviar y recibir mensajes en tiempo real, visualizar notificaciones del sistema y consultar el perfil de otros usuarios registrados.

Amigos:

Estructura:

Ubicación y Organización:

- **Frontend**: Vista ubicada en /friends/amigos.php.
- **Backend**: Conexión a la base de datos mediante ../../database/conexionDB.php.
- **APIs**: Carpeta /api creada en el backend, pero sin APIs implementadas para esta funcionalidad.

Dependencias:

- **PHP**: Manejo de sesiones, consultas con PDO, procesamiento de formularios.
- **PostgreSQL**: Base de datos relacional para almacenar usuarios, amistades y notificaciones.
- **HTML/CSS**: Interfaz con estilos en ./css/style.css.

- **JavaScript**: Script para menú móvil responsivo.
- **Font Awesome**: Iconos (v6.4.0, cargados desde CDN).
- **Avatares**: Imágenes personalizadas en ../../backend/perfil/ o predeterminadas de pravatar.cc.

Funciones:

1. **Autenticación y Seguridad**:
    - Verificación de sesión con session_start() y $_SESSION['usuario_id'].
    - Redirección a index.php si no hay sesión o el usuario no existe.
    - Prevención de XSS con htmlspecialchars().
    - Consultas preparadas con PDO para evitar inyecciones SQL.

2. **Búsqueda de Usuarios**:
    - Formulario POST para buscar usuarios por nombre_usuario usando ILIKE (insensible a mayúsculas/minúsculas, nativo de PostgreSQL).
    - Excluye al usuario actual, amigos existentes, y solicitudes pendientes/aceptadas.
    - Muestra resultados con nombre, correo, avatar, y botón para enviar solicitud.

3. **Gestión de Solicitudes de Amistad**:
    - **Enviar Solicitud**: Inserta en amistades con estado pending y genera notificación.
    - **Aceptar Solicitud**: Actualiza a accepted y notifica al remitente.
    - **Rechazar Solicitud**: Cambia a rejected.
    - Redirección con header("Location: amigos.php") para evitar reenvíos.

4. **Lista de Amigos**:
    - Muestra amigos con estado accepted, incluyendo nombre, correo, avatar, y última conexión.
    - Función getOnlineStatus() calcula si un amigo está en línea (última conexión ≤ 5 minutos).
    - Opciones para chatear, ver perfil, o eliminar amigo.

5. **Notificaciones**:

- Contador de notificaciones no leídas en la barra de navegación.
- Generación automática de notificaciones para solicitudes y aceptaciones.

6. **Interfaz de Usuario**:
    - Navegación responsiva con menú móvil (JavaScript).
    - Secciones para búsqueda, solicitudes, y lista de amigos.
    - Tarjetas (friend-card, request-card) con estados visuales (en línea/fuera de línea).

```
<body>
  <section class="friends-section">
    <div class="container">
      <div class="requests-list">
      </div>

      <!-- Lista de amigos -->
      <div class="friends-list">
        <h2 class="friends-list__title">Mis Amigos</h2>
        <?php if (empty($friends)): ?>
          <p class="friends-list__empty">Aún no tienes amigos. ¡Busca usuarios y envía solicitudes!</p>
        <?php else: ?>
          <?php foreach ($friends as $friend): ?>
            <?php $status = getOnlineStatus($friend['ultima_conexion']); ?>
            <div class="friend-card">
              <div class="friend-card__avatar">
                <img src="<?php echo $friend['imagen'] ? '../../backend/perfil/' . htmlspecialchars($friend['imagen']
              </div>
              <div class="friend-card__info">
                <h3 class="friend-card__name"><?php echo htmlspecialchars($friend['nombre_usuario']); ?></h3>
                <p class="friend-card__email"><?php echo htmlspecialchars($friend['correo']); ?></p>
                <p class="friend-card__status <?php echo $status['is_online'] ? 'friend-card__status--online' : 'frie
                  <?php echo $status['status_text']; ?>
                </p>
              </div>
              <div class="friend-card__actions">
                <a href="../mensajes/mensajes.php?friend_id=<?php echo $friend['id']; ?>" class="friend-card__action
                  Chatear
                </a>
                <a href="../perfil/perfil.php?user_id=<?php echo $friend['id']; ?>" class="friend-card__action friend
                  Ver perfil
                </a>
                <form method="POST">
                  <input type="hidden" name="friend_id" value="<?php echo $friend['id']; ?>">
                  <button type="submit" name="remove_friend" class="friend-card__action friend-card__action--remove">
                </form>
              </div>
            </div>
          <?php endforeach; ?>
        <?php endif; ?>
      </div>
    </div>
  </section>

  <!-- Script para el menú móvil -->
  <script>
    document.getElementById('mobile-menu-button').addEventListener('click', function() {
```

```
    header("Location: amigos.php");
    exit();
  }

  // Aceptar solicitud de amistad
  if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['accept_request'])) {
    $sender_id = $_POST['sender_id'];

    // Actualizar el estado a 'accepted'
    $query = "UPDATE amistades
              SET estado = 'accepted', fecha_creacion = CURRENT_TIMESTAMP
              WHERE usuario_id = :sender_id AND amigo_id = :receiver_id AND estado = 'pending'";
    $stmt = $db->prepare($query);
    $stmt->execute([
      ':sender_id' => $sender_id,
```

Mensajes:

Estructura:

Ubicación y Organización:

- **Frontend**: Vista en /mensajes/mensajes.php.
- **Backend**: Conexión a la base de datos en ../../database/conexionDB.php.
- **API**: Endpoint en ../../backend/api/mensajes.php.
- **Estilos**: CSS en /mensajes/css/style.css y /friends/css/style.css, con estilos inline para acciones de mensajes.

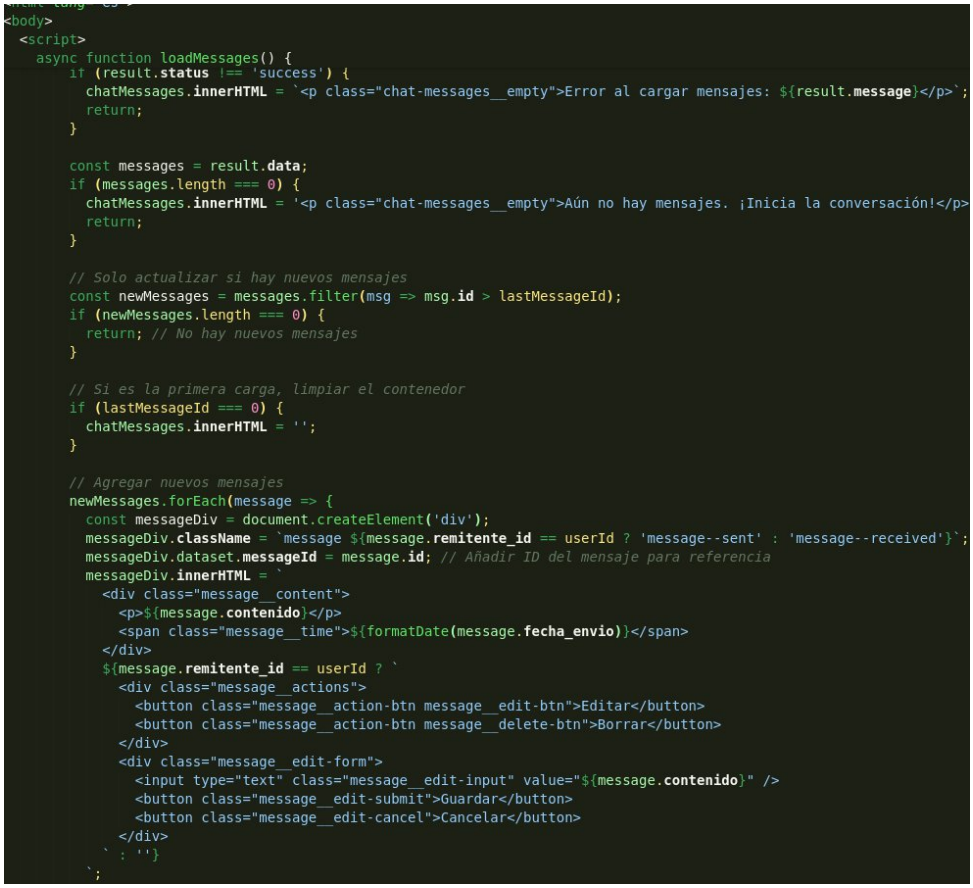
2. **Dependencias**:
    - **PHP**: Manejo de sesiones, consultas con PDO, procesamiento de solicitudes API.
    - **PostgreSQL**: Base de datos relacional para almacenar usuarios, amistades, mensajes y notificaciones.
    - **HTML/CSS**: Interfaz responsiva con tarjetas de mensajes y formulario de entrada.
    - **JavaScript**: Gestión dinámica del chat con fetch, polling cada 5 segundos, y eventos para editar/borrar.
    - **Font Awesome**: Iconos (v6.4.0, CDN).

- **Avatares**: Imágenes en ../../backend/perfil/ o predeterminadas de pravatar.cc.

Funciones:

1. **Autenticación y Seguridad** (mensajes.php y api/mensajes.php):
    - Verificación de sesión con session_start() y $_SESSION['usuario_id'].
    - Redirección a index.php si no hay sesión o el usuario no existe.
    - Validación de que el friend_id corresponde a un amigo con estado accepted en amistades.
    - Prevención de XSS con htmlspecialchars() en datos de usuario.
    - Consultas preparadas con PDO para evitar inyecciones SQL.
    - Verificación de permisos en la API para editar/borrar mensajes (solo el remitente puede modificar).
2. **Interfaz de Chat (mensajes.php)**:
    - Muestra el perfil del amigo (avatar, nombre, correo) en la cabecera.
    - Área de mensajes dinámica que carga mensajes vía API y se actualiza cada 5 segundos.
    - Formulario para enviar mensajes, con soporte para tecla Enter.
    - Opciones para editar/borrar mensajes propios, con formularios dinámicos que aparecen al hacer hover.
    - Navegación responsiva con menú móvil.
3. **Gestión de Mensajes (api/mensajes.php)**:
    - **GET**: Recupera mensajes entre el usuario y un amigo, ordenados por fecha_envio.
    - **POST**: Envía un nuevo mensaje y genera una notificación para el destinatario.
    - **PUT**: Edita el contenido de un mensaje existente, actualizando fecha_envio.
    - **DELETE**: Borra un mensaje, validando que pertenezca al remitente.
    - Respuestas en formato JSON con status (success/error) y mensajes descriptivos.

4. **Notificaciones**:
    - Contador de notificaciones no leídas en la barra de navegación.
    - Generación automática de notificaciones para nuevos mensajes en la API (tipo: new_message).
5. **Interfaz Dinámica**:
    - Polling cada 5 segundos para cargar nuevos mensajes sin recargar la página.
    - Desplazamiento automático al final del chat tras cargar mensajes.
    - Estilos diferenciados para mensajes enviados (message--sent) y recibidos (message--received).
    - Acciones de editar/borrar visibles al hacer hover sobre mensajes propios.



```
<body>
  <script>
    async function loadMessages() {
        if (result.status !== 'success') {
          chatMessages.innerHTML = `<p class="chat-messages__empty">Error al cargar mensajes: ${result.message}</p>`;
          return;
        }

        const messages = result.data;
        if (messages.length === 0) {
          chatMessages.innerHTML = '<p class="chat-messages__empty">Aún no hay mensajes. ¡Inicia la conversación!</p>';
          return;
        }

        // Solo actualizar si hay nuevos mensajes
        const newMessages = messages.filter(msg => msg.id > lastMessageId);
        if (newMessages.length === 0) {
          return; // No hay nuevos mensajes
        }

        // Si es la primera carga, limpiar el contenedor
        if (lastMessageId === 0) {
          chatMessages.innerHTML = '';
        }

        // Agregar nuevos mensajes
        newMessages.forEach(message => {
          const messageDiv = document.createElement('div');
          messageDiv.className = `message ${message.remitente_id == userId ? 'message--sent' : 'message--received'}`;
          messageDiv.dataset.messageId = message.id; // Añadir ID del mensaje para referencia
          messageDiv.innerHTML = `
            <div class="message__content">
              <p>${message.contenido}</p>
              <span class="message__time">${formatDate(message.fecha_envio)}</span>
            </div>
            ${message.remitente_id == userId ? `
              <div class="message__actions">
                <button class="message__action-btn message__edit-btn">Editar</button>
                <button class="message__action-btn message__delete-btn">Borrar</button>
              </div>
              <div class="message__edit-form">
                <input type="text" class="message__edit-input" value="${message.contenido}" />
                <button class="message__edit-submit">Guardar</button>
                <button class="message__edit-cancel">Cancelar</button>
              </div>
            ` : ''}
          `;
```

```php
    $is_friend = $stmt->fetchColumn();

    if ($is_friend == 0) {
      echo json_encode(['status' => 'error', 'message' => 'El usuario no es un amigo']);
      exit();
    }

    // Obtener mensajes
    $query = "SELECT id, remitente_id, destinatario_id, contenido, fecha_envio
              FROM mensajes
              WHERE (remitente_id = :user_id AND destinatario_id = :friend_id)
                 OR (remitente_id = :friend_id AND destinatario_id = :user_id)
              ORDER BY fecha_envio ASC";
    $stmt = $db->prepare($query);
    $stmt->execute([
      ':user_id' => $user_id,
      ':friend_id' => $friend_id
    ]);
    $messages = $stmt->fetchAll(PDO::FETCH_ASSOC);

    echo json_encode(['status' => 'success', 'data' => $messages]);
    exit();

  } elseif ($method === 'POST') {
    // Enviar un nuevo mensaje
    $data = json_decode(file_get_contents('php://input'), true);

    if (!isset($data['friend_id']) || !isset($data['content']) || empty(trim($data['content']))) {
      echo json_encode(['status' => 'error', 'message' => 'Datos incompletos']);
      exit();
    }

    $friend_id = (int)$data['friend_id'];
    $content = trim($data['content']);

    // Verificar que el usuario es un amigo
    $query = "SELECT COUNT(*)
              FROM amistades
              WHERE ((usuario_id = :user_id AND amigo_id = :friend_id)
                     OR (usuario_id = :friend_id AND amigo_id = :user_id))
              AND estado = 'accepted'";
    $stmt = $db->prepare($query);
    $stmt->execute([
      ':user_id' => $user_id,
      ':friend_id' => $friend_id
    ]);
    $is_friend = $stmt->fetchColumn();
```

Notificaciones:

Estructura:

1. Ubicación y Organización: ◦ Frontend: Vista en /notificaciones/notificaciones.php. ◦ Backend: Conexión a la base de datos en ../../database/conexionDB.php. ◦ API: Endpoint en ../../backend/api/notificaciones.php. ◦ Estilos: CSS en /notificaciones/css/style.css, con estilos inline para acciones de notificaciones.
2. Dependencias: ◦ PHP: Manejo de sesiones, consultas con PDO, procesamiento de solicitudes API. ◦ PostgreSQL: Base de datos relacional para almacenar usuarios y notificaciones. ◦ HTML/CSS: Interfaz responsiva con tarjetas de notificaciones y botón para marcar todas como leídas. ◦ JavaScript: Gestión dinámica de notificaciones con fetch, polling cada 5 segundos, y eventos para marcar como leída. ◦ Font Awesome: Iconos (v6.4.0, CDN).

Funciones:

1. Autenticación y Seguridad (notificaciones.php y api/notificaciones.php): ◦ Verificación de sesión con session_start() y $_SESSION['usuario_id']. ◦ Redirección a index.php si no hay sesión o el usuario no existe. ◦ Prevención de XSS con htmlspecialchars() en datos de usuario. ◦ Consultas preparadas con PDO para evitar inyecciones SQL. ◦ Verificación en la API para asegurar que solo el usuario propietario pueda marcar notificaciones como leídas.
2. Interfaz de Notificaciones (notificaciones.php): ◦ Muestra una lista de notificaciones con mensaje, tipo, fecha y estado (leída/no leída). ◦ Botón para marcar todas las notificaciones como leídas, visible solo si hay notificaciones no leídas. ◦ Área de notificaciones dinámica que carga vía API y se actualiza cada 5 segundos. ◦ Opciones para marcar notificaciones individuales como leídas, con botones visibles para notificaciones no leídas. ◦ Navegación responsiva con menú móvil.
3. Gestión de Notificaciones (api/notificaciones.php): ◦ GET: Recupera todas las notificaciones del usuario y el conteo de no leídas, ordenadas por fecha_creacion. ◦ POST (action: mark_as_read): Marca una notificación específica como leída. ◦ POST (action: mark_all_as_read): Marca todas las notificaciones del usuario como leídas. ◦ Respuestas en formato JSON con status (success/error) y mensajes descriptivos.
4. Notificaciones: ◦ Contador de notificaciones no leídas en la barra de navegación (badge en desktop y móvil). ◦ Generación automática de notificaciones en la base de datos para eventos como solicitudes de amistad (tipo: friend_request).
5. Interfaz Dinámica: ◦ Polling cada 5 segundos para cargar nuevas notificaciones sin recargar la página. ◦ Estilos diferenciados para notificaciones leídas (notification-read) y no leídas (notification-unread). ◦ Acciones de marcar como leída visibles solo para notificaciones no leídas. ◦ Formato de fechas en español usando toLocaleString para mejorar la legibilidad.

```php
<?php
header('Content-Type: application/json');

ini_set('session.gc_maxlifetime', 3600);
session_set_cookie_params(3600, '/');


session_start();
require_once "../../database/conexionDB.php";

$response = ['status' => 'error', 'message' => '', 'data' => []];

try {
    if (!isset($_SESSION['usuario_id'])) {
        $response['message'] = 'Sesión no encontrada';
        echo json_encode($response);
        exit();
    }

    $db = conexionDB::getConexion();
    $user_id = $_SESSION['usuario_id'];

    // Manejar solicitudes POST (acciones como marcar como leída)
    if ($_SERVER['REQUEST_METHOD'] === 'POST') {
        $input = json_decode(file_get_contents('php://input'), true);
        $action = $input['action'] ?? '';

        if ($action === 'mark_as_read') {
            $notification_id = $input['notification_id'] ?? 0;
            $query = "UPDATE notificaciones SET leida = TRUE WHERE id = :notification_id AND usuario_id = :user_id";
            $stmt = $db->prepare($query);
            $stmt->execute([
                ':notification_id' => $notification_id,
                ':user_id' => $user_id
            ]);
            $response['status'] = 'success';
            $response['message'] = 'Notificación marcada como leída';
        } elseif ($action === 'mark_all_as_read') {
            $query = "UPDATE notificaciones SET leida = TRUE WHERE usuario_id = :user_id AND leida = FALSE";
            $stmt = $db->prepare($query);
            $stmt->execute([':user_id' => $user_id]);
            $response['status'] = 'success';
            $response['message'] = 'Todas las notificaciones marcadas como leídas';
        } else {
            $response['message'] = 'Acción no válida';
        }

        echo json_encode($response);
```

Ver Perfil:

Estructura:

1. Ubicación y Organización: ◦ Frontend: Vista en /ver_perfil/perfil.php. ◦ Backend: Conexión a la base de datos en ../../database/conexionDB.php. ◦ Estilos: CSS en /ver_perfil/css/style.css y /friends/css/style.css.
2. Dependencias: ◦ PHP: Manejo de sesiones, consultas con PDO. ◦ PostgreSQL: Base de datos relacional para almacenar datos de usuarios,

amistades y notificaciones. ◦ HTML/CSS: Interfaz responsiva con sección de imagen de perfil y campos de información. ◦ JavaScript: Gestión del menú móvil. ◦ Font Awesome: Iconos (v6.4.0, CDN). ◦ Avatares: Imágenes en ../../backend/perfil/ o predeterminadas de pravatar.cc.

Funciones:

1. Autenticación y Seguridad (perfil.php): ◦ Verificación de sesión con session_start() y $_SESSION['usuario_id']. ◦ Redirección a index.php si no hay sesión o el usuario no existe. ◦ Validación de que el user_id proporcionado en $_GET corresponde a un amigo con estado accepted en amistades. ◦ Prevención de XSS con htmlspecialchars() en datos de usuario. ◦ Consultas preparadas con PDO para evitar inyecciones SQL.
2. Interfaz de Perfil (perfil.php): ◦ Muestra la información del perfil de un amigo: nombre, correo, teléfono, profesión, biografía e imagen. ◦ Indicador de estado (en línea/última conexión) basado en ultima_conexion, considerando un usuario en línea si la última conexión fue hace menos de 5 minutos. ◦ Campos con valores predeterminados ("No especificado" o mensaje para biografía vacía) si los datos no están disponibles. ◦ Navegación responsiva con menú móvil. ◦ Botón para volver a la lista de amigos.
3. Gestión de Datos: ◦ Carga los datos del usuario del perfil desde la tabla usuarios, incluyendo nombre_usuario, correo, telefono, profesion, bio, imagen y ultima_conexion. ◦ Contador de notificaciones no leídas en la barra de navegación.
4. Notificaciones: ◦ Muestra el contador de notificaciones no leídas en la barra de navegación (badge en desktop y móvil).
5. Interfaz Dinámica: ◦ Diseño responsivo con una sección para la imagen de perfil y otra para la información, organizadas en filas. ◦ Formato de fechas en español para la última conexión usando DateTime. ◦ Imagen de perfil cargada desde el servidor o pravatar.cc si no está definida, con parámetro de tiempo para evitar caché.

```php
try {
  $db = conexionDB::getConexion();
  $user_id = $_SESSION['usuario_id'];

  // Obtener datos del usuario actual
  $query = "SELECT nombre_usuario, correo FROM usuarios WHERE id = :id";
  $stmt = $db->prepare($query);
  $stmt->execute([':id' => $user_id]);
  $usuario = $stmt->fetch(PDO::FETCH_ASSOC);

  if (!$usuario) {
    error_log("Usuario no encontrado, redirigiendo a index.php");
    header("Location: ../inicio/index.php");
    exit();
  }

  $nombre_usuario = htmlspecialchars($usuario['nombre_usuario']);
  $correo = htmlspecialchars($usuario['correo']);

  // Contar notificaciones no leídas
  $query = "SELECT COUNT(*) FROM notificaciones WHERE usuario_id = :user_id AND leida = FALSE";
  $stmt = $db->prepare($query);
  $stmt->execute([':user_id' => $user_id]);
  $unread_count = $stmt->fetchColumn();

  // Verificar que se haya pasado un user_id y que sea un amigo válido
  if (!isset($_GET['user_id']) || !is_numeric($_GET['user_id'])) {
    error_log("User ID no proporcionado o inválido, redirigiendo a amigos.php");
    header("Location: ../friends/amigos.php");
    exit();
  }

  $profile_user_id = (int)$_GET['user_id'];

  // Verificar que el usuario es un amigo
  $query = "SELECT COUNT(*)
            FROM amistades
            WHERE ((usuario_id = :user_id AND amigo_id = :friend_id)
                   OR (usuario_id = :friend_id AND amigo_id = :user_id))
            AND estado = 'accepted'";
  $stmt = $db->prepare($query);
  $stmt->execute([
    ':user_id' => $user_id,
    ':friend_id' => $profile_user_id
  ]);
  $is_friend = $stmt->fetchColumn();
```

29 de abril de 2025
Implementación de funcionalidades: foro y eventos

el 29 de abril de 2025, se me asignó colaborar con Jhony en el desarrollo de la sección de comunidad de la plataforma "El Rincón de ADSO", específicamente en las funcionalidades de Foro (foro.php) y Eventos (eventos.php). Mi responsabilidad fue implementar la parte del foro, incluyendo su frontend, backend y API, mientras Jhony se encargó de la sección de eventos. El objetivo era crear un espacio interactivo donde los usuarios pudieran participar en discusiones comunitarias mediante la creación de temas y mensajes en el foro. La tarea requirió un enfoque

fullstack, abarcando el diseño de la interfaz, la lógica del servidor y la interacción con la base de datos.

Foro:

Estructura:

1. Ubicación y Organización: ◦ Frontend: Vista en /foro/foro.php. ◦ Backend: Conexión a la base de datos en ../../database/conexionDB.php. ◦ API: Endpoint en ../../backend/api/foro.php. ◦ Estilos: CSS en /foro/css/style.css.
2. Dependencias: ◦ PHP: Manejo de sesiones, consultas con PDO, procesamiento de solicitudes API. ◦ PostgreSQL: Base de datos relacional para almacenar usuarios, temas y mensajes del foro. ◦ HTML/CSS: Interfaz responsiva con formulario para crear temas, lista de temas y sección de mensajes. ◦ JavaScript: Gestión dinámica del foro con fetch, polling cada 5 segundos para mensajes, y eventos para crear temas y responder. ◦ Font Awesome: Iconos (v6.4.0, CDN).

Funciones:

1. Autenticación y Seguridad (foro.php y api/foro.php): ◦ Verificación de sesión con session_start() y $_SESSION['usuario_id']. ◦ Redirección a index.php si no hay sesión o el usuario no existe. ◦ Prevención de XSS con htmlspecialchars() en datos de usuario. ◦ Consultas preparadas con PDO para evitar inyecciones SQL. ◦ Validación en la API para asegurar que los campos requeridos (título, contenido, tema_id) estén presentes al crear temas o mensajes.
2. Interfaz de Foro (foro.php): ◦ Muestra una lista de temas con título, autor y fecha, clickable para ver detalles. ◦ Formulario para crear nuevos temas con título y contenido. ◦ Sección de detalle de tema (oculta inicialmente) con mensajes, título del tema y formulario para responder. ◦ Botón para volver a la lista de temas desde el detalle. ◦ Navegación responsiva con menú móvil.
3. Gestión de Foro (api/foro.php): ◦ GET: Sin topic_id, recupera todos los temas ordenados por fecha_creacion descendente. Con topic_id, obtiene los mensajes de un tema específico ordenados por fecha_creacion ascendente. ◦ POST (action: create_topic): Crea un nuevo tema con título y contenido, asociándolo al usuario. ◦ POST (action: post_message): Publica un mensaje en un tema específico, asociándolo al usuario. ◦ Respuestas en formato JSON con status (success/error) y mensajes descriptivos.
4. Notificaciones: ◦ Contador de notificaciones no leídas en la barra de navegación (badge en desktop y móvil).

5. Interfaz Dinámica: ◦ Carga dinámica de temas y mensajes mediante solicitudes AJAX. ◦ Polling cada 5 segundos para actualizar mensajes en el tema activo sin recargar la página. ◦ Formato de fechas en español usando toLocaleString para mejorar la legibilidad. ◦ Interfaz alternante entre lista de temas y detalle de tema, con estado gestionado mediante clases hidden. ◦ Validación en cliente para evitar envíos vacíos en formularios de tema y mensaje.

```php
<?php
// Establecer configuraciones de la sesión ANTES de iniciarla
ini_set('session.gc_maxlifetime', 3600);
session_set_cookie_params(3600, '/');

// Iniciar la sesión
session_start();
require_once "../../database/conexionDB.php";

if (!isset($_SESSION['usuario_id'])) {
  error_log("Sesión no encontrada, redirigiendo a index.php");
  header("Location: ../inicio/index.php");
  exit();
}

try {
  $db = conexionDB::getConexion();
  $user_id = $_SESSION['usuario_id'];

  // Obtener datos del usuario actual
  $query = "SELECT nombre_usuario, correo FROM usuarios WHERE id = :id";
  $stmt = $db->prepare($query);
  $stmt->execute([':id' => $user_id]);
  $usuario = $stmt->fetch(PDO::FETCH_ASSOC);

  if (!$usuario) {
    error_log("Usuario no encontrado, redirigiendo a index.php");
    header("Location: ../inicio/index.php");
    exit();
  }

  $nombre_usuario = htmlspecialchars($usuario['nombre_usuario']);
  $correo = htmlspecialchars($usuario['correo']);

  // Contar notificaciones no leídas
  $query = "SELECT COUNT(*) FROM notificaciones WHERE usuario_id = :user_id AND leida = FALSE";
  $stmt = $db->prepare($query);
  $stmt->execute([':user_id' => $user_id]);
  $unread_count = $stmt->fetchColumn();

} catch (PDOException $e) {
  error_log("Error de base de datos: " . $e->getMessage());
  header("Location: ../inicio/index.php");
  exit();
}
?>

<!DOCTYPE html>
```

30 de abril de 2025

Funcionalidad de los botones de editar y elimar en los repositorios

Objetivos:

- Implementar la funcionalidad de los botones "Editar" y "Eliminar" en las vistas de recursos (ver_libro.php, ver_video.php, ver_documento.php) y en la sección "Mis Aportes" de panel-usuario.php.
- Utilizar un modal para la edición de recursos, permitiendo al usuario modificar campos como título, descripción, categorías, etiquetas, visibilidad, y archivos (portada o contenido).
- Garantizar que las operaciones de edición y eliminación interactúen correctamente con el backend (por ejemplo, update_resource.php y delete_resource.php) y actualicen la interfaz de usuario.
- Restringir estas acciones al autor del recurso (basado en autor_id coincidente con usuario_id).